FreeBSD Documentation Project Primer for New Contributors

The FreeBSD Documentation Project

FreeBSD Documentation Project Primer for New Contributors

by The FreeBSD Documentation Project

Published $FreeBSD: head/en_US.ISO8859-1/books/fdp-primer/book.xml 42065 2013-06-26 15:40:23Z wblock $

Thank you for becoming a part of the FreeBSD Documentation Project. Your contribution is extremely valuable, and we appreciate it.

This primer covers details needed to start contributing to the FreeBSD Documentation Project, or FDP, including tools, software, and the philosophy behind the Documentation Project.

This is a work in progress. Corrections and additions are always welcome.

Copyright

Table of Contents

List of Examples

Preface

Shell Prompts

This table shows the default system prompt and superuser prompt. The examples use these prompts to indicate which type of user is running the example.

User	Prompt
Normal user	`%`
`root`	`#`

Typographic Conventions

This table describes the typographic conventions used in this book.

Meaning	Examples
The names of commands.	Use `ls -l` to list all files.
The names of files.	Edit `.login`.
On-screen computer output.	`You have mail.`
What the user types, contrasted with on-screen computer output.	`% date +"The time is %H:%M"` `The time is 09:18`
Manual page references.	Use su(1) to change user identity.
User and group names.	Only `root` can do this.
Emphasis.	The user *must* do this.
Text that the user is expected to replace with the actual text.	To search for a keyword in the manual pages, type `man -k` *`keyword`*
Environment variables.	`$HOME` is set to the user's home directory.

Notes, Tips, Important Information, Warnings, and Examples

Notes, warnings, and examples appear within the text.

> **Note:** Notes are represented like this, and contain information to take note of, as it may affect what the user does.

> **Tip:** Tips are represented like this, and contain information helpful to the user, like showing an easier way to do something.

Important: Important information is represented like this. Typically, these show extra steps the user may need to take.

Warning: Warnings are represented like this, and contain information warning about possible damage if the instructions are not followed. This damage may be physical, to the hardware or the user, or it may be non-physical, such as the inadvertent deletion of important files.

Example 1. A Sample Example

Examples are represented like this, and typically contain examples showing a walkthrough, or the results of a particular action.

Acknowledgments

My thanks to Sue Blake, Patrick Durusau, Jon Hamilton, Peter Flynn, and Christopher Maden, who took the time to read early drafts of this document and offer many valuable comments and criticisms.

Chapter 1 Overview

Welcome to the FreeBSD Documentation Project (FDP). Quality documentation is crucial to the success of FreeBSD, and we value your contributions very highly.

This document describes how the FDP is organized, how to write and submit documentation, and how to effectively use the available tools.

Everyone is welcome to contribute to the FDP. Willingness to contribute is the only membership requirement.

This primer shows the reader how to:

- Identify which parts of FreeBSD are maintained by the FDP.
- Install the required documentation tools and files.
- Make changes to the documentation.
- Submit changes back for review and eventual inclusion in the FreeBSD documentation.

1.1 The FreeBSD Documentation Set

The FDP is responsible for four categories of FreeBSD documentation.

- *Handbook*: The Handbook is the comprehensive online resource and reference for FreeBSD users.
- *FAQ*: The FAQ uses a short question and answer format to address questions that are frequently asked on the various mailing lists and forums devoted to FreeBSD. This format does not permit long and comprehensive answers.
- *Manual pages*: The English language system manual pages are usually not written by the FDP, as they are part of the base system. However, the FDP can reword parts of existing manual pages to make them clearer or to correct inaccuracies.
- *Web site*: This is the main FreeBSD presence on the web, visible at http://www.FreeBSD.org/ (http://www.freebsd.org/index.html) and many mirrors around the world. The web site is typically a new user's first exposure to FreeBSD.

Translation teams are responsible for translating the Handbook and web site into different languages. Manual pages are not translated.

Documentation source for the FreeBSD web site, Handbook, and FAQ is available in the Subversion repository at `https://svn.FreeBSD.org/doc/`.

Source for manual pages is available in a separate Subversion repository located at `https://svn.FreeBSD.org/base/`.

Documentation commit messages are visible with **svn**. Commit messages are also archived at http://lists.FreeBSD.org/mailman/listinfo/svn-doc-all.

In addition, many people have written tutorials or how-to articles about FreeBSD. Some are stored as part of the FDP files. In other cases, the author has decided to keep the documentation separate. The FDP endeavors to provide links to as much of this external documentation as possible.

