FreeBSD

FreeBSD

##: 44974
2014-05-29 # lwhsu.
© 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007 DocEng

##

FreeBSD ####(###FDP, FreeBSD Documentation Project)##############

##################################(############) ###########

################################* ######

Copyright

Redistribution and use in source (XML DocBook) and 'compiled' forms (XML, HTML, PDF, PostScript, RTF and so forth) with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code (XML DocBook) must retain the above copyright notice, this list of conditions and the following disclaimer as the first lines of this file unmodified.

2. Redistributions in compiled form (transformed to other DTDs, converted to PDF, PostScript, RTF and other formats) must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

##

THIS DOCUMENTATION IS PROVIDED BY THE FREEBSD DOCUMENTATION PROJECT "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE FREEBSD DOCUMENTATION PROJECT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

####

..... vii
1. Shell ####(Prompts) vii
2. ######### vii
3. #Note#Tip#Important#Warning#Example#### vii
4. ## viii
1. ## 1
1.1. FreeBSD ####### 1
1.2. #####… 2
1.3. ##### 2
2. ## 5
2.1. #### 5
2.2. #### 6
3. SGML Primer 9
3.1. ## 9
3.2. Elements, tags, and attributes 11
3.3. The DOCTYPE declaration 17
3.4. Escaping back to SGML 20
3.5. ## 20
3.6. Entities 22
3.7. Using entities to include files 25
3.8. Marked sections 28
3.9. Conclusion 33
4. SGML Markup 35
4.1. HTML 35
4.2. DocBook 47
5. * Stylesheets 81
5.1. * DSSSL 81
5.2. CSS 81
6. Structuring documents under `doc/` 83
6.1. The top level, `doc/` 83
6.2. The `lang.encoding/` directories 83
6.3. Document specific information 84
7. The Documentation Build Process 87
7.1. The FreeBSD Documentation Build Toolset 87
7.2. Understanding Makefiles in the Documentation tree 87
7.3. FreeBSD Documentation Project make includes 90
8. ## Website 95
8.1. #### 95
8.2. Build the web pages from scratch 95
8.3. ############## 96
8.4. #### 96
9. ######## 97
10. ####### 101
10.1. Style guide 102

10.2. ### 105
11. Using sgml-mode with Emacs 107
12. #### 109
12.1. The FreeBSD Documentation Project 109
12.2. SGML 109
12.3. HTML 109
12.4. DocBook 109
12.5. The Linux Documentation Project 109
A. ## 111
A.1. DocBook book 111
A.2. DocBook article 112
A.3. Producing formatted output 113
..... 117

####

1. ###### viii
3.1. Using an element (start and end tags) 12
3.2. Using an element (start tag only) 12
3.3. Elements within elements; em 12
3.4. Using an element with an attribute 13
3.5. Single quotes around attributes 14
3.6. .profile, for sh(1) and bash(1) users 14
3.7. .cshrc, for csh(1) and tcsh(1) users 15
3.8. SGML generic comment 21
3.9. Erroneous SGML comments 21
3.10. Defining general entities 23
3.11. Defining parameter entities 23
3.12. Using general entities to include files 25
3.13. Using parameter entities to include files 26
3.14. Structure of a marked section 29
3.15. Using a CDATA marked section 30
3.16. Using INCLUDE and IGNORE in marked sections 31
3.17. Using a parameter entity to control a marked section 31
4.1. Normal HTML document structure 36
4.2. h1, h2, etc. 37
4.3. Bad ordering of h*n* elements 37
4.4. p 38
4.5. blockquote 38
4.6. ul and ol 39
4.7. Definition lists with dl 39
4.8. pre 40
4.9. Simple use of table 41
4.10. Using rowspan 42
4.11. Using colspan 42
4.12. Using rowspan and colspan together 43
4.13. em and strong 43
4.14. b and i 44
4.15. tt 44
4.16. big, small, and font 45
4.17. Using <a href="..."> 45
4.18. Using <a name="..."> 46
4.19. Linking to a named part of another document 46
4.20. Linking to a named part of the same document 46
4.21. Boilerplate book with bookinfo 49
4.22. Boilerplate article with articleinfo 49
4.23. A simple chapter 50
4.24. Empty chapters 51
4.25. Sections in chapters 51
4.26. para 53

4.27. blockquote 53
4.28. warning 54
4.29. itemizedlist , orderedlist , and procedure 55
4.30. programlisting 57
4.31. co and calloutlist 57
4.32. informaltable 59
4.33. Tables where frame="none" 60
4.34. screen , prompt , and userinput 61
4.35. emphasis 62
4.36. Quotations 62
4.37. Keys, mouse buttons, and combinations 63
4.38. Applications, commands, and options. 64
4.39. filename 65
4.40. filename tag with package role 66
4.41. devicename 67
4.42. hostid and roles 68
4.43. username 70
4.44. maketarget and makevar 70
4.45. literal 71
4.46. replaceable 72
4.47. errorname 73
4.48. id on chapters and sections 77
4.49. anchor 77
4.50. Using xref 78
4.51. Using link 79
4.52. ulink 80
A.1. DocBook book 111
A.2. DocBook article 112
A.3. ## DocBook # HTML (####) 113
A.4. ## DocBook # HTML (####) 113
A.5. ## DocBook # Postscript(PS) ## 114
A.6. ## DocBook # PDF ## 115

##

1. Shell ####(Prompts)

########## root ####################(prompt) ##############

##	####(Prompt)
####	%
root	#

2.

################

####	##
##	## ls -a #########
##	## .login ##
#########	You have mail.
##################	% **su** Password:
########(manual)	# su(1) ######
#####(user)###(group)######...	## root ########
#####	###########
###########	########## rm ######
######	$HOME ############

3. #Note#Tip#Important#Warning#Example####

###################################

##

##

##

####################

##

##################################

##

##############
###########...#

1.

##

4.

Sue Blake, Patrick Durusau, Jon Hamilton, Peter Flynn, Christopher Maden ######

1.

FreeBSD ############### FreeBSD ########## # FreeBSD ####(#### FDP ### FreeBSD Documentation Project ###)

###################FDP ################## FDP## ################

############## FDP ###FDP ################### ########## FreeBSD documentation project #### #

##########

- ######### FDP #####
- #### FDP #### SGML #####
- ############
- ###################### FreeBSD ####

1.1. FreeBSD

FDP #### FreeBSD # 4 #######

####(manual)
: ###### manual #### FDP ############ base system #### ###FDP ##(######)####### ########################

  ######################## ###### FDP ###

FAQ
: FAQ ########## newsgroup ############ FreeBSD ####### # (#############) #### ####################

####(Handbook)
: ######## FreeBSD ###############

Web site
: FreeBSD ######### WWW ####### http://www.FreeBSD.org/ ###### mirror ###### ######## FreeBSD ####

############# FreeBSD CVS tree ############################## ############ CSup, CVSup # CTM ######(checkout)###################

################## FreeBSD ######(#######)########## FreeBSD ## CVS repository ## ################# FreeBSD repository ####### ###FDP #############

1.2. #####...

##########

- ### FreeBSD CVS repository ######## FreeBSD ####(# CVS # CSup # CVSup ## CTM) # ## CVSup ### *checked-out* ###
- ### FreeBSD Ports ####### pkg_add(1) #########

1.3.

##############################

1. ## textproc/docproj ##### port(meta-port)#

   ```
   # cd /usr/ports/textproc/docproj
   # make JADETEX=no install
   ```

2. ## FreeBSD doc tree ##### #### CSup # CVSup # checkout ### ####### CVS repository ########

   ############# CVS repository ######## checkout # doc/share ## doc/en_US.ISO8859-1/share ########

   ```
   % cvs checkout doc/share
   % cvs checkout doc/en_US.ISO8859-1/share
   ```

   ###################### doc # check out ###

   ```
   % cvs checkout doc
   ```

3. ##### repository # checkout ################## ############################ ##

   ################### FreeBSD # Windows 2000 #### VPN ### ###############

   1. Check out articles ###

      ```
      % cvs checkout doc/en_US.ISO8859-1/articles
      ```

   2. ############################## vpn-w2k #####

      ```
      % cd doc/en_US.ISO8859-1/articles
      % cp -R committers-guide vpn-w2k
      ```

   ############ FAQ(## doc/en_US.ISO8859-1/books/faq) ##### repository #### (check out)#

   ```
   % cvs checkout doc/en_US.ISO8859-1/books/faq
   ```

4. ####### .xml ##

5. # lint ######################## ##################################

```
% make lint
```

################# FORMATS ############### ########## html, html-split , txt, ps, pdf, rtf # ############### doc/share/mk/doc.docbook.mk ## #### ######## #################(quotes)##########

########## html ##########

```
% make FORMATS=html
```

html # txt ##### ####### make(1)

```
% make FORMATS=html
% make FORMATS=txt
```

###############

```
% make FORMATS="html txt"
```

6. #### send-pr(1) #########

2.

FDP ########## FreeBSD ########## ####### FDP #################

######## Ports # Packages ###############

########################## ##################

textproc/docproj

textproc/docproj ########## #### port(meta-port)########## ############## #### port ########################## ###### ####### chinese/docproj

packages ########## JadeTeX ## macro ### ####### macro # ######## TeX### TeX ######### ####### Postscript # PDF #######

########################## JadeTeX (## TeX) ############

```
# make JADETEX=yes install
```

########

```
# make JADETEX=no install
```

######## textproc/docproj-jadetex ## textproc/docproj-nojadetex ######## ######### JADETEX ### slave ports# ##### docproj ##### ### JadeTeX ### ######### HTML # ASCII ########## JadeTeX# ## ### PostScript#PDF ####### TeX ###

2.1.

2.1.1.

####### FreeeBSD ################# ########## HTML#plain text## RTF ########## textproc/docproj #########

Jade (textproc/jade)
: DSSSL ##################(marked up)##########HTML # TeX#

Tidy (www/tidy)
: HTML “pretty printer”########## HTML #################

Links (www/links)
: ####### WWW ###(browser)### HTML ### plain text ###

peps (graphics/peps)
: ######### EPS ########### PNG ### ###############

2.1.2. DTD # Entity

FDP ##### DTD # Entity################

HTML DTD (textproc/html)
: HTML ### WWW ######### FreeBSD #########

DocBook DTD (textproc/docbook)
: DocBook ################### FreeBSD ####### DocBook #####

ISO 8879 entities (textproc/iso8879)
: # ISO 8879:1986 ### 19 # entity ### DTD ###### ##############(##########)#######

2.1.3. ###(Stylesheets)

############################

Modular DocBook ### (textproc/dsssl-docbook-modular)
: Modular DocBook ######## DocBook ################## HTML # RTF#

2.2.

################################ ##############

2.2.1.

JadeTeX # teTeX (print/jadetex # print/teTeX)
: Jade # teTeX #### DocBook ###### DVI, Postscript # PDF ########### JadeTeX ## macro################

  ##############(##### HTML, plain text, RTF ########) ####### JadeTeX # teTeX# # ################### ## teTeX ##### 30MB ###

##

JadeTeX ## teTeX ######## JadeTeX ### ##### teTeX # ## `print/jadetex/pkg-message`

2.

Emacs # XEmacs (editors/emacs # editors/xemacs)

########## SGML DTD ######### ##################################

################################## ##################################

########## SGML ######### Documentation Engineering Team <doceng@FreeBSD.org > ### ##################

3. SGML Primer

FDP ####### SGML ############ SGML #### ######### SGML ############### SGML ###

########## Mark Galassi ### Get Going With DocBook#

3.1.

Way back when, electronic text was simple to deal with. Admittedly, you had to know which character set your document was written in (ASCII, EBCDIC, or one of a number of others) but that was about it. Text was text, and what you saw really was what you got. No frills, no formatting, no intelligence.

Inevitably, this was not enough. Once you have text in a machine-usable format, you expect machines to be able to use it and manipulate it intelligently. You would like to indicate that certain phrases should be emphasized, or added to a glossary, or be hyperlinks. You might want filenames to be shown in a “typewriter” style font for viewing on screen, but as “italics” when printed, or any of a myriad of other options for presentation.

It was once hoped that Artificial Intelligence (AI) would make this easy. Your computer would read in the document and automatically identify key phrases, filenames, text that the reader should type in, examples, and more. Unfortunately, real life has not happened quite like that, and our computers require some assistance before they can meaningfully process our text.

More precisely, they need help identifying what is what. You or I can look at

> To remove `/tmp/foo` use rm(1).
>
> ```
> % rm /tmp/foo
> ```

and easily see which parts are filenames, which are commands to be typed in, which parts are references to manual pages, and so on. But the computer processing the document cannot. For this we need markup.

“Markup” is commonly used to describe “adding value” or “increasing cost”. The term takes on both these meanings when applied to text. Markup is additional text included in the document, distinguished from the document's content in some way, so that programs that process the document can read the markup and use it when making decisions about the document. Editors can hide the markup from the user, so the user is not distracted by it.

The extra information stored in the markup *adds value* to the document. Adding the markup to the document must typically be done by a person——after all, if computers

could recognize the text sufficiently well to add the markup then there would be no need to add it in the first place. This *increases the cost* (i.e., the effort required) to create the document.

The previous example is actually represented in this document like this:

```
<para>To remove <filename>/tmp/foo</filename> use &man.rm.1;.</para>

<screen>&prompt.user; <userinput>rm /tmp/foo</userinput></screen>
```

As you can see, the markup is clearly separate from the content.

Obviously, if you are going to use markup you need to define what your markup means, and how it should be interpreted. You will need a markup language that you can follow when marking up your documents.

Of course, one markup language might not be enough. A markup language for technical documentation has very different requirements than a markup language that was to be used for cookery recipes. This, in turn, would be very different from a markup language used to describe poetry. What you really need is a first language that you use to write these other markup languages. A *meta markup language*.

This is exactly what the Standard Generalized Markup Language (SGML) is. Many markup languages have been written in SGML, including the two most used by the FDP, HTML and DocBook.

Each language definition is more properly called a Document Type Definition (DTD). The DTD specifies the name of the elements that can be used, what order they appear in (and whether some markup can be used inside other markup) and related information. A DTD is sometimes referred to as an *application* of SGML.

A DTD is a *complete* specification of all the elements that are allowed to appear, the order in which they should appear, which elements are mandatory, which are optional, and so forth. This makes it possible to write an SGML *parser* which reads in both the DTD and a document which claims to conform to the DTD. The parser can then confirm whether or not all the elements required by the DTD are in the document in the right order, and whether there are any errors in the markup. This is normally referred to as "validating the document".

##

This processing simply confirms that the choice of elements, their ordering, and so on, conforms to that listed in the DTD. It does *not* check that you have used *appropriate* markup for the content. If you

> tried to mark up all the filenames in your document as function names, the parser would not flag this as an error (assuming, of course, that your DTD defines elements for filenames and functions, and that they are allowed to appear in the same place).

It is likely that most of your contributions to the Documentation Project will consist of content marked up in either HTML or DocBook, rather than alterations to the DTDs. For this reason this book will not touch on how to write a DTD.

3.2. Elements, tags, and attributes

All the DTDs written in SGML share certain characteristics. This is hardly surprising, as the philosophy behind SGML will inevitably show through. One of the most obvious manifestations of this philosophy is that of *content* and *elements*.

Your documentation (whether it is a single web page, or a lengthy book) is considered to consist of content. This content is then divided (and further subdivided) into elements. The purpose of adding markup is to name and identify the boundaries of these elements for further processing.

For example, consider a typical book. At the very top level, the book is itself an element. This "book" element obviously contains chapters, which can be considered to be elements in their own right. Each chapter will contain more elements, such as paragraphs, quotations, and footnotes. Each paragraph might contain further elements, identifying content that was direct speech, or the name of a character in the story.

You might like to think of this as "chunking" content. At the very top level you have one chunk, the book. Look a little deeper, and you have more chunks, the individual chapters. These are chunked further into paragraphs, footnotes, character names, and so on.

Notice how you can make this differentiation between different elements of the content without resorting to any SGML terms. It really is surprisingly straightforward. You could do this with a highlighter pen and a printout of the book, using different colors to indicate different chunks of content.

Of course, we do not have an electronic highlighter pen, so we need some other way of indicating which element each piece of content belongs to. In languages written in SGML (HTML, DocBook, et al) this is done by means of *tags*.

A tag is used to identify where a particular element starts, and where the element ends. *The tag is not part of the element itself.* Because each DTD was normally written to mark up specific types of information, each one will recognize different elements, and will therefore have different names for the tags.

For an element called *`element-name`* the start tag will normally look like `<element-name>`. The corresponding closing tag for this element is `</element-name>` .

3.1. Using an element (start and end tags)

HTML has an element for indicating that the content enclosed by the element is a paragraph, called `p`. This element has both start and end tags.

```
<p>This is a paragraph.  It starts with the start tag for
  the 'p' element, and it will end with the end tag for the 'p'
  element.</p>

<p>This is another paragraph.  But this one is much shorter.</
p>
```

Not all elements require an end tag. Some elements have no content. For example, in HTML you can indicate that you want a horizontal line to appear in the document. Obviously, this line has no content, so just the start tag is required for this element.

3.2. Using an element (start tag only)

HTML has an element for indicating a horizontal rule, called `hr`. This element does not wrap content, so only has a start tag.

```
<p>This is a paragraph.</p>

<hr>

<p>This is another paragraph.  A horizontal rule separates this
  from the previous paragraph.</p>
```

If it is not obvious by now, elements can contain other elements. In the book example earlier, the book element contained all the chapter elements, which in turn contained all the paragraph elements, and so on.

3.3. Elements within elements; `em`

```
<p>This is a simple <em>paragraph</em> where some
```

```
  of the <em>words</em> have been <em>emphasized</em>.</p>
```

The DTD will specify the rules detailing which elements can contain other elements, and exactly what they can contain.

##

People often confuse the terms tags and elements, and use the terms as if they were interchangeable. They are not.

An element is a conceptual part of your document. An element has a defined start and end. The tags mark where the element starts and end.

When this document (or anyone else knowledgeable about SGML) refers to “the <p> tag” they mean the literal text consisting of the three characters <, p, and >. But the phrase “the <p> element” refers to the whole element.

This distinction *is* very subtle. But keep it in mind.

Elements can have attributes. An attribute has a name and a value, and is used for adding extra information to the element. This might be information that indicates how the content should be rendered, or might be something that uniquely identifies that occurrence of the element, or it might be something else.

An element's attributes are written *inside* the start tag for that element, and take the form `attribute-name="attribute-value"` .

In sufficiently recent versions of HTML, the `p` element has an attribute called `align`, which suggests an alignment (justification) for the paragraph to the program displaying the HTML.

The `align` attribute can take one of four defined values, `left`, `center`, `right` and `justify`. If the attribute is not specified then the default is `left`.

3.4. Using an element with an attribute

```
<p align="left">The inclusion of the align attribute
  on this paragraph was superfluous, since the default is ↺
left.</p>
```

```
<p align="center">This may appear in the center.</p>
```

Some attributes will only take specific values, such as `left` or `justify`. Others will allow you to enter anything you want. If you need to include quotes (") within an attribute then use single quotes around the attribute value.

3.5. Single quotes around attributes

```
<p align='right'>I am on the right!</p>
```

Sometimes you do not need to use quotes around attribute values at all. However, the rules for doing this are subtle, and it is far simpler just to *always* quote your attribute values.

The information on attributes, elements, and tags is stored in SGML catalogs. The various Documentation Project tools use these catalog files to validate your work. The tools in textproc/docproj include a variety of SGML catalog files. The FreeBSD Documentation Project includes its own set of catalog files. Your tools need to know about both sorts of catalog files.

3.2.1. For you to do...

In order to run the examples in this document you will need to install some software on your system and ensure that an environment variable is set correctly.

1. Download and install textproc/docproj from the FreeBSD ports system. This is a *meta-port* that should download and install all of the programs and supporting files that are used by the Documentation Project.

2. Add lines to your shell startup files to set `SGML_CATALOG_FILES` . (If you are not working on the English version of the documentation, you will want to substitute the correct directory for your language.)

3.6. **`.profile`**, for sh(1) and bash(1) users

```
SGML_ROOT=/usr/local/share/xml
SGML_CATALOG_FILES=${SGML_ROOT}/jade/catalog
SGML_CATALOG_FILES=${SGML_ROOT}/iso8879/catalog:
$SGML_CATALOG_FILES
```

```
SGML_CATALOG_FILES=${SGML_ROOT}/html/catalog:
$SGML_CATALOG_FILES
SGML_CATALOG_FILES=${SGML_ROOT}/docbook/4.1/catalog:
$SGML_CATALOG_FILES
SGML_CATALOG_FILES=/usr/doc/share/xml/catalog:
$SGML_CATALOG_FILES
SGML_CATALOG_FILES=/usr/doc/en_US.ISO8859-1/share/xml/
catalog:$SGML_CATALOG_FILES
export SGML_CATALOG_FILES
```

3.7. **`.cshrc`**, for csh(1) and tcsh(1) users

```
setenv SGML_ROOT /usr/local/share/xml
setenv SGML_CATALOG_FILES ${SGML_ROOT}/jade/catalog
setenv SGML_CATALOG_FILES ${SGML_ROOT}/iso8879/catalog:
$SGML_CATALOG_FILES
setenv SGML_CATALOG_FILES ${SGML_ROOT}/html/catalog:
$SGML_CATALOG_FILES
setenv SGML_CATALOG_FILES ${SGML_ROOT}/docbook/4.1/catalog:
$SGML_CATALOG_FILES
setenv SGML_CATALOG_FILES /usr/doc/share/xml/catalog:
$SGML_CATALOG_FILES
setenv SGML_CATALOG_FILES /usr/doc/en_US.ISO8859-1/share/
xml/catalog:$SGML_CATALOG_FILES
```

Then either log out, and log back in again, or run those commands from the command line to set the variable values.

1. Create `example.xml` , and enter the following text:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
  <head>
    <title>An example HTML file</title>
  </head>

  <body>
    <p>This is a paragraph containing some text.</p>

    <p>This paragraph contains some more text.</p>

    <p align="right">This paragraph might be right-justified.</p>
  </body>
</html>
```

2. Try to validate this file using an SGML parser.

   Part of textproc/docproj is the nsgmls validating parser [10]. Normally nsgmls reads in a document marked up according to an SGML DTD and returns a copy of the document's Element Structure Information Set (ESIS, but that is not important right now).

   However, when `nsgmls` is given the `-s` parameter, `nsgmls` will suppress its normal output, and just print error messages. This makes it a useful way to check to see if your document is valid or not.

   Use `nsgmls` to check that your document is valid:

   ```
   % nsgmls -s example.xml
   ```

   As you will see, `nsgmls` returns without displaying any output. This means that your document validated successfully.

3. See what happens when required elements are omitted. Try removing the `title` and `/title` tags, and re-run the validation.

   ```
   % nsgmls -s example.xml
   nsgmls:example.xml:5:4:E: character data is not allowed here
   nsgmls:example.xml:6:8:E: end tag for "HEAD" which is not finished
   ```

   The error output from `nsgmls` is organized into colon-separated groups, or columns.

Column	Meaning
1	The name of the program generating the error. This will always be `nsgmls`.
2	The name of the file that contains the error.
3	Line number where the error appears.
4	Column number where the error appears.
5	A one letter code indicating the nature of the message. `I` indicates an informational message, `W` is for warnings, and `E` is for errors[a], and `X` is for cross-references. As you can see, these messages are errors.
6	The text of the error message.

[a] It is not always the fifth column either. `nsgmls -sv` displays `nsgmls:I: SP version "1.3"` (depending on the installed version). As you can see, this is an informational message.

   Simply omitting the `title` tags has generated 2 different errors.

The first error indicates that content (in this case, characters, rather than the start tag for an element) has occurred where the SGML parser was expecting something else. In this case, the parser was expecting to see one of the start tags for elements that are valid inside `head` (such as `title`).

The second error is because `head` elements *must* contain a `title` element. Because it does not `nsgmls` considers that the element has not been properly finished. However, the closing tag indicates that the element has been closed before it has been finished.

4. Put the `title` element back in.

3.3. The DOCTYPE declaration

The beginning of each document that you write must specify the name of the DTD that the document conforms to. This is so that SGML parsers can determine the DTD and ensure that the document does conform to it.

This information is generally expressed on one line, in the DOCTYPE declaration.

A typical declaration for a document written to conform with version 4.0 of the HTML DTD looks like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN">
```

That line contains a number of different components.

`<!`
: Is the *indicator* that indicates that this is an SGML declaration. This line is declaring the document type.

`DOCTYPE`
: Shows that this is an SGML declaration for the document type.

`html`
: Names the first element that will appear in the document.

`PUBLIC "-//W3C//DTD HTML 4.0//EN"`
: Lists the Formal Public Identifier (FPI) for the DTD that this document conforms to. Your SGML parser will use this to find the correct DTD when processing this document.

  `PUBLIC` is not a part of the FPI, but indicates to the SGML processor how to find the DTD referenced in the FPI. Other ways of telling the SGML parser how to find the DTD are shown later.

>
Returns to the document.

3.3.1. Formal Public Identifiers (FPIs)

##

You do not need to know this, but it is useful background, and might help you debug problems when your SGML processor can not locate the DTD you are using.

FPIs must follow a specific syntax. This syntax is as follows:

```
"Owner//Keyword Description//Language"
```

Owner
This indicates the owner of the FPI.

If this string starts with "ISO" then this is an ISO owned FPI. For example, the FPI `"ISO 8879:1986//ENTITIES Greek Symbols//EN"` lists `ISO 8879:1986` as being the owner for the set of entities for Greek symbols. ISO 8879:1986 is the ISO number for the SGML standard.

Otherwise, this string will either look like `-//Owner` or `+//Owner` (notice the only difference is the leading + or -).

If the string starts with - then the owner information is unregistered, with a + it identifies it as being registered.

ISO 9070:1991 defines how registered names are generated; it might be derived from the number of an ISO publication, an ISBN code, or an organization code assigned according to ISO 6523. In addition, a registration authority could be created in order to assign registered names. The ISO council delegated this to the American National Standards Institute (ANSI).

Because the FreeBSD Project has not been registered the owner string is `-//FreeBSD`. And as you can see, the W3C are not a registered owner either.

Keyword
There are several keywords that indicate the type of information in the file. Some of the most common keywords are `DTD`, `ELEMENT`, `ENTITIES`, and `TEXT`. `DTD` is used only for DTD files, `ELEMENT` is usually used for DTD fragments that contain only entity or element declarations. `TEXT` is used for SGML content (text and tags).

Description
: Any description you want to supply for the contents of this file. This may include version numbers or any short text that is meaningful to you and unique for the SGML system.

Language
: This is an ISO two-character code that identifies the native language for the file. `EN` is used for English.

3.3.1.1. `catalog` files

If you use the syntax above and process this document using an SGML processor, the processor will need to have some way of turning the FPI into the name of the file on your computer that contains the DTD.

In order to do this it can use a catalog file. A catalog file (typically called `catalog`) contains lines that map FPIs to filenames. For example, if the catalog file contained the line:

```
PUBLIC "-//W3C//DTD HTML 4.0//EN"              "4.0/strict.dtd"
```

The SGML processor would know to look up the DTD from `strict.dtd` in the `4.0` subdirectory of whichever directory held the `catalog` file that contained that line.

Look at the contents of `/usr/local/share/xml/html/catalog`. This is the catalog file for the HTML DTDs that will have been installed as part of the textproc/docproj port.

3.3.1.2. `SGML_CATALOG_FILES`

In order to locate a `catalog` file, your SGML processor will need to know where to look. Many of them feature command line parameters for specifying the path to one or more catalogs.

In addition, you can set `SGML_CATALOG_FILES` to point to the files. This environment variable should consist of a colon-separated list of catalog files (including their full path).

Typically, you will want to include the following files:

- `/usr/local/share/xml/docbook/4.1/catalog`
- `/usr/local/share/xml/html/catalog`
- `/usr/local/share/xml/iso8879/catalog`
- `/usr/local/share/xml/jade/catalog`

You should already have done this.

3.3.2. Alternatives to FPIs

Instead of using an FPI to indicate the DTD that the document conforms to (and therefore, which file on the system contains the DTD) you can explicitly specify the name of the file.

The syntax for this is slightly different:

```
<!DOCTYPE html SYSTEM "/path/to/file.dtd">
```

The `SYSTEM` keyword indicates that the SGML processor should locate the DTD in a system specific fashion. This typically (but not always) means the DTD will be provided as a filename.

Using FPIs is preferred for reasons of portability. You do not want to have to ship a copy of the DTD around with your document, and if you used the `SYSTEM` identifier then everyone would need to keep their DTDs in the same place.

3.4. Escaping back to SGML

Earlier in this primer I said that SGML is only used when writing a DTD. This is not strictly true. There is certain SGML syntax that you will want to be able to use within your documents. For example, comments can be included in your document, and will be ignored by the parser. Comments are entered using SGML syntax. Other uses for SGML syntax in your document will be shown later too.

Obviously, you need some way of indicating to the SGML processor that the following content is not elements within the document, but is SGML that the parser should act upon.

These sections are marked by `<! ... >` in your document. Everything between these delimiters is SGML syntax as you might find within a DTD.

As you may just have realized, the DOCTYPE declaration is an example of SGML syntax that you need to include in your document...

3.5.

Comments are an SGML construction, and are normally only valid inside a DTD. However, as # 3.4, “Escaping back to SGML” shows, it is possible to use SGML syntax within your document.

The delimiter for SGML comments is the string “`--`”. The first occurrence of this string opens a comment, and the second closes it.

3.8. SGML generic comment

```
<!-- #### -->
```

```
<!-- #### -->

<!-- #####    -->

<!-- #########
     ######### -->

<!-- #######   --
  -- ####### -->
```

If you have used HTML before you may have been shown different rules for comments. In particular, you may think that the string `<!--` opens a comment, and it is only closed by `-->`.

This is *not* the case. A lot of web browsers have broken HTML parsers, and will accept that as valid. However, the SGML parsers used by the Documentation Project are much stricter, and will reject documents that make that error.

3.9. Erroneous SGML comments

```
<!-- This is in the comment --

     THIS IS OUTSIDE THE COMMENT!

  -- back inside the comment -->
```

The SGML parser will treat this as though it were actually:

```
<!THIS IS OUTSIDE THE COMMENT>
```

This is not valid SGML, and may give confusing error messages.

```
<!----- This is a very bad idea ----->
```

As the example suggests, *do not* write comments like that.

```
<!--================================================-->
```

That is a (slightly) better approach, but it still potentially confusing to people new to SGML.

3.5.1. For you to do...

1. Add some comments to `example.xml` , and check that the file still validates using `nsgmls`.

2. Add some invalid comments to `example.xml` , and see the error messages that `nsgmls` gives when it encounters an invalid comment.

3.6. Entities

Entities are a mechanism for assigning names to chunks of content. As an SGML parser processes your document, any entities it finds are replaced by the content of the entity.

This is a good way to have re-usable, easily changeable chunks of content in your SGML documents. It is also the only way to include one marked up file inside another using SGML.

There are two types of entities which can be used in two different situations; *general entities* and *parameter entities*.

3.6.1. General Entities

You cannot use general entities in an SGML context (although you define them in one). They can only be used in your document. Contrast this with parameter entities.

Each general entity has a name. When you want to reference a general entity (and therefore include whatever text it represents in your document), you write `&entity-name;`. For example, suppose you had an entity called `current.version` which expanded to the current version number of your product. You could write:

```
<para>The current version of our product is
  &current.version;.</para>
```

When the version number changes you can simply change the definition of the value of the general entity and reprocess your document.

You can also use general entities to enter characters that you could not otherwise include in an SGML document. For example, < and & cannot normally appear in an SGML document. When the SGML parser sees the < symbol it assumes that a tag (either a start

tag or an end tag) is about to appear, and when it sees the & symbol it assumes the next text will be the name of an entity.

Fortunately, you can use the two general entities `<` and `&` whenever you need to include one or other of these.

A general entity can only be defined within an SGML context. Typically, this is done immediately after the DOCTYPE declaration.

3.10. Defining general entities

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY current.version    "3.0-RELEASE">
<!ENTITY last.version       "2.2.7-RELEASE">
]>
```

Notice how the DOCTYPE declaration has been extended by adding a square bracket at the end of the first line. The two entities are then defined over the next two lines, before the square bracket is closed, and then the DOCTYPE declaration is closed.

The square brackets are necessary to indicate that we are extending the DTD indicated by the DOCTYPE declaration.

3.6.2. Parameter entities

Like general entities, parameter entities are used to assign names to reusable chunks of text. However, where as general entities can only be used within your document, parameter entities can only be used within an SGML context.

Parameter entities are defined in a similar way to general entities. However, instead of using `&entity-name;` to refer to them, use `%entity-name;` [1]. The definition also includes the `%` between the `ENTITY` keyword and the name of the entity.

3.11. Defining parameter entities

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY % param.some "some">
```

[1] Parameter entities use the *Percent* symbol.

```
<!ENTITY % param.text "text">
<!ENTITY % param.new  "%param.some more %param.text">
]>
```

This may not seem particularly useful. It will be.