1.2 Quick Start

Here we describe the steps contributors must take before making changes to the FDP. New contributors will interact with other members of the FreeBSD Documentation Team, which can assist in learning to use XML and the suggestions in Section 11.3. If a new user contributes regularly, a Documentation Team member may be assigned as a mentor to guide the user through the process from contributor to documentation committer.

1. Subscribe to the FreeBSD documentation project mailing list (http://lists.FreeBSD.org/mailman/listinfo/freebsd-doc). Some members of the mailing list also interact on the `#bsddocs` IRC channel on EFnet (http://www.efnet.org/).
2. Install the `textproc/docproj` package or port. This meta-port installs all of the utilities needed by the FDP.
3. Install a local working copy of the documentation from a mirror of the FreeBSD repository. For the fastest download, pick the nearest mirror from the list of Subversion mirror sites (http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/handbook/subversion-mirrors.html). If `/usr/doc` already exists, move or delete it first to prevent file conflicts.

   ```
   % svn checkout https://svn0.us-west.FreeBSD.org/doc/head /usr/doc
   ```

4. Before making any documentation edits, configure your editor to conform to FDP standards. How to do so varies by editor. Some editor configurations are listed in Chapter 11. The editor should be configured as follows:
   - Word wrap set to 70 characters.
   - Tab stops set to 2.
   - Replace each group of 8 leading spaces with a single tab.
5. Locate the file to edit. Run `svn up` within the local working copy to make sure that it is up to date. Before making major changes to a file, discuss the proposed changes with the FreeBSD documentation project mailing list (http://lists.FreeBSD.org/mailman/listinfo/freebsd-doc).

   When making edits, determine which tags and entities are needed to achieve the desired formatting. One way to learn is to compare some text in the HTML formatted version of the document to the tags which surround the text or the entities that represent that text in the XML file. References to the commonly used tags and entities can be found in Chapter 4 and Chapter 5.
6. After edits are complete, check for problems by running:

   ```
   % igor -R filename.xml | less -RS
   ```

   Review the output and edit the file to fix any listed tab errors, spelling mistakes, and improper grammar. Save the changes and rerun this command to find any remaining problems. Repeat until all of the errors that you deem fixable are resolved. If you get stuck trying to fix errors, ask for assistance on the FreeBSD documentation project mailing list (http://lists.FreeBSD.org/mailman/listinfo/freebsd-doc).
7. *Always* build-test changes before submitting them. By default, typing **`make`** in the top-level directory of the type of documentation being edited will generate that documentation in split HTML format. For example, to build the English version of the Handbook, type `make` in the `en_US.ISO8859-1/books/handbook/` directory. This step is necessary to make sure that the edits do not break the build.
8. In order to build the output in other formats, other **make** targets are defined in `head/share/mk/doc.docbook.mk`. Use quotes around the list of formats when building more than one format with a single command.

   For example, to convert the document to both `.html` and `.txt`, use this command:

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

Once these steps are successfully completed, generate a “diff file” of the changes. While in `/usr/doc`, run this command, replacing *`bsdinstall`* with the name of the directory containing the edits:

```
% svn diff > bsdinstall.diff.txt
```

9. Submit the diff file using the web-based Problem Report (http://www.FreeBSD.org/support.html#gnats) system or with send-pr(1). If using the web form, input a synopsis of *[patch] `short description of problem`*. Select the category `docs` and the class `doc-bug`. The body of the message should contain a short description of the edits and any important discussion points. Use the [Browse...] button to attach the `.diff.txt` file and enter the captcha phrase.

   It is important to remember that the FDP is comprised of volunteers who review edits in their spare time and who live in different time zones around the globe. It takes time to review edits and to either commit them or respond if additional edits are required. If you do not receive a response in a reasonable amount of time, send a follow-up email to the FreeBSD documentation project mailing list (http://lists.FreeBSD.org/mailman/listinfo/freebsd-doc) and ask if anyone has had a chance to review the patch or if additional information is required.

Chapter 2 Tools

The FDP uses a number of different software tools to help manage the FreeBSD documentation, convert it to different output formats, and so on. You will need to use these tools yourself if you are to work with the FreeBSD documentation.

All these tools are available as FreeBSD Ports and Packages, greatly simplifying the work you have to do to install them.