3.6.3. For you to do...

1. Add a general entity to `example.xml` .

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" [
<!ENTITY version "1.1">
]>

<html>
  <head>
    <title>An example HTML file</title>
  </head>

  <body>
    <p>This is a paragraph containing some text.</p>

    <p>This paragraph contains some more text.</p>

    <p align="right">This paragraph might be right-justified.</p>

    <p>The current version of this document is: &version;</p>
  </body>
</html>
```

2. Validate the document using `nsgmls` .

3. Load `example.xml` into your web browser (you may need to copy it to `example.html` before your browser recognizes it as an HTML document).

   Unless your browser is very advanced, you will not see the entity reference `&version;` replaced with the version number. Most web browsers have very simplistic parsers which do not handle proper SGML[2].

4. The solution is to *normalize* your document using an SGML normalizer. The normalizer reads in valid SGML and outputs equally valid SGML which has been transformed in some way. One of the ways in which the normalizer transforms the SGML is to expand all the entity references in the document, replacing the entities with the text that they represent.

[2]This is a shame. Imagine all the problems and hacks (such as Server Side Includes) that could be avoided if they did.

You can use `sgmlnorm` to do this.

```
% sgmlnorm example.xml > example.html
```

You should find a normalized (i.e., entity references expanded) copy of your document in `example.html`, ready to load into your web browser.

5. If you look at the output from `sgmlnorm` you will see that it does not include a DOCTYPE declaration at the start. To include this you need to use the `-d` option:

```
% sgmlnorm -d example.xml > example.html
```

3.7. Using entities to include files

Entities (both general and parameter) are particularly useful when used to include one file inside another.

3.7.1. Using general entities to include files

Suppose you have some content for an SGML book organized into files, one file per chapter, called `chapter1.xml`, `chapter2.xml`, and so forth, with a `book.xml` file that will contain these chapters.

In order to use the contents of these files as the values for your entities, you declare them with the `SYSTEM` keyword. This directs the SGML parser to use the contents of the named file as the value of the entity.

3.12. Using general entities to include files

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY chapter.1 SYSTEM "chapter1.xml">
<!ENTITY chapter.2 SYSTEM "chapter2.xml">
<!ENTITY chapter.3 SYSTEM "chapter3.xml">
]>

<html>

  &chapter.1;
  &chapter.2;
  &chapter.3;
</html>
```

##

When using general entities to include other files within a document, the files being included (`chapter1.xml` , `chapter2.xml` , and so on) *must not* start with a DOCTYPE declaration. This is a syntax error.

3.7.2. Using parameter entities to include files

Recall that parameter entities can only be used inside an SGML context. Why then would you want to include a file within an SGML context?

You can use this to ensure that you can reuse your general entities.

Suppose that you had many chapters in your document, and you reused these chapters in two different books, each book organizing the chapters in a different fashion.

You could list the entities at the top of each book, but this quickly becomes cumbersome to manage.

Instead, place the general entity definitions inside one file, and use a parameter entity to include that file within your document.

3.13. Using parameter entities to include files

First, place your entity definitions in a separate file, called `chapters.ent` . This file contains the following:

```
<!ENTITY chapter.1 SYSTEM "chapter1.xml">
<!ENTITY chapter.2 SYSTEM "chapter2.xml">
<!ENTITY chapter.3 SYSTEM "chapter3.xml">
```

Now create a parameter entity to refer to the contents of the file. Then use the parameter entity to load the file into the document, which will then make all the general entities available for use. Then use the general entities as before:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY % chapters SYSTEM "chapters.ent">
%chapters;
]>
```

```
<html>
  &chapter.1;
  &chapter.2;
  &chapter.3;
</html>
```

3.7.3. For you to do...

3.7.3.1. Use general entities to include files

1. Create three files, `para1.xml` , `para2.xml` , and `para3.xml` .

   Put content similar to the following in each file:

   ```
   <p>This is the first paragraph.</p>
   ```

2. Edit `example.xml` so that it looks like this:

   ```
   <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
   <!ENTITY version "1.1">
   <!ENTITY para1 SYSTEM "para1.xml">
   <!ENTITY para2 SYSTEM "para2.xml">
   <!ENTITY para3 SYSTEM "para3.xml">
   ]>

   <html>
     <head>
       <title>An example HTML file</title>
     </head>

     <body>
       <p>The current version of this document is: &version;</p>

       &para1;
       &para2;
       &para3;
     </body>
   </html>
   ```

3. Produce `example.html` by normalizing `example.xml` .

   ```
   % sgmlnorm -d example.xml > example.html
   ```

4. Load `example.html` into your web browser, and confirm that the `paran.xml` files have been included in `example.html` .

3.7.3.2. Use parameter entities to include files

##

You must have taken the previous steps first.

1. Edit `example.xml` so that it looks like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY % entities SYSTEM "entities.xml"> %entities;
]>

<html>
  <head>
    <title>An example HTML file</title>
  </head>

  <body>
    <p>The current version of this document is: &version;</p>

    &para1;
    &para2;
    &para3;
  </body>
</html>
```

2. Create a new file, `entities.xml` , with this content:

```
<!ENTITY version "1.1">
<!ENTITY para1 SYSTEM "para1.xml">
<!ENTITY para2 SYSTEM "para2.xml">
<!ENTITY para3 SYSTEM "para3.xml">
```

3. Produce `example.html` by normalizing `example.xml` .

```
% sgmlnorm -d example.xml > example.html
```

4. Load `example.html` into your web browser, and confirm that the `paran.xml` files have been included in `example.html` .

3.8. Marked sections

SGML provides a mechanism to indicate that particular pieces of the document should be processed in a special way. These are termed “marked sections”.

3.14. Structure of a marked section

```
<![ KEYWORD [
  Contents of marked section
]]>
```

As you would expect, being an SGML construct, a marked section starts with `<!`.

The first square bracket begins to delimit the marked section.

`KEYWORD` describes how this marked section should be processed by the parser.

The second square bracket indicates that the content of the marked section starts here.

The marked section is finished by closing the two square brackets, and then returning to the document context from the SGML context with `>`.

3.8.1. Marked section keywords

3.8.1.1. `CDATA`, `RCDATA`

These keywords denote the marked sections *content model*, and allow you to change it from the default.

When an SGML parser is processing a document it keeps track of what is called the "content model".

Briefly, the content model describes what sort of content the parser is expecting to see, and what it will do with it when it finds it.

The two content models you will probably find most useful are `CDATA` and `RCDATA`.

`CDATA` is for "Character Data". If the parser is in this content model then it is expecting to see characters, and characters only. In this model the `<` and `&` symbols lose their special status, and will be treated as ordinary characters.

`RCDATA` is for "Entity references and character data" If the parser is in this content model then it is expecting to see characters *and* entities. `<` loses its special status, but `&` will still be treated as starting the beginning of a general entity.

This is particularly useful if you are including some verbatim text that contains lots of `<` and `&` characters. While you could go through the text ensuring that every `<` is converted

to a `<` and every `&` is converted to a `&`, it can be easier to mark the section as only containing CDATA. When the SGML parser encounters this it will ignore the < and & symbols embedded in the content.

##

When you use `CDATA` or `RCDATA` in examples of text marked up in SGML, keep in mind that the content of `CDATA` is not validated. You have to check the included SGML text using other means. You could, for example, write the example in another document, validate the example code, and then paste it to your `CDATA` content.

3.15. Using a CDATA marked section

```
<para>Here is an example of how you would include some text
  that contained many <literal>&lt;</literal>
  and <literal>&amp;</literal> symbols.  The sample
  text is a fragment of HTML.  The surrounding text (<para> and
  <programlisting>) are from DocBook.</para>

<programlisting>
  <![CDATA[
    <p>This is a sample that shows you some of the elements ↺
within
      HTML.  Since the angle brackets are used so many times, ↺
it is
      simpler to say the whole example is a CDATA marked ↺
section
      than to use the entity names for the left and right angle
      brackets throughout.</p>

    <ul>
      <li>This is a listitem</li>
      <li>This is a second listitem</li>
      <li>This is a third listitem</li>
    </ul>

    <p>This is the end of the example.</p>
  -]]>
</programlisting>
```

If you look at the source for this document you will see this technique used throughout.

3.8.1.2. `INCLUDE` and `IGNORE`

If the keyword is `INCLUDE` then the contents of the marked section will be processed. If the keyword is `IGNORE` then the marked section is ignored and will not be processed. It will not appear in the output.

3.16. Using `INCLUDE` and `IGNORE` in marked sections

```
<![ INCLUDE [
  This text will be processed and included.
]]>

<![ IGNORE [
  This text will not be processed or included.
]]>
```

By itself, this is not too useful. If you wanted to remove text from your document you could cut it out, or wrap it in comments.

It becomes more useful when you realize you can use parameter entities to control this. Remember that parameter entities can only be used in SGML contexts, and the keyword of a marked section *is* an SGML context.

For example, suppose that you produced a hard-copy version of some documentation and an electronic version. In the electronic version you wanted to include some extra content that was not to appear in the hard-copy.

Create a parameter entity, and set its value to `INCLUDE`. Write your document, using marked sections to delimit content that should only appear in the electronic version. In these marked sections use the parameter entity in place of the keyword.

When you want to produce the hard-copy version of the document, change the parameter entity's value to `IGNORE` and reprocess the document.

3.17. Using a parameter entity to control a marked section

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY % electronic.copy "INCLUDE">
]]>
```

```
...

<![ %electronic.copy [
  This content should only appear in the electronic
  version of the document.
]]>
```

When producing the hard-copy version, change the entity's definition to:

```
<!ENTITY % electronic.copy "IGNORE">
```

On reprocessing the document, the marked sections that use `%electronic.copy` as their keyword will be ignored.

3.8.2. For you to do...

1. Create a new file, `section.xml` , that contains the following:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY % text.output "INCLUDE">
]>

<html>
  <head>
    <title>An example using marked sections</title>
  </head>

  <body>
    <p>This paragraph <![CDATA[contains many <
      characters (< < < < <) so it is easier
      to wrap it in a CDATA marked section -]]></p>

    <![IGNORE[
    <p>This paragraph will definitely not be included in the
      output.</p>
    -]]>

    <![%text.output [
    <p>This paragraph might appear in the output, or it
      might not.</p>

    <p>Its appearance is controlled by the %text.output
      parameter entity.</p>
    -]]>
  </body>
</html>
```

2. Normalize this file using sgmlnorm(1) and examine the output. Notice which paragraphs have appeared, which have disappeared, and what has happened to the content of the CDATA marked section.

3. Change the definition of the `text.output` entity from `INCLUDE` to `IGNORE`. Re-normalize the file, and examine the output to see what has changed.

3.9. Conclusion

That is the conclusion of this SGML primer. For reasons of space and complexity several things have not been covered in depth (or at all). However, the previous sections cover enough SGML for you to be able to follow the organization of the FDP documentation.

4. SGML Markup

This chapter describes the two markup languages you will encounter when you contribute to the FreeBSD documentation project. Each section describes the markup language, and details the markup that you are likely to want to use, or that is already in use.

These markup languages contain a large number of elements, and it can be confusing sometimes to know which element to use for a particular situation. This section goes through the elements you are most likely to need, and gives examples of how you would use them.

This is *not* an exhaustive list of elements, since that would just reiterate the documentation for each language. The aim of this section is to list those elements more likely to be useful to you. If you have a question about how best to markup a particular piece of content, please post it to the FreeBSD documentation project ####.

Inline vs. block

In the remainder of this document, when describing elements, *inline* means that the element can occur within a block element, and does not cause a line break. A *block* element, by comparison, will cause a line break (and other processing) when it is encountered.

4.1. HTML

HTML, the HyperText Markup Language, is the markup language of choice on the World Wide Web. More information can be found at <URL:`http://www.w3.org/` >.

HTML is used to markup pages on the FreeBSD web site. It should not (generally) be used to mark up other documentation, since DocBook offers a far richer set of elements to choose from. Consequently, you will normally only encounter HTML pages if you are writing for the web site.

HTML has gone through a number of versions, 1, 2, 3.0, 3.2, and the latest, 4.0 (available in both *strict* and *loose* variants).

The HTML DTDs are available from the ports collection in the textproc/html port. They are automatically installed as part of the textproc/docproj port.

4.1.1. Formal Public Identifier (FPI)

There are a number of HTML FPIs, depending upon the version (also known as the level) of HTML that you want to declare your document to be compliant with.

The majority of HTML documents on the FreeBSD web site comply with the loose version of HTML 4.0.

```
PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
```

4.1.2. Sectional elements

An HTML document is normally split into two sections. The first section, called the *head*, contains meta-information about the document, such as its title, the name of the author, the parent document, and so on. The second section, the *body*, contains the content that will be displayed to the user.

These sections are indicated with `head` and `body` elements respectively. These elements are contained within the top-level `html` element.

4.1. Normal HTML document structure

```
<html>
  <head>
   <title>The document's title </title>
  </head>

  <body>

    …

  </body>
</html>
```

4.1.3. Block elements

4.1.3.1. Headings

HTML allows you to denote headings in your document, at up to six different levels.

The largest and most prominent heading is `h1`, then `h2`, continuing down to `h6`.

The element's content is the text of the heading.

4.2. `h1`, `h2`, etc.

Use:

```
<h1>First section</h1>

<!-- Document introduction goes here -->

<h2>This is the heading for the first section</h2>

<!-- Content for the first section goes here -->

<h3>This is the heading for the first sub-section</h3>

<!-- Content for the first sub-section goes here -->

<h2>This is the heading for the second section</h2>

<!-- Content for the second section goes here -->
```

Generally, an HTML page should have one first level heading (`h1`). This can contain many second level headings (`h2`), which can in turn contain many third level headings. Each `h`*n* element should have the same element, but one further up the hierarchy, preceding it. Leaving gaps in the numbering is to be avoided.

4.3. Bad ordering of `h`*n* elements

Use:

```
<h1>First section</h1>

<!-- Document introduction -->

<h3>Sub-section</h3>

<!-- This is bad, <h2> has been left out -->
```

4.1.3.2. Paragraphs

HTML supports a single paragraph element, `p`.

4.4. `p`

Use:

```
<p>This is a paragraph.  It can contain just about any
  other element.</p>
```

4.1.3.3. Block quotations

A block quotation is an extended quotation from another document that should not appear within the current paragraph.

4.5. `blockquote`

Use:

```
<p>A small excerpt from the US Constitution:</p>

<blockquote>We the People of the United States, in Order to ↺
form
  a more perfect Union, establish Justice, insure domestic
  Tranquility, provide for the common defence, promote the ↺
general
  Welfare, and secure the Blessings of Liberty to ourselves ↺
and our
  Posterity, do ordain and establish this Constitution for the
  United States of America.</blockquote>
```

4.1.3.4. Lists

You can present the user with three types of lists, ordered, unordered, and definition.

Typically, each entry in an ordered list will be numbered, while each entry in an unordered list will be preceded by a bullet point. Definition lists are composed of two sections for each entry. The first section is the term being defined, and the second section is the definition of the term.

Ordered lists are indicated by the `ol` element, unordered lists by the `ul` element, and definition lists by the `dl` element.

Ordered and unordered lists contain listitems, indicated by the `li` element. A listitem can contain textual content, or it may be further wrapped in one or more `p` elements.

Definition lists contain definition terms (`dt`) and definition descriptions (`dd`). A definition term can only contain inline elements. A definition description can contain other block elements.