You will need to install these tools before you work through any of the examples in later chapters. The actual usage of these tools is covered in later chapters.

> **Use `textproc/docproj` If Possible:** You can save yourself a lot of time if you install the `textproc/docproj` port. This is a *meta-port* which does not contain any software itself. Instead, it depends on various other ports being installed correctly. Installing this port *should* automatically download and install all of the packages listed in this chapter that you need.
>
> One of the packages that you might need is the **JadeTeX** macro set. In turn, this macro set requires TEX to be installed. TEX is a large package, and you only need it if you want to produce Postscript or PDF output.
>
> To save yourself time and space you must specify whether or not you want **JadeTeX** (and therefore TEX) installed when you install this port. Either do:
>
> ```
> # make JADETEX=yes install
> ```
>
> or
>
> ```
> # make JADETEX=no install
> ```
>
> as necessary. Alternatively you may install `textproc/docproj-jadetex` or `textproc/docproj-nojadetex`. These slave ports define the `JADETEX` variable for you, therefore they will install the same suite of applications on your machine. Note that you can produce only XHTML or ASCII text output if you do not install **JadeTeX**. PostScript or PDF output requires TEX.

2.1 Mandatory Tools

2.1.1 Software

These programs are required before you can usefully work with the FreeBSD documentation, and they will allow you to convert the documentation to XHTML, plain text, and RTF formats. They are all included in `textproc/docproj`.

Jade (`textproc/jade`)

A DSSSL implementation. Used for converting marked up documents to other formats, including HTML and TEX.

Links (`www/links`)

A text-mode WWW browser that can also convert XHTML files to plain text.

peps (`graphics/peps`)

Some of the documentation includes images, some of which are stored as EPS files. These must be converted to PNG before most web browsers will display them.

2.1.2 DTDs and Entities

These are the DTDs and entity sets used by the FDP. They need to be installed before you can work with any of the documentation.

XHTML DTD (`textproc/xhtml`)

XHTML is the markup language of choice for the World Wide Web, and is used throughout the FreeBSD web site.

DocBook DTD (`textproc/docbook-xml-450`)

DocBook is designed for marking up technical documentation. All the FreeBSD documentation is written in DocBook.

ISO 8879 entities (`textproc/iso8879`)

19 of the ISO 8879:1986 character entity sets used by many DTDs. Includes named mathematical symbols, additional characters in the Latin character set (accents, diacriticals, and so on), and Greek symbols.

2.2 Optional Tools

You do not need to have any of the following installed. However, you may find it easier to work with the documentation if you do, and they may give you more flexibility in the output formats that can be generated.

2.2.1 Software

JadeTeX, **teTeX** and Modular DocBook Stylesheets (`print/jadetex`, `print/teTeX` and `textproc/dsssl-docbook-modular`)

Jade, **teTeX** and Modular DocBook Stylesheets are used to convert DocBook documents to DVI, Postscript, and PDF formats. The **JadeTeX** macros are needed in order to do this.

If you do not intend to convert your documentation to one of these formats (i.e., HTML and plain text are sufficient) then you do not need to install these.

> **Important:** If you decide to install **JadeTeX** and **teTeX** then you will need to configure **teTeX** after **JadeTeX** has been installed. `print/jadetex/pkg-message` contains detailed instructions explaining what you need to do.

Emacs or **XEmacs** (`editors/emacs` or `editors/xemacs`)

Both these editors include a special mode for editing documents marked up according to an SGML DTD. This mode includes commands to reduce the amount of typing you need, and help reduce the possibility of errors.

You do not need to use them; any text editor can be used to edit marked up documents. You may find they make you more efficient.

If anyone has recommendations for other software that is useful when manipulating XML documents, please let Documentation Engineering Team <`doceng@FreeBSD.org`> know, so they can be added to this list.

Chapter 3 XML Primer

The majority of FDP documentation is written in applications of XML. This chapter explains exactly what that means, how to read and understand the source to the documentation, and the sort of XML tricks you will see used in the documentation.

Portions of this section were inspired by Mark Galassi's Get Going With DocBook (http://www.galassi.org/mark/mydocs/docbook-intro/docbook-intro.html).

3.1 Overview

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. Let's look at this text:

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

It is easy to 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 eXtensible Markup Language (XML) is. Many markup languages have been written in XML, including the two most used by the FDP, XHTML and DocBook.

Each language definition is more properly called a grammar, vocabulary, schema or Document Type Definition (DTD). There are various languages to specify an XML grammar, for example, DTD (yes, it also means the specification language itself), XML Schema (XSD) or RELANG NG. The schema 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 schema 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 XML *parser* which reads in both the schema and a document which claims to conform to the schema. The parser can then confirm whether or not all the elements required by the vocabulary 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”.

> **Note:** This processing simply confirms that the choice of elements, their ordering, and so on, conforms to that listed in the grammar. 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 schema 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 XHTML or DocBook, rather than alterations to the schemas. For this reason this book will not touch on how to write a vocabulary.

3.2 Elements, Tags, and Attributes

All the vocabularies written in XML share certain characteristics. This is hardly surprising, as the philosophy behind XML 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 XML 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 XML (XHTML, 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 grammar 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>`.

Example 3-1. Using an Element (Start and End Tags)

XHTML has an element for indicating that the content enclosed by the element is a paragraph, called `<p>`.

```
<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>
```

Some elements have no content. For example, in XHTML you can indicate that you want a horizontal line to appear in the document.

For such elements, that have no content at all, XML introduced a shorthand form, which is ccompletely equivalent to the above form:

```
<hr/>
```

Example 3-2. Using an Element (Without Content)

XHTML has an element for indicating a horizontal rule, called `<hr>`. This element does not wrap content, so it looks like this.

```
<p>One paragraph.</p>
<hr></hr>

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

For such elements, that have no content at all, XML introduced a shorthand form, which is ccompletely equivalent to the above form:

```
<p>One 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.

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

> **Important:** 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 XML) 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 XHTML, the <p> element has an attribute called `<align>`, which suggests an alignment (justification) for the paragraph to the program displaying the XHTML.

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`.

Example 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.

Example 3-5. Single Quotes Around Attributes

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

XML requires you to quote each attribute value with either single or double quotes. It is more habitual to use double quotes but you may use single quotes, as well. Using single quotes is practical if you want to include double quotes in the attribute value.

The information on attributes, elements, and tags is stored in XML catalogs. The various Documentation Project tools use these catalog files to validate your work. The tools in `textproc/docproj` include a variety of XML 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.)

   Example 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}/docbook/4.1/catalog:$SGML_CATALOG_FILES
   SGML_CATALOG_FILES=${SGML_ROOT}/html/catalog:$SGML_CATALOG_FILES
   SGML_CATALOG_FILES=${SGML_ROOT}/iso8879/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
   ```

   Example 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 ${SGML_ROOT}/docbook/4.1/catalog:$SGML_CATALOG_FILES
   setenv SGML_CATALOG_FILES ${SGML_ROOT}/html/catalog:$SGML_CATALOG_FILES
   setenv SGML_CATALOG_FILES ${SGML_ROOT}/iso8879/catalog:$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 XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DT

   <html xmlns="http://www.w3.org/1999/xhtml">
     <head>
       <title>An Example XHTML 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 XML parser.

   Part of `textproc/docproj` is the `xmllint` validating parser.

   Use `xmllint` in the following way to check that your document is valid:

   ```
   % xmllint --valid --noout example.xml
   ```

   As you will see, `xmllint` 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.

   ```
   % xmllint --valid --noout example.xml
   example.xml:5: element head: validity error : Element head content does not follow the DTD, ex
   ```

   This line tells you that the validation error comes from the *`fifth`* line of the *`example.xml`* file and that the content of the `<head>` is the part, which does not follow the rules described by the XHTML grammar.

   Below this line `xmllint` will show you the line where the error has been found and will also mark the exact character position with a ^ sign.

4. Put the `<title>` element back in.

3.3 The DOCTYPE Declaration

The beginning of each document that you write may specify the name of the DTD that the document conforms to in case you use the DTD specification language. Other specification languages, like XML Schema and RELAX NG are not referred in the source document. This DOCTYPE declaration serves the XML parsers so that they can determine the DTD and ensure that the document does conform to it.

A typical declaration for a document written to conform with version 1.0 of the XHTML DTD looks like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/x
```

That line contains a number of different components.

`<!`

Is the *indicator* that indicates that this is an XML declaration. This line is declaring the document type.

`DOCTYPE`

Shows that this is an XML declaration for the document type.

`html`

Names the first element that will appear in the document.

```
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
```

Lists the Formal Public Identifier (FPI) for the DTD that this document conforms to. Your XML 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 XML processor how to find the DTD referenced in the FPI. Other ways of telling the XML parser how to find the DTD are shown later.