4.6. `ul` and `ol`

Use:

```
<p>An unordered list.  Listitems will probably be
  preceded by bullets.</p>

<ul>
  <li>First item</li>

  <li>Second item</li>

  <li>Third item</li>
</ul>

<p>An ordered list, with list items consisting of multiple
  paragraphs.  Each item (note: not each paragraph) will be
  numbered.</p>

<ol>
  <li><p>This is the first item.  It only has one paragraph.</
p></li>

  <li><p>This is the first paragraph of the second item.</p>

    <p>This is the second paragraph of the second item.</p></
li>

  <li><p>This is the first and only paragraph of the third
    item.</p></li>
</ol>
```

4.7. Definition lists with `dl`

Use:

```
<dl>
```

```
  <dt>Term 1</dt>

  <dd><p>Paragraph 1 of definition 1.</p>

    <p>Paragraph 2 of definition 1.</p></dd>

  <dt>Term 2</dt>

  <dd><p>Paragraph 1 of definition 2.</p></dd>

  <dt>Term 3</dt>

  <dd><p>Paragraph 1 of definition 3.</p></dd>
</dl>
```

4.1.3.5. Pre-formatted text

You can indicate that text should be shown to the user exactly as it is in the file. Typically, this means that the text is shown in a fixed font, multiple spaces are not merged into one, and line breaks in the text are significant.

In order to do this, wrap the content in the `pre` element.

4.8. **pre**

You could use `pre` to mark up an email message:

```
<pre>  From: nik@FreeBSD.org
  To: freebsd-doc@FreeBSD.org
  Subject: New documentation available

  There is a new copy of my primer for contributors to the ↺
FreeBSD
  Documentation Project available at

    &lt;URL:http://people.FreeBSD.org/~nik/primer/index.↺
html&gt;

  Comments appreciated.

  N</pre>
```

Keep in mind that `<` and `&` still are recognized as special characters in pre-formatted text. This is why the example shown had to use `<` instead of `<`. For consistency, `>` was used in place of `>`, too. Watch out for the special characters

that may appear in text copied from a plain-text source, e.g., an email message or program code.

4.1.3.6. Tables

##

Most text-mode browsers (such as Lynx) do not render tables particularly effectively. If you are relying on the tabular display of your content, you should consider using alternative markup to prevent confusion.

Mark up tabular information using the `table` element. A table consists of one or more table rows (`tr`), each containing one or more cells of table data (`td`). Each cell can contain other block elements, such as paragraphs or lists. It can also contain another table (this nesting can repeat indefinitely). If the cell only contains one paragraph then you do not need to include the `p` element.

4.9. Simple use of `table`

Use:

```
<p>This is a simple 2x2 table.</p>

<table>
  <tr>
    <td>Top left cell</td>

    <td>Top right cell</td>
  </tr>

  <tr>
    <td>Bottom left cell</td>

    <td>Bottom right cell</td>
  </tr>
</table>
```

A cell can span multiple rows and columns. To indicate this, add the `rowspan` and/or `colspan` attributes, with values indicating the number of rows of columns that should be spanned.

4.10. Using `rowspan`

Use:

```
<p>One tall thin cell on the left, two short cells next to
  it on the right.</p>

<table>
  <tr>
    <td rowspan="2">Long and thin</td>
  </tr>

  <tr>
    <td>Top cell</td>

    <td>Bottom cell</td>
  </tr>
</table>
```

4.11. Using `colspan`

Use:

```
<p>One long cell on top, two short cells below it.</p>

<table>
  <tr>
    <td colspan="2">Top cell</td>
  </tr>

  <tr>
    <td>Bottom left cell</td>

    <td>Bottom right cell</td>
  </tr>
</table>
```

4.12. Using `rowspan` and `colspan` together

Use:

```
<p>On a 3x3 grid, the top left block is a 2x2 set of
  cells merged into one.  The other cells are normal.</p>

<table>
  <tr>
    <td colspan="2" rowspan="2">Top left large cell</td>

    <td>Top right cell</td>
  </tr>

  <tr>
    <td>Middle right cell</td>
  </tr>

  <tr>
    <td>Bottom left cell</td>

    <td>Bottom middle cell</td>

    <td>Bottom right cell</td>
  </tr>
</table>
```

4.1.4. In-line elements

4.1.4.1. Emphasizing information

You have two levels of emphasis available in HTML, `em` and `strong`. `em` is for a normal level of emphasis and `strong` indicates stronger emphasis.

Typically, `em` is rendered in italic and `strong` is rendered in bold. This is not always the case, however, and you should not rely on it.

4.13. `em` and `strong`

Use:

```
<p><em>This</em> has been emphasized, while
```

```
  <strong>this</strong> has been strongly emphasized.</p>
```

4.1.4.2. Bold and italics

Because HTML includes presentational markup, you can also indicate that particular content should be rendered in bold or italic. The elements are `b` and `i` respectively.

4.14. `b` and `i`

```
<p><b>This</b> is in bold, while <i>this</i> is
  in italics.</p>
```

4.1.4.3. Indicating fixed pitch text

If you have content that should be rendered in a fixed pitch (typewriter) typeface, use `tt` (for "teletype").

4.15. `tt`

Use:

```
<p>This document was originally written by
  Nik Clayton, who can be reached by email as
  <tt>nik@FreeBSD.org</tt>.</p>
```

4.1.4.4. Content size

You can indicate that content should be shown in a larger or smaller font. There are three ways of doing this.

1. Use `big` and `small` around the content you wish to change size. These tags can be nested, so `<big><big>This is much bigger</big></big>` is possible.

2. Use `font` with the `size` attribute set to `+1` or `-1` respectively. This has the same effect as using `big` or `small`. However, the use of this approach is deprecated.

3. Use `font` with the `size` attribute set to a number between 1 and 7. The default font size is 3. This approach is deprecated.

4.16. `big`, `small`, and `font`

The following fragments all do the same thing.

```
<p>This text is <small>slightly smaller</small>.  But
  this text is <big>slightly bigger</big>.</p>

<p>This text is <font size="-1">slightly smaller</font>.  But
  this text is <font size="+1">slightly bigger</font.</p>

<p>This text is <font size="2">slightly smaller</font>.  But
  this text is <font size="4">slightly bigger</font>.</p>
```

4.1.5. Links

##

Links are also in-line elements.

4.1.5.1. Linking to other documents on the WWW

In order to include a link to another document on the WWW you must know the URL of the document you want to link to.

The link is indicated with `a`, and the `href` attribute contains the URL of the target document. The content of the element becomes the link, and is normally indicated to the user in some way (underlining, change of color, different mouse cursor when over the link, and so on).

4.17. Using `<a href="...">`

Use:

```
<p>More information is available at the
  <a href="http://www.FreeBSD.org/">FreeBSD web site</a>.</p>
```

These links will take the user to the top of the chosen document.

4.1.5.2. Linking to other parts of documents

Linking to a point within another document (or within the same document) requires that the document author include anchors that you can link to.

Anchors are indicated with `a` and the `name` attribute instead of `href`.

4.18. Using `<a name="...">`

Use:

```
<p><a name="para1">This</a> paragraph can be referenced
  in other links with the name <tt>para1</tt>.</p>
```

To link to a named part of a document, write a normal link to that document, but include the name of the anchor after a `#` symbol.

4.19. Linking to a named part of another document

Assume that the `para1` example resides in a document called `foo.html`.

```
<p>More information can be found in the
  <a href="foo.html#para1">first paragraph</a> of
  <tt>foo.html</tt>.</p>
```

If you are linking to a named anchor within the same document then you can omit the document's URL, and just include the name of the anchor (with the preceding #).

4.20. Linking to a named part of the same document

Assume that the `para1` example resides in this document:

```
<p>More information can be found in the
  <a href="#para1">first paragraph</a> of this
  document.</p>
```

4.2. DocBook

DocBook was originally developed by HaL Computer Systems and O'Reilly & Associates to be a DTD for writing technical documentation [1]. Since 1998 it is maintained by the DocBook Technical Committee. As such, and unlike LinuxDoc and HTML, DocBook is very heavily oriented towards markup that describes *what* something is, rather than describing *how* it should be presented.

`formal` VS. `informal`

Some elements may exist in two forms, *formal* and *informal*. Typically, the formal version of the element will consist of a title followed by the informal version of the element. The informal version will not have a title.

The DocBook DTD is available from the ports collection in the textproc/docbook port. It is automatically installed as part of the textproc/docproj port.

4.2.1. FreeBSD extensions

The FreeBSD Documentation Project has extended the DocBook DTD by adding some new elements. These elements serve to make some of the markup more precise.

Where a FreeBSD specific element is listed below it is clearly marked.

Throughout the rest of this document, the term "DocBook" is used to mean the FreeBSD extended DocBook DTD.

##

There is nothing about these extensions that is FreeBSD specific, it was just felt that they were useful enhancements for this particular project. Should anyone from any of the other *nix camps (NetBSD, OpenBSD, Linux, ...) be interested in collaborating on a standard DocBook extension set, please get in touch with Documentation Engineering Team <doceng@FreeBSD.org >.

The FreeBSD extensions are not (currently) in the ports collection. They are stored in the FreeBSD CVS tree, as doc/share/xml/freebsd.dtd.

[1] A short history can be found under http://www.oasis-open.org/committees/docbook/intro.shtml.

4.2.2. Formal Public Identifier (FPI)

In compliance with the DocBook guidelines for writing FPIs for DocBook customizations, the FPI for the FreeBSD extended DocBook DTD is:

```
PUBLIC "-//FreeBSD//DTD DocBook V4.1-Based Extension//EN"
```

4.2.3. Document structure

DocBook allows you to structure your documentation in several ways. In the FreeBSD Documentation Project we are using two primary types of DocBook document: the book and the article.

A book is organized into `chapter`s. This is a mandatory requirement. There may be `part`s between the book and the chapter to provide another layer of organization. The Handbook is arranged in this way.

A chapter may (or may not) contain one or more sections. These are indicated with the `sect1` element. If a section contains another section then use the `sect2` element, and so on, up to `sect5`.

Chapters and sections contain the remainder of the content.

An article is simpler than a book, and does not use chapters. Instead, the content of an article is organized into one or more sections, using the same `sect1` (and `sect2` and so on) elements that are used in books.

Obviously, you should consider the nature of the documentation you are writing in order to decide whether it is best marked up as a book or an article. Articles are well suited to information that does not need to be broken down into several chapters, and that is, relatively speaking, quite short, at up to 20-25 pages of content. Books are best suited to information that can be broken up into several chapters, possibly with appendices and similar content as well.

The FreeBSD tutorials are all marked up as articles, while this document, the FreeBSD FAQ, and the FreeBSD Handbook are all marked up as books.

4.2.3.1. Starting a book

The content of the book is contained within the `book` element. As well as containing structural markup, this element can contain elements that include additional information about the book. This is either meta-information, used for reference purposes, or additional content used to produce a title page.

This additional information should be contained within `bookinfo`.

4.21. Boilerplate **book** with **bookinfo**

```
<book>
  <bookinfo>
    <title>Your title here </title>

    <author>
      <firstname>Your first name </firstname>
      <surname>Your surname </surname>
      <affiliation>
        <address><email>Your email address </email></address>
      </affiliation>
    </author>

    <copyright>
      <year>1998</year>
      <holder role="mailto:your email address ">Your name </
holder>
    </copyright>

    <releaseinfo>$FreeBSD$</releaseinfo>

    <abstract>
      <para>Include an abstract of the book's contents here.  </
para>
    </abstract>
  </bookinfo>

  …

</book>
```

4.2.3.2. Starting an article

The content of the article is contained within the `article` element. As well as containing structural markup, this element can contain elements that include additional information about the article. This is either meta-information, used for reference purposes, or additional content used to produce a title page.

This additional information should be contained within `articleinfo` .

4.22. Boilerplate **article** with **articleinfo**

```
<article>
```

```
  <articleinfo>
    <title>Your title here </title>

    <author>
      <firstname>Your first name </firstname>
      <surname>Your surname </surname>
      <affiliation>
        <address><email>Your email address </email></address>
      </affiliation>
    </author>

    <copyright>
      <year>1998</year>
      <holder role="mailto:your email address ">Your name </
holder>
    </copyright>

    <releaseinfo>$FreeBSD$</releaseinfo>

    <abstract>
      <para>Include an abstract of the article's contents ↺
here.</para>
    </abstract>
  </articleinfo>

  …

</article>
```

4.2.3.3. Indicating chapters

Use `chapter` to mark up your chapters. Each chapter has a mandatory `title`. Articles do not contain chapters, they are reserved for books.

4.23. A simple chapter

```
<chapter>
  <title>The chapter's title</title>

  ...
</chapter>
```

A chapter cannot be empty; it must contain elements in addition to `title`. If you need to include an empty chapter then just use an empty paragraph.

4.24. Empty chapters

```
<chapter>
  <title>This is an empty chapter</title>

  <para></para>
</chapter>
```

4.2.3.4. Sections below chapters

In books, chapters may (but do not need to) be broken up into sections, subsections, and so on. In articles, sections are the main structural element, and each article must contain at least one section. Use the `sect`*n* element. The *n* indicates the section number, which identifies the section level.

The first `sect`*n* is `sect1`. You can have one or more of these in a chapter. They can contain one or more `sect2` elements, and so on, down to `sect5`.

4.25. Sections in chapters

```
<chapter>
  <title>A sample chapter</title>

  <para>Some text in the chapter.</para>

  <sect1>
    <title>First section (1.1)</title>

    &hellip;
  </sect1>

  <sect1>
    <title>Second section (1.2)</title>

    <sect2>
      <title>First sub-section (1.2.1)</title>

      <sect3>
        <title>First sub-sub-section (1.2.1.1)</title>

        &hellip;
      </sect3>
```

```
    </sect2>

    <sect2>
      <title>Second sub-section (1.2.2)</title>

      &hellip;
    </sect2>
  </sect1>
</chapter>
```

##

This example includes section numbers in the section titles. You should not do this in your documents. Adding the section numbers is carried out by the stylesheets (of which more later), and you do not need to manage them yourself.

4.2.3.5. Subdividing using `part`s

You can introduce another layer of organization between `book` and `chapter` with one or more `part`s. This cannot be done in an `article`.

```
<part>
  <title>Introduction</title>

  <chapter>
    <title>Overview</title>

    ...
  </chapter>

  <chapter>
    <title>What is FreeBSD?</title>

    ...
  </chapter>

  <chapter>
    <title>History</title>

    ...
  </chapter>
</part>
```

4.2.4. Block elements

4.2.4.1. Paragraphs

DocBook supports three types of paragraphs: `formalpara` , `para`, and `simpara`.

Most of the time you will only need to use `para`. `formalpara` includes a `title` element, and `simpara` disallows some elements from within `para`. Stick with `para`.

4.26. **para**

Use:

```
<para>This is a paragraph.  It can contain just about any
  other element.</para>
```

Appearance:

This is a paragraph. It can contain just about any other element.

4.2.4.2. Block quotations

A block quotation is an extended quotation from another document that should not appear within the current paragraph. You will probably only need it infrequently.

Blockquotes can optionally contain a title and an attribution (or they can be left untitled and unattributed).

4.27. **blockquote**

Use:

```
<para>A small excerpt from the US Constitution:</para>

<blockquote>
  <title>Preamble to the Constitution of the United States</
title>

  <attribution>Copied from a web site somewhere</attribution>

  <para>We the People of the United States, in Order to form ↺
a more perfect
    Union, establish Justice, insure domestic Tranquility, ↺
provide for the
```

```
    common defence, promote the general Welfare, and secure ↺
the Blessings
    of Liberty to ourselves and our Posterity, do ordain and ↺
establish this
    Constitution for the United States of America.</para>
</blockquote>
```

Appearance:

> Preamble to the Constitution of the United States
>
> We the People of the United States, in Order to form a more perfect Union, establish Justice, insure domestic Tranquility, provide for the common defence, promote the general Welfare, and secure the Blessings of Liberty to ourselves and our Posterity, do ordain and establish this Constitution for the United States of America.
>
> —Copied from a web site somewhere

4.2.4.3. Tips, notes, warnings, cautions, important information and sidebars.

You may need to include extra information separate from the main body of the text. Typically this is "meta" information that the user should be aware of.

Depending on the nature of the information, one of `tip`, `note`, `warning`, `caution`, and `important` should be used. Alternatively, if the information is related to the main text but is not one of the above, use `sidebar`.

The circumstances in which to choose one of these elements over another is unclear. The DocBook documentation suggests:

- A Note is for information that should be heeded by all readers.
- An Important element is a variation on Note.
- A Caution is for information regarding possible data loss or software damage.
- A Warning is for information regarding possible hardware damage or injury to life or limb.

4.28. **warning**

Use:

```
<warning>
  <para>Installing FreeBSD may make you want to delete ↺
Windows from your
    hard disk.</para>
</warning>
```

##

Installing FreeBSD may make you want to delete Windows from your hard disk.

4.2.4.4. Lists and procedures

You will often need to list pieces of information to the user, or present them with a number of steps that must be carried out in order to accomplish a particular goal.

In order to do this, use `itemizedlist` , `orderedlist` , or `procedure` [2]

`itemizedlist` and `orderedlist` are similar to their counterparts in HTML, `ul` and `ol`. Each one consists of one or more `listitem` elements, and each `listitem` contains one or more block elements. The `listitem` elements are analogous to HTML's `li` tags. However, unlike HTML, they are required.

`procedure` is slightly different. It consists of `step`s, which may in turn consists of more `step`s or `substep`s. Each `step` contains block elements.

4.29. `itemizedlist` , `orderedlist` , and `procedure`

Use:

```
<itemizedlist>
  <listitem>
    <para>This is the first itemized item.</para>
  </listitem>

  <listitem>
    <para>This is the second itemized item.</para>
  </listitem>
</itemizedlist>
```

[2] There are other types of list element in DocBook, but we are not concerned with those at the moment.

```
<orderedlist>
  <listitem>
    <para>This is the first ordered item.</para>
  </listitem>

  <listitem>
    <para>This is the second ordered item.</para>
  </listitem>
</orderedlist>

<procedure>
  <step>
    <para>Do this.</para>
  </step>

  <step>
    <para>Then do this.</para>
  </step>

  <step>
    <para>And now do this.</para>
  </step>
</procedure>
```

Appearance:

- This is the first itemized item.
- This is the second itemized item.

1. This is the first ordered item.
2. This is the second ordered item.

1. Do this.
2. Then do this.
3. And now do this.

4.2.4.5. Showing file samples

If you want to show a fragment of a file (or perhaps a complete file) to the user, wrap it in the `programlisting` element.

White space and line breaks within `programlisting` *are* significant. In particular, this means that the opening tag should appear on the same line as the first line of the output, and the closing tag should appear on the same line as the last line of the output, otherwise spurious blank lines may be included.

4.30. `programlisting`

Use:

```
<para>When you have finished, your program should look like
  this:</para>

<programlisting>#include &lt;stdio.h&gt;

int
main(void)
{
    printf("hello, world\n");
}</programlisting>
```

Notice how the angle brackets in the `#include` line need to be referenced by their entities instead of being included literally.

Appearance:

When you have finished, your program should look like this:

```
#include <stdio.h>

int
main(void)
{
    printf("hello, world\n");
}
```

4.2.4.6. Callouts

A callout is a mechanism for referring back to an earlier piece of text or specific position within an earlier example without linking to it within the text.

To do this, mark areas of interest in your example (`programlisting` , `literallayout` , or whatever) with the `co` element. Each element must have a unique `id` assigned to it. After the example include a `calloutlist` that refers back to the example and provides additional commentary.

4.31. `co` and `calloutlist`

```
<para>When you have finished, your program should look like
  this:</para>
```

```
<programlisting>#include &lt;stdio.h&gt; <co id="co-ex-
include"/>

int <co id="co-ex-return"/>
main(void)
{
    printf("hello, world\n"); <co id="co-ex-printf"/>
}</programlisting>

<calloutlist>
  <callout arearefs="co-ex-include">
    <para>Includes the standard IO header file.</para>
  </callout>

  <callout arearefs="co-ex-return">
    <para>Specifies that <function>main()</function> returns an
      int.</para>
  </callout>

  <callout arearefs="co-ex-printf">
    <para>The <function>printf()</function> call that writes
      <literal>hello, world</literal> to standard output.</
para>
  </callout>
</calloutlist>
```

Appearance:

When you have finished, your program should look like this:

```
#include <stdio.h> ❶

int ❷
main(void)
{
    printf("hello, world\n"); ❸
}
```

❶ Includes the standard IO header file.
❷ Specifies that `main()` returns an int.
❸ The `printf()` call that writes `hello, world` to standard output.

4.2.4.7. Tables

Unlike HTML, you do not need to use tables for layout purposes, as the stylesheet handles those issues for you. Instead, just use tables for marking up tabular data.

In general terms (and see the DocBook documentation for more detail) a table (which can be either formal or informal) consists of a `table` element. This contains at least one

`tgroup` element, which specifies (as an attribute) the number of columns in this table group. Within the tablegroup you can then have one `thead` element, which contains elements for the table headings (column headings), and one `tbody` which contains the body of the table.

Both `tgroup` and `thead` contain `row` elements, which in turn contain `entry` elements. Each `entry` element specifies one cell in the table.

4.32. `informaltable`

Use:

```
<informaltable frame="none" pgwide="1">
  <tgroup cols="2">
    <thead>
      <row>
        <entry>This is column head 1</entry>
        <entry>This is column head 2</entry>
      </row>
    </thead>

    <tbody>
      <row>
        <entry>Row 1, column 1</entry>
        <entry>Row 1, column 2</entry>
      </row>

      <row>
        <entry>Row 2, column 1</entry>
        <entry>Row 2, column 2</entry>
      </row>
    </tbody>
  </tgroup>
</informaltable>
```

Appearance:

This is column head 1	This is column head 2
Row 1, column 1	Row 1, column 2
Row 2, column 1	Row 2, column 2

Always use the `pgwide` attribute with a value of `1` with the `informaltable` element. A bug in Internet Explorer can cause the table to render incorrectly if this is omitted.

If you do not want a border around the table the `frame` attribute can be added to the `informaltable` element with a value of `none` (i.e., `<informaltable frame="none">`).

4.33. Tables where `frame="none"`

Appearance:

This is column head 1	This is column head 2
Row 1, column 1	Row 1, column 2
Row 2, column 1	Row 2, column 2

4.2.4.8. Examples for the user to follow

A lot of the time you need to show examples for the user to follow. Typically, these will consist of dialogs with the computer; the user types in a command, the user gets a response back, they type in another command, and so on.

A number of distinct elements and entities come into play here.

`screen`
: Everything the user sees in this example will be on the computer screen, so the next element is `screen`.

  Within `screen`, white space is significant.

`prompt`, `&prompt.root;` and `&prompt.user;`
: Some of the things the user will be seeing on the screen are prompts from the computer (either from the operating system, command shell, or application). These should be marked up using `prompt`.

  As a special case, the two shell prompts for the normal user and the root user have been provided as entities. Every time you want to indicate the user is at a shell prompt, use one of `&prompt.root;` and `&prompt.user;` as necessary. They do not need to be inside `prompt`.

##

`&prompt.root;` and `&prompt.user;` are FreeBSD extensions to DocBook, and are not part of the original DTD.

`userinput`
: When displaying text that the user should type in, wrap it in `userinput` tags. It will probably be displayed differently to the user.

4.34. `screen`, `prompt`, and `userinput`

Use:

```
<screen>&prompt.user; <userinput>ls -1</userinput>
foo1
foo2
foo3
&prompt.user; <userinput>ls -1 | grep foo2</userinput>
foo2
&prompt.user; <userinput>su</userinput>
<prompt>Password: </prompt>
&prompt.root; <userinput>cat foo2</userinput>
This is the file called 'foo2'</screen>
```

Appearance:

```
% ls -1
foo1
foo2
foo3
% ls -1 | grep foo2
foo2
% su
Password:
# cat foo2
This is the file called 'foo2'
```

##

Even though we are displaying the contents of the file `foo2`, it is *not* marked up as `programlisting` . Reserve `programlisting` for showing fragments of files outside the context of user actions.

4.2.5. In-line elements

4.2.5.1. Emphasizing information

When you want to emphasize a particular word or phrase, use `emphasis` . This may be presented as italic, or bold, or might be spoken differently with a text-to-speech system.

There is no way to change the presentation of the emphasis within your document, no equivalent of HTML's `b` and `i`. If the information you are presenting is important then consider presenting it in `important` rather than `emphasis`.

4.35. `emphasis`

Use:

```
<para>FreeBSD is without doubt <emphasis>the</emphasis>
  premiere Unix like operating system for the Intel ↺
architecture.</para>
```

Appearance:

FreeBSD is without doubt *the* premiere Unix like operating system for the Intel architecture.

4.2.5.2. Quotations

To quote text from another document or source, or to denote a phrase that is used figuratively, use `quote`. Within a `quote` tag, you may use most of the markup tags available for normal text.

4.36. Quotations

Use:

```
<para>However, make sure that the search does not go beyond the
  <quote>boundary between local and public administration</
quote>,
  as RFC 1535 calls it.</para>
```

Appearance:

However, make sure that the search does not go beyond the “boundary between local and public administration”, as RFC 1535 calls it.

4.2.5.3. Keys, mouse buttons, and combinations

To refer to a specific key on the keyboard, use `keycap`. To refer to a mouse button, use `mousebutton`. And to refer to combinations of key presses or mouse clicks, wrap them all in `keycombo`.

`keycombo` has an attribute called `action`, which may be one of `click`, `double-click`, `other`, `press`, `seq`, or `simul`. The last two values denote whether the keys or buttons should be pressed in sequence, or simultaneously.

The stylesheets automatically add any connecting symbols, such as +, between the key names, when wrapped in `keycombo`.

4.37. Keys, mouse buttons, and combinations

Use:

```
<para>To switch to the second virtual terminal, press
  <keycombo action="simul"><keycap>Alt</keycap>
    <keycap>F1</keycap></keycombo>.</para>

<para>To exit <command>vi</command> without saving your work, ↺
type
  <keycombo action="seq"><keycap>Esc</keycap><keycap>:</keycap>
    <keycap>q</keycap><keycap>!</keycap></keycombo>.</para>

<para>My window manager is configured so that
  <keycombo action="simul"><keycap>Alt</keycap>
    <mousebutton>right</mousebutton>
  </keycombo> mouse button is used to move windows.</para>
```

Appearance:

To switch to the second virtual terminal, press Alt+F1.

To exit `vi` without saving your work, type Esc : q !.

My window manager is configured so that Alt+right mouse button is used to move windows.

4.2.5.4. Applications, commands, options, and cites

You will frequently want to refer to both applications and commands when writing for the Handbook. The distinction between them is simple: an application is the name for a suite (or possibly just 1) of programs that fulfil a particular task. A command is the name of a program that the user can run.

In addition, you will occasionally need to list one or more of the options that a command might take.

Finally, you will often want to list a command with its manual section number, in the “command(number)” format so common in Unix manuals.

Mark up application names with `application`.

When you want to list a command with its manual section number (which should be most of the time) the DocBook element is `citerefentry`. This will contain a further two elements, `refentrytitle` and `manvolnum`. The content of `refentrytitle` is the name of the command, and the content of `manvolnum` is the manual page section.

This can be cumbersome to write, and so a series of general entities have been created to make this easier. Each entity takes the form `&man.manual-page.manual-section;`.

The file that contains these entities is in `doc/share/xml/man-refs.ent`, and can be referred to using this FPI:

```
PUBLIC "-//FreeBSD//ENTITIES DocBook Manual Page Entities//EN"
```

Therefore, the introduction to your documentation will probably look like this:

```
<!DOCTYPE book PUBLIC "-//FreeBSD//DTD DocBook V4.1-Based Extension//
EN" [

<!ENTITY % man PUBLIC "-//FreeBSD//ENTITIES DocBook Manual Page ↺
Entities//EN">
%man;

…

]>
```

Use `command` when you want to include a command name “in-line” but present it as something the user should type in.

Use `option` to mark up the options which will be passed to a command.

When referring to the same command multiple times in close proximity it is preferred to use the `&man.command.section;` notation to markup the first reference and use `command` to markup subsequent references. This makes the generated output, especially HTML, appear visually better.

This can be confusing, and sometimes the choice is not always clear. Hopefully this example makes it clearer.

4.38. Applications, commands, and options.

Use:

```
<para><application>Sendmail</application> is the most
```

```
  widely used Unix mail application.</para>

<para><application>Sendmail</application> includes the
  <citerefentry>
    <refentrytitle>sendmail</refentrytitle>
    <manvolnum>8</manvolnum>
  </citerefentry>, &man.mailq.8;, and &man.newaliases.8;
  programs.</para>

<para>One of the command line parameters to <citerefentry>
    <refentrytitle>sendmail</refentrytitle>
    <manvolnum>8</manvolnum>
  </citerefentry>, <option>-bp</option>, will display the ↺
current
  status of messages in the mail queue.  Check this on the ↺
command
  line by running <command>sendmail -bp</command>.</para>
```

Appearance:

Sendmail is the most widely used Unix mail application.

Sendmail includes the sendmail(8), mailq(8), and newaliases(8) programs.

One of the command line parameters to sendmail(8), `-bp`, will display the current status of messages in the mail queue. Check this on the command line by running `sendmail -bp` .

##

Notice how the `&man.command.section;` notation is easier to follow.

4.2.5.5. Files, directories, extensions

Whenever you wish to refer to the name of a file, a directory, or a file extension, use `filename` .

4.39. **`filename`**

Use:

```
<para>The SGML source for the Handbook in English can be
```

```
  found in <filename>/usr/doc/en/handbook/</filename>.  The ↺
first
  file is called <filename>handbook.xml</filename> in that
  directory.  You should also see a <filename>Makefile</
filename>
  and a number of files with a <filename>.ent</filename>
  extension.</para>
```

Appearance:

The SGML source for the Handbook in English can be found in `/usr/doc/en/handbook/`. The first file is called `handbook.xml` in that directory. You should also see a `Makefile` and a number of files with a `.ent` extension.

4.2.5.6. The name of ports

FreeBSD extension

These elements are part of the FreeBSD extension to DocBook, and do not exist in the original DocBook DTD.

You might need to include the name of a program from the FreeBSD Ports Collection in the documentation. Use the `filename` tag with the `role` attribute set to `package` to identify these. Since ports can be installed in any number of locations, only include the category and the port name; do not include `/usr/ports`.

4.40. `filename` tag with `package` role

Use:

```
<para>Install the <filename role="package">net/ethereal</
filename> port to view network traffic.</para>
```

Appearance:

Install the net/ethereal port to view network traffic.

4.2.5.7. Devices

FreeBSD extension

These elements are part of the FreeBSD extension to DocBook, and do not exist in the original DocBook DTD.

When referring to devices you have two choices. You can either refer to the device as it appears in `/dev`, or you can use the name of the device as it appears in the kernel. For this latter course, use `devicename` .

Sometimes you will not have a choice. Some devices, such as networking cards, do not have entries in `/dev`, or the entries are markedly different from those entries.

4.41. **devicename**

Use:

```
<para><devicename>sio</devicename> is used for serial
  communication in FreeBSD.  <devicename>sio</devicename> ↺
manifests
  through a number of entries in <filename>/dev</filename>, ↺
including
  <filename>/dev/ttyd0</filename> and <filename>/dev/cuaa0</
filename>.</para>

<para>By contrast, the networking devices, such as
  <devicename>ed0</devicename> do not appear in <filename>/
dev</filename>.</para>

<para>In MS-DOS, the first floppy drive is referred to as
  <devicename>a:</devicename>.  In FreeBSD it is
  <filename>/dev/fd0</filename>.</para>
```

Appearance:

`sio` is used for serial communication in FreeBSD. `sio` manifests through a number of entries in `/dev`, including `/dev/ttyd0` and `/dev/cuaa0` .

By contrast, the networking devices, such as `ed0` do not appear in `/dev`.

In MS-DOS, the first floppy drive is referred to as `a:`. In FreeBSD it is `/dev/fd0` .

4.2.5.8. Hosts, domains, IP addresses, and so forth

FreeBSD extension

These elements are part of the FreeBSD extension to DocBook, and do not exist in the original DocBook DTD.

You can markup identification information for networked computers (hosts) in several ways, depending on the nature of the information. All of them use `hostid` as the element, with the `role` attribute selecting the type of the marked up information.

No role attribute, or `role="hostname"`
: With no role attribute (i.e., `hostid …/hostid`) the marked up information is the simple hostname, such as `freefall` or `wcarchive.` You can explicitly specify this with `role="hostname"` .

`role="domainname"`
: The text is a domain name, such as `FreeBSD.org` or `ngo.org.uk`. There is no hostname component.

`role="fqdn"`
: The text is a Fully Qualified Domain Name, with both hostname and domain name parts.

`role="ipaddr"`
: The text is an IP address, probably expressed as a dotted quad.

`role="ip6addr"`
: The text is an IPv6 address.

`role="netmask"`
: The text is a network mask, which might be expressed as a dotted quad, a hexadecimal string, or as a `/` followed by a number.

`role="mac"`
: The text is an Ethernet MAC address, expressed as a series of 2 digit hexadecimal numbers separated by colons.

4.42. **`hostid`** and roles

Use:

```
<para>The local machine can always be referred to by the
  name <hostid>localhost</hostid>, which will have the IP ↺
address
  <hostid role="ipaddr">127.0.0.1</hostid>.</para>

<para>The <hostid role="domainname">FreeBSD.org</hostid> domain
  contains a number of different hosts, including
  <hostid role="fqdn">freefall.FreeBSD.org</hostid> and
  <hostid role="fqdn">bento.FreeBSD.org</hostid>.</para>

<para>When adding an IP alias to an interface (using
  <command>ifconfig</command>) <emphasis>always</emphasis> ↺
use a
  netmask of <hostid role="netmask">255.255.255.255</hostid>
  (which can also be expressed as <hostid
    role="netmask">0xffffffff</hostid>.</para>

<para>The MAC address uniquely identifies every network card
  in existence.  A typical MAC address looks like <hostid
    role="mac">08:00:20:87:ef:d0</hostid>.</para>
```

Appearance:

The local machine can always be referred to by the name `localhost` , which will have the IP address `127.0.0.1` .

The `FreeBSD.org` domain contains a number of different hosts, including `freefall.FreeBSD.org` and `bento.FreeBSD.org` .

When adding an IP alias to an interface (using `ifconfig`) *always* use a netmask of `255.255.255.255` (which can also be expressed as `0xffffffff` .

The MAC address uniquely identifies every network card in existence. A typical MAC address looks like `08:00:20:87:ef:d0` .

4.2.5.9. Usernames

FreeBSD extension

These elements are part of the FreeBSD extension to DocBook, and do not exist in the original DocBook DTD.

When you need to refer to a specific username, such as `root` or `bin`, use `username` .

4.43. `username`

Use:

```
<para>To carry out most system administration functions you
  will need to be <username>root</username>.</para>
```

Appearance:

To carry out most system administration functions you will need to be `root`.

4.2.5.10. Describing `Makefile`s

FreeBSD extension

These elements are part of the FreeBSD extension to DocBook, and do not exist in the original DocBook DTD.

Two elements exist to describe parts of `Makefile` s, `maketarget` and `makevar`.

`maketarget` identifies a build target exported by a `Makefile` that can be given as a parameter to `make`. `makevar` identifies a variable that can be set (in the environment, on the `make` command line, or within the `Makefile`) to influence the process.

4.44. `maketarget` and `makevar`

Use:

```
<para>Two common targets in a <filename>Makefile</filename>
  are <maketarget>all</maketarget> and <maketarget>clean</
maketarget>.</para>

<para>Typically, invoking <maketarget>all</maketarget> will ↺
rebuild the
  application, and invoking <maketarget>clean</maketarget> ↺
will remove
  the temporary files (<filename>.o</filename> for example) ↺
created by
  the build process.</para>
```

```
<para><maketarget>clean</maketarget> may be controlled by a ↺
number of
  variables, including <makevar>CLOBBER</makevar> and
  <makevar>RECURSE</makevar>.</para>
```

Appearance:

Two common targets in a `Makefile` are `all` and `clean`.

Typically, invoking `all` will rebuild the application, and invoking `clean` will remove the temporary files (`.o` for example) created by the build process.

`clean` may be controlled by a number of variables, including `CLOBBER` and `RECURSE`.

4.2.5.11. Literal text

You will often need to include "literal" text in the Handbook. This is text that is excerpted from another file, or which should be copied from the Handbook into another file verbatim.

Some of the time, `programlisting` will be sufficient to denote this text. `programlisting` is not always appropriate, particularly when you want to include a portion of a file "in-line" with the rest of the paragraph.

On these occasions, use `literal`.

4.45. `literal`

Use:

```
<para>The <literal>maxusers 10</literal> line in the kernel
  configuration file determines the size of many system ↺
tables, and is
  a rough guide to how many simultaneous logins the system will
  support.</para>
```

Appearance:

The `maxusers 10` line in the kernel configuration file determines the size of many system tables, and is a rough guide to how many simultaneous logins the system will support.

4.2.5.12. Showing items that the user must fill in

There will often be times when you want to show the user what to do, or refer to a file, or command line, or similar, where the user cannot simply copy the examples that you provide, but must instead include some information themselves.

`replaceable` is designed for this eventuality. Use it *inside* other elements to indicate parts of that element's content that the user must replace.

4.46. `replaceable`

Use:

```
<informalexample>
  <screen>&prompt.user; <userinput>man <replaceable>command</
replaceable></userinput></screen>
</informalexample>
```

Appearance:

```
% man command
```

`replaceable` can be used in many different elements, including `literal`. This example also shows that `replaceable` should only be wrapped around the content that the user *is* meant to provide. The other content should be left alone.

Use:

```
<para>The <literal>maxusers <replaceable>n</replaceable></
literal>
  line in the kernel configuration file determines the size ↺
of many system
  tables, and is a rough guide to how many simultaneous ↺
logins the system will
  support.</para>

<para>For a desktop workstation, <literal>32</literal> is a ↺
good value
  for <replaceable>n</replaceable>.</para>
```

Appearance:

The `maxusers n` line in the kernel configuration file determines the size of many system tables, and is a rough guide to how many simultaneous logins the system will support.

For a desktop workstation, 32 is a good value for *n*.

4.2.5.13. Quoting system errors

You might want to show errors generated by FreeBSD. Mark these with `errorname` . This indicates the exact error that appears.

4.47. `errorname`

Use:

```
<screen><errorname>Panic: cannot mount root</errorname></
screen>
```

Appearance:

```
Panic: cannot mount root
```

4.2.6. Images

##

Image support in the documentation is currently extremely experimental. I think the mechanisms described here are unlikely to change, but that is not guaranteed.

You will also need to install the graphics/ImageMagick port, which is used to convert between the different image formats. This is a big port, and most of it is not required. However, while we are working on the `Makefile` s and other infrastructure it makes things easier. This port is *not* in the textproc/docproj meta port, you must install it by hand.

The best example of what follows in practice is the `doc/en_US.ISO8859-1/articles/vm-design/` document. If you are unsure of the description that follows, take a look at the files in that directory to see how everything hangs together. Experiment with creating different formatted versions of the document to see how the image markup appears in the formatted output.

4.2.6.1. Image formats

We currently support two formats for images. The format you should use will depend on the nature of your image.

For images that are primarily vector based, such as network diagrams, time lines, and similar, use Encapsulated Postscript, and make sure that your images have the `.eps` extension.

For bitmaps, such as screen captures, use the Portable Network Graphic format, and make sure that your images have the `.png` extension.

These are the *only* formats in which images should be committed to the CVS repository.

Use the right format for the right image. It is to be expected that your documentation will have a mix of EPS and PNG images. The `Makefile`s ensure that the correct format image is chosen depending on the output format that you use for your documentation. *Do not commit the same image to the repository in two different formats.*

##

It is anticipated that the Documentation Project will switch to using the Scalable Vector Graphic (SVG) format for vector images. However, the current state of SVG capable editing tools makes this impractical.

4.2.6.2. Markup

The markup for an image is relatively simple. First, markup a `mediaobject`. The `mediaobject` can contain other, more specific objects. We are concerned with two, the `imageobject` and the `textobject`.

You should include one `imageobject`, and two `textobject` elements. The `imageobject` will point to the name of the image file that will be used (without the extension). The `textobject` elements contain information that will be presented to the user as well as, or instead of, the image.

There are two circumstances where this can happen.

- When the reader is viewing the documentation in HTML. In this case, each image will need to have associated alternate text to show the user, typically whilst the image is loading, or if they hover the mouse pointer over the image.
- When the reader is viewing the documentation in plain text. In this case, each image should have an ASCII art equivalent to show the user.

An example will probably make things easier to understand. Suppose you have an image, called `fig1.png` , that you want to include in the document. This image is of a rectangle with an A inside it. The markup for this would be as follows.

```
<mediaobject>
  <imageobject>
    <imagedata fileref="fig1"> ❶
  </imageobject>

  <textobject>
    <literallayout class="monospaced">+---------------+ ❷
|       A       |
+---------------+</literallayout>
  </textobject>

  <textobject>
    <phrase>A picture</phrase> ❸
  </textobject>
</mediaobject>
```

❶ Include an `imagedata` element inside the `imageobject` element. The `fileref` attribute should contain the filename of the image to include, without the extension. The stylesheets will work out which extension should be added to the filename automatically.

❷ The first `textobject` should contain a `literallayout` element, where the `class` attribute is set to `monospaced` . This is your opportunity to demonstrate your ASCII art skills. This content will be used if the document is converted to plain text.

Notice how the first and last lines of the content of the `literallayout` element butt up next to the element's tags. This ensures no extraneous white space is included.

❸ The second `textobject` should contain a single `phrase` element. The contents of this will become the `alt` attribute for the image when this document is converted to HTML.

4.2.6.3. `Makefile` entries

Your images must be listed in the `Makefile` in the `IMAGES` variable. This variable should contain the name of all your *source* images. For example, if you have created three figures, `fig1.eps` , `fig2.png` , `fig3.png` , then your `Makefile` should have lines like this in it.

```
…
IMAGES= fig1.eps fig2.png fig3.png
…
```

or

```
…
IMAGES=  fig1.eps
IMAGES+= fig2.png
IMAGES+= fig3.png
```

```
…
```

Again, the `Makefile` will work out the complete list of images it needs to build your source document, you only need to list the image files *you* provided.

4.2.6.4. Images and chapters in subdirectories

You must be careful when you separate your documentation into smaller files (see # 3.7.1, "Using general entities to include files") in different directories.

Suppose you have a book with three chapters, and the chapters are stored in their own directories, called `chapter1/chapter.xml` , `chapter2/chapter.xml` , and `chapter3/chapter.xml` . If each chapter has images associated with it, I suggest you place those images in each chapter's subdirectory (`chapter1/` , `chapter2/` , and `chapter3/`).

However, if you do this you must include the directory names in the `IMAGES` variable in the `Makefile` , *and* you must include the directory name in the `imagedata` element in your document.

For example, if you have `chapter1/fig1.png` , then `chapter1/chapter.xml` should contain:

```
<mediaobject>
  <imageobject>
    <imagedata fileref="chapter1/fig1"> ❶
  </imageobject>

  …

</mediaobject>
```

❶ The directory name must be included in the `fileref` attribute.

The `Makefile` must contain:

```
…
IMAGES=  chapter1/fig1.png
…
```

Then everything should just work.

4.2.7. Links

##

Links are also in-line elements.

4.2.7.1. Linking to other parts of the same document

Linking within the same document requires you to specify where you are linking from (i.e., the text the user will click, or otherwise indicate, as the source of the link) and where you are linking to (the link's destination).

Each element within DocBook has an attribute called `id`. You can place text in this attribute to uniquely name the element it is attached to.

This value will be used when you specify the link source.

Normally, you will only be linking to chapters or sections, so you would add the `id` attribute to these elements.

4.48. `id` on chapters and sections

```
<chapter id="chapter1">
  <title>Introduction</title>

  <para>This is the introduction.  It contains a subsection,
    which is identified as well.</para>

  <sect1 id="chapter1-sect1">
    <title>Sub-sect 1</title>

    <para>This is the subsection.</para>
  </sect1>
</chapter>
```

Obviously, you should use more descriptive values. The values must be unique within the document (i.e., not just the file, but the document the file might be included in as well). Notice how the `id` for the subsection is constructed by appending text to the `id` of the chapter. This helps to ensure that they are unique.

If you want to allow the user to jump into a specific portion of the document (possibly in the middle of a paragraph or an example), use `anchor`. This element has no content, but takes an `id` attribute.

4.49. `anchor`

```
<para>This paragraph has an embedded
  <anchor id="para1">link target in it.  It will not show up in
```

```
  the document.</para>
```

When you want to provide the user with a link they can activate (probably by clicking) to go to a section of the document that has an `id` attribute, you can use either `xref` or `link`.

Both of these elements have a `linkend` attribute. The value of this attribute should be the value that you have used in a `id` attribute (it does not matter if that value has not yet occurred in your document; this will work for forward links as well as backward links).

If you use `xref` then you have no control over the text of the link. It will be generated for you.

4.50. Using `xref`

Assume that this fragment appears somewhere in a document that includes the `id` example:

```
<para>More information can be found
  in <xref linkend="chapter1">.</para>

<para>More specific information can be found
  in <xref linkend="chapter1-sect1">.</para>
```

The text of the link will be generated automatically, and will look like (*emphasized* text indicates the text that will be the link):

> More information can be found in *Chapter One*.
>
> More specific information can be found in *the section called Subsect 1*.

Notice how the text from the link is derived from the section title or the chapter number.

##

This means that you *cannot* use `xref` to link to an `id` attribute on an `anchor` element. The `anchor` has no content, so the `xref` cannot generate the text for the link.

If you want to control the text of the link then use `link`. This element wraps content, and the content will be used for the link.

4.51. Using `link`

Assume that this fragment appears somewhere in a document that includes the `id` example.

```
<para>More information can be found in
  <link linkend="chapter1">the first chapter</link>.</para>

<para>More specific information can be found in
  <link linkend="chapter1-sect1">this</link> section.</para>
```

This will generate the following (*emphasized* text indicates the text that will be the link):

> More information can be found in *the first chapter*.
>
> More specific information can be found in *this* section.

##

That last one is a bad example. Never use words like “this” or “here” as the source for the link. The reader will need to hunt around the surrounding context to see where the link is actually taking them.

##

You *can* use `link` to include a link to an `id` on an `anchor` element, since the `link` content defines the text that will be used for the link.

4.2.7.2. Linking to documents on the WWW

Linking to external documents is much simpler, as long as you know the URL of the document you want to link to. Use `ulink`. The `url` attribute is the URL of the page that the link points to, and the content of the element is the text that will be displayed for the user to activate.

4.52. `ulink`

Use:

```
<para>Of course, you could stop reading this document and
  go to the <ulink url="&url.base;/index.html">FreeBSD
  home page</ulink> instead.</para>
```

Appearance:

Of course, you could stop reading this document and go to the FreeBSD home page instead.

5. * Stylesheets

SGML says nothing about how a document should be displayed to the user, or rendered on paper. To do that, various languages have been developed to describe stylesheets, including DynaText, Panorama, SPICE, JSSS, FOSI, CSS, and DSSSL.

For DocBook, we are using stylesheets written in DSSSL. For HTML we are using CSS.

5.1. * DSSSL

The Documentation Project uses a slightly customized version of Norm Walsh's modular DocBook stylesheets.

These can be found in textproc/dsssl-docbook-modular.

The modified stylesheets are not in the ports system. Instead they are part of the Documentation Project source repository, and can be found in `doc/share/xml/freebsd.dsl` . It is well commented, and pending completion of this section you are encouraged to examine that file to see how some of the available options in the standard stylesheets have been configured in order to customize the output for the FreeBSD Documentation Project. That file also contains examples showing how to extend the elements that the stylesheet understands, which is how the FreeBSD specific elements have been formatted.

5.2. CSS

Cascading Stylesheets (CSS) are a mechanism for attaching style information (font, weight, size, color, and so forth) to elements in an HTML document without abusing HTML to do so.

5.2.1. The Web site (HTML documents)

The FreeBSD web site does not currently use CSS. Unfortunately, the look and feel is constructed using abuses of HTML of varying degrees. This should be fixed, and would be a good project for someone looking to contribute to the documentation project.

5.2.2. The DocBook documents

The FreeBSD DSSSL stylesheets include a reference to a stylesheet, `docbook.css` , which is expected to appear in the same directory as the HTML files. The project-wide CSS file is copied from `doc/share/misc/docbook.css` when documents are converted to HTML, and is installed automatically.

6. Structuring documents under doc/

The doc/ tree is organized in a particular fashion, and the documents that are part of the FDP are in turn organized in a particular fashion. The aim is to make it simple to add new documentation into the tree and:

1. make it easy to automate converting the document to other formats;

2. promote consistency between the different documentation organizations, to make it easier to switch between working on different documents;

3. make it easy to decide where in the tree new documentation should be placed.

In addition, the documentation tree has to accommodate documentation that could be in many different languages and in many different encodings. It is important that the structure of the documentation tree does not enforce any particular defaults or cultural preferences.

6.1. The top level, doc/

There are two types of directory under doc/, each with very specific directory names and meanings.
Directory: share/
Meaning: Contains files that are not specific to the various translations and encodings of the documentation. Contains subdirectories to further categorize the information. For example, the files that comprise the make(1) infrastructure are in share/mk, while the additional SGML support files (such as the FreeBSD extended DocBook DTD) are in share/xml.
Directory: lang.encoding/
Meaning: One directory exists for each available translation and encoding of the documentation, for example en_US.ISO8859-1/ and zh_TW.UTF-8/. The names are long, but by fully specifying the language and encoding we prevent any future headaches should a translation team want to provide the documentation in the same language but in more than one encoding. This also completely isolates us from any problems that might be caused by a switch to Unicode.

6.2. The lang.encoding/ directories

These directories contain the documents themselves. The documentation is split into up to three more categories at this level, indicated by the different directory names.

Directory: `articles`
Contents: Documentation marked up as a DocBook `article` (or equivalent). Reasonably short, and broken up into sections. Normally only available as one HTML file.
Directory: `books`
Contents: Documentation marked up as a DocBook `book` (or equivalent). Book length, and broken up into chapters. Normally available as both one large HTML file (for people with fast connections, or who want to print it easily from a browser) and as a collection of linked, smaller files.
Directory: `man`
Contents: For translations of the system manual pages. This directory will contain one or more `mann` directories, corresponding to the sections that have been translated.

Not every `lang.encoding` directory will contain all of these directories. It depends on how much translation has been accomplished by that translation team.

6.3. Document specific information

This section contains specific notes about particular documents managed by the FDP.

6.3.1. The Handbook

`books/handbook/`

The Handbook is written to comply with the FreeBSD DocBook extended DTD.

The Handbook is organized as a DocBook `book`. It is then divided into `part`s, each of which may contain several `chapter` s. `chapter` s are further subdivided into sections (`sect1`) and subsections (`sect2`, `sect3`) and so on.

6.3.1.1. Physical organization

There are a number of files and directories within the `handbook` directory.

##

The Handbook's organization may change over time, and this document may lag in detailing the organizational changes. If you have any questions about how the Handbook is organized, please contact the FreeBSD documentation project *####*.

6.3.1.1.1. Makefile

The Makefile defines some variables that affect how the SGML source is converted to other formats, and lists the various source files that make up the Handbook. It then includes the standard doc.project.mk file, to bring in the rest of the code that handles converting documents from one format to another.

6.3.1.1.2. book.xml

This is the top level document in the Handbook. It contains the Handbook's DOCTYPE declaration, as well as the elements that describe the Handbook's structure.

book.xml uses parameter entities to load in the files with the .ent extension. These files (described later) then define general entities that are used throughout the rest of the Handbook.

6.3.1.1.3. directory/chapter.xml

Each chapter in the Handbook is stored in a file called chapter.xml in a separate directory from the other chapters. Each directory is named after the value of the id attribute on the chapter element.

For example, if one of the chapter files contains:

```
<chapter id="kernelconfiguration">
...
</chapter>
```

then it will be called chapter.xml in the kernelconfiguration directory. In general, the entire contents of the chapter will be held in this file.

When the HTML version of the Handbook is produced, this will yield kernelconfiguration.html . This is because of the id value, and is not related to the name of the directory.

In earlier versions of the Handbook the files were stored in the same directory as book.xml , and named after the value of the id attribute on the file's chapter element. Moving them into separate directories prepares for future plans for the Handbook. Specifically, it will soon be possible to include images in each chapter. It makes more sense for each image to be stored in a directory with the text for the chapter than to try to keep the text for all the chapters, and all the images, in one large directory. Namespace collisions would be inevitable, and it is easier to work with several directories with a few files in them than it is to work with one directory that has many files in it.

A brief look will show that there are many directories with individual chapter.xml files, including basics/chapter.xml , introduction/chapter.xml , and printing/chapter.xml .

##

Chapters and/or directories should not be named in a fashion that reflects their ordering within the Handbook. This ordering might change as the content within the Handbook is reorganized; this sort of reorganization should not (generally) include the need to rename files (unless entire chapters are being promoted or demoted within the hierarchy).

Each `chapter.xml` file will not be a complete SGML document. In particular, they will not have their own DOCTYPE lines at the start of the files.

This is unfortunate as it makes it impossible to treat these as generic SGML files and simply convert them to HTML, RTF, PS, and other formats in the same way the main Handbook is generated. This *would* force you to rebuild the Handbook every time you want to see the effect a change has had on just one chapter.

7. The Documentation Build Process

This chapter's main purpose is to clearly explain *how the documentation build process is organized*, and *how to affect modifications to this process.*

After you have finished reading this chapter you should:

- Know what you need to build the FDP documentation, in addition to those mentioned in the SGML tools chapter.
- Be able to read and understand the make instructions that are present in each document's `Makefile` s, as well as an overview of the `doc.project.mk` includes.
- Be able to customize the build process by using make variables and make targets.

7.1. The FreeBSD Documentation Build Toolset

Here are your tools. Use them every way you can.

- The primary build tool you will need is make, but specifically Berkeley Make.
- Package building is handled by FreeBSD's pkg_create. If you are not using FreeBSD, you will either have to live without packages, or compile the source yourself.
- gzip is needed to create compressed versions of the document. bzip2 compression and zip archives are also supported. tar is supported, but package building demands it.
- install is the default method to install the documentation. There are alternatives, however.

##

It is unlikely you will have any trouble finding these last two, they are mentioned for completeness only.

7.2. Understanding Makefiles in the Documentation tree

There are three main types of `Makefile` s in the FreeBSD Documentation Project tree.

- Subdirectory `Makefile`s simply pass commands to those directories below them.
- Documentation `Makefile`s describe the document(s) that should be produced from this directory.
- Make includes are the glue that perform the document production, and are usually of the form `doc.xxx.mk`.

7.2.1. Subdirectory Makefiles

These `Makefile`s usually take the form of:

```
SUBDIR =articles
SUBDIR+=books

COMPAT_SYMLINK = en

DOC_PREFIX?= ${.CURDIR}/..
.include "${DOC_PREFIX}/share/mk/doc.project.mk"
```

In quick summary, the first four non-empty lines define the make variables, `SUBDIR`, `COMPAT_SYMLINK`, and `DOC_PREFIX`.

The first `SUBDIR` statement, as well as the `COMPAT_SYMLINK` statement, shows how to assign a value to a variable, overriding any previous value.

The second `SUBDIR` statement shows how a value is appended to the current value of a variable. The `SUBDIR` variable is now `articles books`.

The `DOC_PREFIX` assignment shows how a value is assigned to the variable, but only if it is not already defined. This is useful if `DOC_PREFIX` is not where this `Makefile` thinks it is - the user can override this and provide the correct value.

Now what does it all mean? `SUBDIR` mentions which subdirectories below this one the build process should pass any work on to.

`COMPAT_SYMLINK` is specific to compatibility symlinks (amazingly enough) for languages to their official encoding (`doc/en` would point to `en_US.ISO-8859-1`).

`DOC_PREFIX` is the path to the root of the FreeBSD Document Project tree. This is not always that easy to find, and is also easily overridden, to allow for flexibility. `.CURDIR` is a make builtin variable with the path to the current directory.

The final line includes the FreeBSD Documentation Project's project-wide make system file `doc.project.mk` which is the glue which converts these variables into build instructions.

7.2.2. Documentation Makefiles

These `Makefile`s set a bunch of make variables that describe how to build the documentation contained in that directory.

Here is an example:

```
MAINTAINER=nik@FreeBSD.org

DOC?= book

FORMATS?= html-split html

INSTALL_COMPRESSED?= gz
INSTALL_ONLY_COMPRESSED?=

# SGML content
SRCS=  book.xml

DOC_PREFIX?= ${.CURDIR}/../../..

.include "$(DOC_PREFIX)/share/mk/docproj.docbook.mk"
```

The `MAINTAINER` variable is a very important one. This variable provides the ability to claim ownership over a document in the FreeBSD Documentation Project, whereby you gain the responsibility for maintaining it.

`DOC` is the name (sans the `.xml` extension) of the main document created by this directory. `SRCS` lists all the individual files that make up the document. This should also include important files in which a change should result in a rebuild.

`FORMATS` indicates the default formats that should be built for this document. `INSTALL_COMPRESSED` is the default list of compression techniques that should be used in the document build. `INSTALL_ONLY_COMPRESS` , empty by default, should be non-empty if only compressed documents are desired in the build.

##

We covered optional variable assignments in the previous section.

The `DOC_PREFIX` and include statements should be familiar already.

7.3. FreeBSD Documentation Project make includes

This is best explained by inspection of the code. Here are the system include files:

- `doc.project.mk` is the main project include file, which includes all the following include files, as necessary.
- `doc.subdir.mk` handles traversing of the document tree during the build and install processes.
- `doc.install.mk` provides variables that affect ownership and installation of documents.
- `doc.docbook.mk` is included if `DOCFORMAT` is `docbook` and `DOC` is set.

7.3.1. doc.project.mk

By inspection:

```
DOCFORMAT?= docbook
MAINTAINER?= doc@FreeBSD.org

PREFIX?= /usr/local
PRI_LANG?= en_US.ISO8859-1

.if defined(DOC)
.if ${DOCFORMAT} == "docbook"
.include "doc.docbook.mk"
.endif
.endif

.include "doc.subdir.mk"
.include "doc.install.mk"
```

7.3.1.1. Variables

`DOCFORMAT` and `MAINTAINER` are assigned default values, if these are not set by the document make file.

`PREFIX` is the prefix under which the documentation building tools are installed. For normal package and port installation, this is `/usr/local` .

`PRI_LANG` should be set to whatever language and encoding is natural amongst users these documents are being built for. US English is the default.

##

PRI_LANG in no way affects what documents can, or even will, be built. Its main use is creating links to commonly referenced documents into the FreeBSD documentation install root.

7.3.1.2. Conditionals

The `.if defined(DOC)` line is an example of a make conditional which, like in other programs, defines behavior if some condition is true or if it is false. `defined` is a function which returns whether the variable given is defined or not.

`.if ${DOCFORMAT} == "docbook"` , next, tests whether the `DOCFORMAT` variable is `"docbook"` , and in this case, includes `doc.docbook.mk` .

The two `.endif`s close the two above conditionals, marking the end of their application.

7.3.2. doc.subdir.mk

This is too long to explain by inspection, you should be able to work it out with the knowledge gained from the previous chapters, and a little help given here.

7.3.2.1. Variables

- `SUBDIR` is a list of subdirectories that the build process should go further down into.
- `ROOT_SYMLINKS` is the name of directories that should be linked to the document install root from their actual locations, if the current language is the primary language (specified by `PRI_LANG`).
- `COMPAT_SYMLINK` is described in the Subdirectory Makefile section.

7.3.2.2. Targets and macros

Dependencies are described by `target: dependency1 dependency2 ...` tuples, where to build `target` , you need to build the given dependencies first.

After that descriptive tuple, instructions on how to build the target may be given, if the conversion process between the target and its dependencies are not previously defined, or if this particular conversion is not the same as the default conversion method.

A special dependency `.USE` defines the equivalent of a macro.

```
_SUBDIRUSE: .USE
```

```
.for entry in ${SUBDIR}
 @${ECHO} "===> ${DIRPRFX}${entry}"
 @(cd ${.CURDIR}/${entry} && \
 ${MAKE} ${.TARGET:S/realpackage/package/:S/realinstall/install/} ↺
DIRPRFX=${DIRPRFX}${entry}/ )
.endfor
```

In the above, `_SUBDIRUSE` is now a macro which will execute the given commands when it is listed as a dependency.

What sets this macro apart from other targets? Basically, it is executed *after* the instructions given in the build procedure it is listed as a dependency to, and it does not adjust `.TARGET`, which is the variable which contains the name of the target currently being built.

```
clean: _SUBDIRUSE
 rm -f ${CLEANFILES}
```

In the above, `clean` will use the `_SUBDIRUSE` macro after it has executed the instruction `rm -f ${CLEANFILES}`. In effect, this causes `clean` to go further and further down the directory tree, deleting built files as it goes *down*, not on the way back up.

7.3.2.2.1. Provided targets

- `install` and `package` both go down the directory tree calling the real versions of themselves in the subdirectories (`realinstall` and `realpackage` respectively).
- `clean` removes files created by the build process (and goes down the directory tree too). `cleandir` does the same, and also removes the object directory, if any.

7.3.2.3. More on conditionals

- `exists` is another condition function which returns true if the given file exists.
- `empty` returns true if the given variable is empty.
- `target` returns true if the given target does not already exist.

7.3.2.4. Looping constructs in make (.for)

`.for` provides a way to repeat a set of instructions for each space-separated element in a variable. It does this by assigning a variable to contain the current element in the list being examined.

```
_SUBDIRUSE: .USE
.for entry in ${SUBDIR}
 @${ECHO} "===> ${DIRPRFX}${entry}"
 @(cd ${.CURDIR}/${entry} && \
 ${MAKE} ${.TARGET:S/realpackage/package/:S/realinstall/install/} ↺
DIRPRFX=${DIRPRFX}${entry}/ )
```

```
.endfor
```

In the above, if `SUBDIR` is empty, no action is taken; if it has one or more elements, the instructions between `.for` and `.endfor` would repeat for every element, with `entry` being replaced with the value of the current element.

8. ## Website

8.1.

200MB ########## SGML #####CVS tree# ########################## SGML # #### CVS tree ########## 100MB

##

################## ports ###### ################# pkg_delete(1) ######## ##### port# ########## jade-1.1# ######## ### jade-1.2################

```
# pkg_delete jade-1.1
```

####### CVS repository##### www, doc, ports ### CVS tree(###### CVSROOT)# ### CVSup ## ###### mirror a CVS tree ### CVS tree#

cvsup collections ##www, doc-all , cvs-base ## ports-base

######## 105MB ###

CVS tree - ## src, doc, www ## ports - #### 940MB#

8.2. Build the web pages from scratch

1. #########(#### 60MB ##)#########

   ```
   # mkdir /var/tmp/webbuild
   # cd /var/tmp/webbuild
   ```

2. # CVS tree # checkout ### SGML ##

   ```
   # cvs -R co www doc
   ```

3. ## www/en ###### make(1) all ######

   ```
   # cd en
   # make all
   ```

8.3.

1. ####### en ###############

   ```
   # cd path/www/en
   ```

2. ## make(1) install # ## DESTDIR ###############

   ```
   # make DESTDIR=/usr/local/www install
   ```

3. ###################### ################### ###################### ####### ################

   ```
   # find /usr/local/www -ctime 3 -print0 | xargs -0 rm
   ```

8.4.

CVSROOT
: ## CVS tree ###########

  ```
  # CVSROOT=/home/ncvs; export CVSROOT
  ```

ENGLISH_ONLY
: ################### makefiles ############ ################

  ```
  # make ENGLISH_ONLY=YES all install
  ```

  ######### ENGLISH_ONLY ################### ENGLISH_ONLY ##########

  ```
  # make ENGLISH_ONLY="" all install clean
  ```

WEB_ONLY
: ############ makefiles #### www ####### HTML ### ### doc ############# (Handbook, FAQ, Tutorials)# ###

  ```
  # make WEB_ONLY=YES all install
  ```

NOPORTSCVS
: #########makefiles #### ports cvs repository ##### ###### /usr/ports (## PORTSBASE #####) ######

CVSROOT ###### ############ dot files (## ~/.profile) # #########

WEB_ONLY #ENGLISH_ONLY # NOPORTSCVS ## makefile ### #### /etc/make.conf #Makefile.inc ################### dot files ##########

9.

FreeBSD ##(###FAQ, Handbook, tutorials, manual pages#)#####(FAQ)#

FreeBSD ######### FAQ ####### ###### Frank Gründer <elwood@mc5sys.in-berlin.de > ### Bernd Warken <bwarken@mayn.de >

The FAQ is maintained by the Documentation Engineering Team <doceng@FreeBSD.org >.

FAQ ####?

freebsd-doc ########## FreeBSD ############ ###### FAQ ##########

i18n # l10n

i18n # internationalization ##### l10n ## localization

i18n ##### “i” ### 18 ####### “n”# ### l10n ##### “l” ### 10 ####### “n”#

mailing list

mailing lists### ###### ########### mailing lists

#############

#############

##################

###########

######################################

####################(Spanish)# FAQ #######(Hungarian)#

############

FreeBSD CVS repository ###(######)#### CTM # CVSup ## ##Handbook ## "##### FreeBSD"

####### CVS ### ####################

[XXX To Do(#########) -- ######(tutorial)###### CVSup ####################]

##################

######

###################################### FreeBSD documentation project ####
##

##################

“FreeBSD #### ######”

############################## ##

FreeBSD documentation project ####

########### FreeBSD # mirror(##) ############## ######################### ######### email ### mailing list ###

################################### —— ## FAQ ################

################

############(#######)#

############(################### FreeBSD ##) ################# FreeBSD ### (########)

########################

##

########################

####################

###FreeBSD ########## doc/ #### ##################### ISO639 ##(/usr/share/misc/iso639 ### FreeBSD ### 1999/01/20 ##)#

################(#####) ##########################

#################

###########(Swedish)#############

```
doc/
    sv_SE.ISO8859-1/
                    Makefile
                    books/
                          faq/
                              Makefile
                              book.xml
```

sv_SE.ISO8859-1 ### ##(lang).##(encoding) ########## ######### Makefiles ###########

tar(1) # gzip(1)

```
% cd doc
% tar cf swedish-docs.tar sv_SE.ISO8859-1
% gzip -9 swedish-docs.tar
```

swedish-docs.tar.gz ####################(ISP###) ######### Documentation Engineering Team <doceng@FreeBSD.org >

send-pr(1) ################## ########################## ########

######(############ Documentation Engineering Team <doceng@FreeBSD.org > ##) ###################################

1. ######### RCS tag (## "ID" ###)#

2. sv_SE.ISO8859-1 ###### make all ####

3. make install ########

################################

#################### commit ####

########################

##########

############ Handbook ####### ################ Handbook #####

############################(############# ...) ################## FreeBSD ##### ######### FreeBSD ##################

############(# send-pr(1))###### Handbook ##### ##################### Handbook ##

######

####################

ASCII(Non-ASCII) ####### SGML entities

############ & ##(&)##### entity #########(;)#

entity #### ISO8879 ###### port tree ### textproc/iso8879#

########
Entity##: é
####: é

##: # “e”#######(acute accent)
Entity##: É
####: É
##: # “E”#######(acute accent)
Entity##: ü
####: ü
##: # “u”##############(umlaut)

iso8879 ## port ####### /usr/local/share/xml/iso8879

########

“you” ##############/#######

########################

##################

####

######################

```
<!--
     The FreeBSD Documentation Project

     $FreeBSD: doc/en_US.ISO8859-1/books/fdp-primer/translations/
chapter.xml,v 1.5 2000/07/07 18:38:38 dannyboy Exp $
-->
```

###################### $FreeBSD$ ##### The FreeBSD Documentation Project ### ####$FreeBSD ######## CVS ############ #############(###### $FreeBSD $ ###)#

########### $FreeBSD$ ###### FreeBSD Documentation Project #### The FreeBSD #### Documentation Project #

##

#########(Spanish)#############

```
<!--
     The FreeBSD Spanish Documentation Project

     $FreeBSD: doc/es_ES.ISO8859-1/books/fdp-primer/translations/
chapter.xml,v 1.3 1999/06/24 19:12:32 jesusr Exp $
     Original revision: 1.11
-->
```

10.

FreeBSD #########################

######

#################### ################### ### ### “color”### “colour”# ### “rationalize”### “rationalise” #######

> ##
>
> ############################### # ################ manual #############

#####

#####(contraction)# ############ ### “Don't use contractions” #############

############################## ###########

serial comma

#########(,)################ ######################## “and”# ##########

############

This is a list of one, two and three items.

#############(“one”#“two” #“three”)###########(“one”# “two and three”)##

######### serial comma #############

This is a list of one, two, and three items.

###############(,)######(#)### “and” ################

######

#########(redundant phrase)# ### “####”#“####”#“man ##” ##############

###(command)####################

`cvsup`

`cvsup`

###(filename)####################

… ### `/etc/rc.local` ## …

... # /etc/rc.local # ...

man(manual)#################(### SGML citerefentry ##)#

man csh

csh(1)#

##########
############ Emacs ############################

#####(.)######### ##################### ############### ## "Jordan K. Hubbard"
######################### H##############

############ William Strunk ### Elements of Style#

10.1. Style guide

Handbook ######################

10.1.1.

Tag ############### <para> ### <PARA> #

SGML ############### <!ENTITY...> # <!DOCTYPE...> # ### <!entity...> # <!doctype...>

10.1.2.

###(acronym)####################### ###"Network Time Protocol (NTP)"# ######### ##########(####### ################)###### ###################### ############ ############

################### <acronym> ### ######## role ####### ######################## #### #########

10.1.3.

##########(indentation)### 0

################# ################ ### 8 ###### tab #### #### tab ############## ######## ## tag ############################

####################

```
+--- ## 0 ##
V
<chapter>
```

```
  <title>...</title>

  <sect1>
    <title>...</title>

    <sect2>
      <title>##</title>

      <para><emphasis>##</emphasis> #########
        ##########(indentation)### 0 #####</para>

      ...
    </sect2>
  </sect1>
</chapter>
```

Emacs # XEmacs ############ sgml-mode ###

Vim ###############

```
augroup sgmledit
  autocmd FileType sgml set formatoptions=cq2l " ######
  autocmd FileType sgml set textwidth=70       " # 70 ########
  autocmd FileType sgml set shiftwidth=2       " #### 2 ###
  autocmd FileType sgml set softtabstop=2      " # Tab ############
  autocmd FileType sgml set tabstop=8          " # 8 ##### tab
  autocmd FileType sgml set autoindent         " ####
augroup END
```

10.1.4. Tag

10.1.4.1. Tag

############################### ###

```
<article lang='zh_tw'>
  <articleinfo>
    <title>NIS</title>

    <pubdate>October 1999</pubdate>

    <abstract>
      <para>...
 ...
 ...</para>
    </abstract>
  </articleinfo>

  <sect1>
    <title>...</title>

    <para>...</para>
```

```
  </sect1>

  <sect1>
    <title>...</title>

    <para>...</para>
  </sect1>
</article>
```

10.1.4.2.

itemizedlist ##################################

para # term ############## ##################

######################

########################

################## ############### ######## ############

############################

10.1.5.

commit ##############

####### Handbook ################# ############################

################################ 80 ####### commmit ### ################## commit ## ##### commit ########## whitespace-only (######) ################## # commit # #

10.1.6. Nonbreaking space

################################ ################## ############ HTML ### #################

```
Data capacity ranges from 40 MB to 15
GB.  Hardware compression …
```

nonbreaking spaces#

- #########

  ```
  57600 bps
  ```

- ###########

  ```
  FreeBSD 4.7
  ```

- multiword ## (######### “The FreeBSD Brazilian Portuguese Documentation Project” ############ #####)#

```
Sun Microsystems
```

10.2.

FreeBSD ################# ############# O'Reilly word list#

- 2.2.X
- 4.X-STABLE
- CD-ROM
- DoS *(Denial of Service)*
- Ports Collection
- IPsec
- Internet
- MHz
- Soft Updates
- Unix
- disk label
- email
- file system
- manual page
- mail server
- name server
- null-modem
- web server

11. Using `sgml-mode` with Emacs

Recent versions of Emacs or XEmacs (available from the ports collection) contain a very useful package called PSGML. Automatically invoked when a file with the `.xml` extension is loaded, or by typing `M-x sgml-mode` , it is a major mode for dealing with SGML files, elements and attributes.

An understanding of some of the commands provided by this mode can make working with SGML documents such as the Handbook much easier.

`C-c C-e`
: Runs `sgml-insert-element` . You will be prompted for the name of the element to insert at the current point. You can use the TAB key to complete the element. Elements that are not valid at the current point will be disallowed.

  The start and end tags for the element will be inserted. If the element contains other, mandatory, elements then these will be inserted as well.

`C-c =`
: Runs `sgml-change-element-name` . Place the point within an element and run this command. You will be prompted for the name of the element to change to. Both the start and end tags of the current element will be changed to the new element.

`C-c C-r`
: Runs `sgml-tag-region` . Select some text (move to start of text, C-space, move to end of text, C-space) and then run this command. You will be prompted for the element to use. This element will then be inserted immediately before and after your marked region.

`C-c -`
: Runs `sgml-untag-element` . Place the point within the start or end tag of an element you want to remove, and run this command. The element's start and end tags will be removed.

`C-c C-q`
: Runs `sgml-fill-element` . Will recursively fill (i.e., reformat) content from the current element in. The filling *will* affect content in which whitespace is significant, such as within `programlisting` elements, so run this command with care.

`C-c C-a`
: Runs `sgml-edit-attributes` . Opens a second buffer containing a list of all the attributes for the closest enclosing element, and their current values. Use TAB to navigate between attributes, `C-k` to remove an existing value and replace it with a new one, `C-c C-c` to close this buffer and return to the main document.

`C-c C-v`
: Runs `sgml-validate` . Prompts you to save the current document (if necessary) and then runs an SGML validator. The output from the validator is captured into a new

buffer, and you can then navigate from one troublespot to the next, fixing markup errors as you go.

`C-c /`
: Runs `sgml-insert-end-tag` . Inserts the end tag for the current open element.

Doubtless there are other useful functions of this mode, but those are the ones I use most often.

You can also use the following entries in `.emacs` to set proper spacing, indentation, and column width for working with the Documentation Project.

```
(defun local-sgml-mode-hook
  (setq fill-column 70
        indent-tabs-mode nil
        next-line-add-newlines nil
        standard-indent 4
        sgml-indent-data t)
  (auto-fill-mode t)
  (setq sgml-catalog-files '("/usr/local/share/xml/catalog")))
(add-hook 'psgml-mode-hook
  '(lambda () (local-psgml-mode-hook)))
```

12.

This document is deliberately not an exhaustive discussion of SGML, the DTDs listed, and the FreeBSD Documentation Project. For more information about these, you are encouraged to see the following web sites.

12.1. The FreeBSD Documentation Project

- The FreeBSD Documentation Project web pages
- The FreeBSD Handbook

12.2. SGML

- The SGML/XML web page, a comprehensive SGML resource
- Gentle introduction to SGML

12.3. HTML

- The World Wide Web Consortium
- The HTML 4.0 specification

12.4. DocBook

- The DocBook Technical Committee, maintainers of the DocBook DTD
- DocBook: The Definitive Guide, the online documentation for the DocBook DTD.
- The DocBook Open Repository contains DSSSL stylesheets and other resources for people using DocBook.

12.5. The Linux Documentation Project

- The Linux Documentation Project web pages

A.

####### SGML ################## ################################

########## —— ############ ##########(##################) ###### DocBook ##### ########## CSup#CVSup #### doc tree ############## SGML ### ########## `http://www.FreeBSD.org/cgi/cvsweb.cgi/doc/` #

#################### DocBook 4.1 DTD ## FreeBSD ### DTD# ##### Norm Walsh ##### (stylesheets)### FreeBSD ############## ##### DocBook ########################

A.1. DocBook book

A.1. DocBook book

```
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook V4.1//EN">

<book lang='zh_tw'>
  <bookinfo>
    <title>######</title>

    <author>
      <firstname>#(first name)</firstname>
      <surname>#(surname)</surname>
      <affiliation>
        <address><email>foo@example.com</email></address>
      </affiliation>
    </author>

    <copyright>
      <year>2000</year>
      <holder>######</holder>
    </copyright>

    <abstract>
      <para>#############</para>
    </abstract>
  </bookinfo>

  <preface>
    <title>##</title>

    <para>#############</para>
  </preface>

  <chapter>
    <title>###</title>
```

```
    <para>##########</para>

    <sect1>
      <title>###</title>

      <para>########</para>
    </sect1>
  </chapter>
</book>
```

A.2. DocBook article

A.2. DocBook article

```
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook V4.1//EN">

<article lang='zh_tw'>
  <articleinfo>
    <title>####</title>

    <author>
      <firstname>#(first name)</firstname>
      <surname>#(surname)</surname>
      <affiliation>
        <address><email>foo@example.com</email></address>
      </affiliation>
    </author>

    <copyright>
      <year>2000</year>
      <holder>######</holder>
    </copyright>

    <abstract>
      <para>###############</para>
    </abstract>
  </articleinfo>

  <sect1>
    <title>###</title>

    <para>########</para>

    <sect2>
      <title>####(sub-section)</title>
```

```
      <para>########(sub-section)</para>
    </sect2>
  </sect1>
</article>
```

A.3. Producing formatted output

############## textproc/docproj ############### port ########## ############ / usr/local/ ###### ################## PATH ######### ######################

A.3.1. ## Jade

A.3. ## DocBook # HTML (####)

```
% jade -V nochunks \   ❶
    -c /usr/local/share/xml/docbook/dsssl/modular/catalog \    ❷
    -c /usr/local/share/xml/docbook/catalog \
    -c /usr/local/share/xml/jade/catalog \
    -d /usr/local/share/xml/docbook/dsssl/modular/html/docbook.↺
dsl \❸
    -t sgml ❹ file.xml > file.html  ❺
```

❶ Specifies the nochunks parameter to the stylesheets, forcing all output to be written to STDOUT (using Norm Walsh's stylesheets).

❷ Specifies the catalogs that Jade will need to process. Three catalogs are required. The first is a catalog that contains information about the DSSSL stylesheets. The second contains information about the DocBook DTD. The third contains information specific to Jade.

❸ Specifies the full path to the DSSSL stylesheet that Jade will use when processing the document.

❹ Instructs Jade to perform a *transformation* from one DTD to another. In this case, the input is being transformed from the DocBook DTD to the HTML DTD.

❺ Specifies the file that Jade should process, and redirects output to the specified .html file.

A.4. ## DocBook # HTML (####)

```
% jade \
```

```
    -c /usr/local/share/xml/docbook/dsssl/modular/catalog \    ❶
    -c /usr/local/share/xml/docbook/catalog \
    -c /usr/local/share/xml/jade/catalog \
    -d /usr/local/share/xml/docbook/dsssl/modular/html/docbook.↺
dsl \❷
    -t sgml ❸ file.xml ❹
```

❶ Specifies the catalogs that Jade will need to process. Three catalogs are required. The first is a catalog that contains information about the DSSSL stylesheets. The second contains information about the DocBook DTD. The third contains information specific to Jade.

❷ Specifies the full path to the DSSSL stylesheet that Jade will use when processing the document.

❸ Instructs Jade to perform a *transformation* from one DTD to another. In this case, the input is being transformed from the DocBook DTD to the HTML DTD.

❹ Specifies the file that Jade should process. The stylesheets determine how the individual HTML files will be named, and the name of the "root" file (i.e., the one that contains the start of the document.

This example may still only generate one HTML file, depending on the structure of the document you are processing, and the stylesheet's rules for splitting output.

A.5. ## DocBook # Postscript(PS)

The source SGML file must be converted to a TeX file.

```
% jade -Vtex-backend \  ❶
    -c /usr/local/share/xml/docbook/dsssl/modular/catalog \    ❷
    -c /usr/local/share/xml/docbook/catalog \
    -c /usr/local/share/xml/jade/catalog \
    -d /usr/local/share/xml/docbook/dsssl/modular/print/
docbook.dsl \❸
    -t tex ❹ file.xml
```

❶ Customizes the stylesheets to use various options specific to producing output for TeX.

❷ Specifies the catalogs that Jade will need to process. Three catalogs are required. The first is a catalog that contains information about the DSSSL stylesheets. The second contains information about the DocBook DTD. The third contains information specific to Jade.

❸ Specifies the full path to the DSSSL stylesheet that Jade will use when processing the document.

❹ Instructs Jade to convert the output to TeX.

The generated `.tex` file must now be run through `tex`, specifying the `&jadetex` macro package.

```
% tex "&jadetex" file.tex
```

You have to run `tex` *at least* three times. The first run processes the document, and determines areas of the document which are referenced from other parts of the document, for use in indexing, and so on.

Do not be alarmed if you see warning messages such as `LaTeX Warning: Reference `136' on page 5 undefined on input line 728.` at this point.

The second run reprocesses the document now that certain pieces of information are known (such as the document's page length). This allows index entries and other cross-references to be fixed up.

The third pass performs any final cleanup necessary.

The output from this stage will be `file.dvi`.

Finally, run `dvips` to convert the `.dvi` file to Postscript.

```
% dvips -o file.ps file.dvi
```

A.6. ## DocBook # PDF

The first part of this process is identical to that when converting DocBook to Postscript, using the same `jade` command line (## A.5, "## DocBook # Postscript(PS) ##").

When the `.tex` file has been generated you run pdfTeX. However, use the &pdfjadetex macro package instead.

```
% pdftex "&pdfjadetex" file.tex
```

Again, run this command three times.

This will generate `file.pdf`, which does not need to be processed any further.

##

F

Formal Public Identifier, 17, 18

M

Membership, 1