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
```

A local filename or an URL to find the DTD.

```
>
```

Returns to the document.

3.3.1 Formal Public Identifiers (FPIs)

Note: You do not need to know this, but it is useful background, and might help you debug problems when your XML 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, the predecessor (and a superset) of XML.

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 XML 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 XML 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 XML 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 XHTML 1.0 Transitional//EN"            "1.0/transitional.dtd"
```

The XML processor would know to look up the DTD from `transitional.dtd` in the `1.0` subdirectory of whichever directory held the `catalog` file that contained that line.

Look at the contents of `/usr/local/share/xml/dtd/xhtml/catalog.xml`. This is the catalog file for the XHTML 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 XML 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 XML 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

As mentioned earlier, XML is only used when writing a DTD. This is not strictly true. There is certain XML 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 XML syntax. Other uses for XML syntax in your document will be shown later too.

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

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

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

3.5 Comments

Comments are an XML construction, and are normally only valid inside a DTD. However, as Section 3.4 shows, it is possible to use XML syntax within your document.

The delimiter for XML comments is the string "--". The first occurrence of this string opens a comment, and the second closes it.

Example 3-8. XML Generic Comment

```
<!-- test comment -->
<!-- This is inside the comment -->

<!-- This is another comment    -->

<!-- This is one way
     of doing multiline comments -->

<!-- This is another way of   --
  -- doing multiline comments -->
```

If you have used XHTML 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 XHTML parsers, and will accept that as valid. However, the XML parsers used by the Documentation Project are much stricter, and will reject documents that make that error.

Example 3-9. Erroneous XML Comments

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

     THIS IS OUTSIDE THE COMMENT!

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

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

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

This is not valid XML, 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 XML.

3.5.1 For You to Do. . .

1. Add some comments to `example.xml`, and check that the file still validates using `xmllint`.
2. Add some invalid comments to `example.xml`, and see the error messages that `xmllint` gives when it encounters an invalid comment.

3.6 Entities

Entities are a mechanism for assigning names to chunks of content. As an XML 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 XML documents. It is also the only way to include one marked up file inside another using XML.

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 XML 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 XML document. For example, < and & cannot normally appear in an XML document. When the XML 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 XML context. Typically, this is done immediately after the DOCTYPE declaration.

Example 3-10. Defining General Entities

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" [
<!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, whereas general entities can only be used within your document, parameter entities can only be used within an XML 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.

Example 3-11. Defining Parameter Entities

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" [
<!ENTITY % param.some "some">
<!ENTITY % param.text "text">
<!ENTITY % param.new  "%param.some more %param.text">

<!-- %param.new now contains "some more 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 XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" [
<!ENTITY version "1.1">
]>

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>An Example XHTML File</title>
  </head>

  <!-- You might well have some comments in here as well -->

  <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 `xmllint`.

3. Load `example.xml` into your web browser (you may need to copy it to `example.html` before your browser recognizes it as an XHTML 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 XML DTD constructs. Furthermore, the closing `]<` of the XML context are not recognized properly by browser and will probably be rendered.

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

   You can use `xmllint` to do this. It also has an option to drop the initial DTD section so that the closing `]<` does not confuse browsers:

   ```
   % xmllint --noent --dropdtd 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.

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 XML 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 XML parser to use the contents of the named file as the value of the entity.

Example 3-12. Using General Entities to Include Files

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" [
<!ENTITY chapter.1 SYSTEM "chapter1.xml">
<!ENTITY chapter.2 SYSTEM "chapter2.xml">
<!ENTITY chapter.3 SYSTEM "chapter3.xml">
<!-- And so forth -->
]>

<html xmlns="http://www.w3.org/1999/xhtml">
  <!-- Use the entities to load in the chapters -->

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

> **Warning:** 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 because entities are low-level constructs and they are resolved before any parsing happens.

3.7.2 Using Parameter Entities to Include Files

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

Example 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 XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" [
<!-- Define a parameter entity to load in the chapter general entities -->
<!ENTITY % chapters SYSTEM "chapters.ent">

<!-- Now use the parameter entity to load in this file -->
%chapters;
]>

<html xmlns="http://www.w3.org/1999/xhtml">
  &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 XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" [
   <!ENTITY version "1.1">
   <!ENTITY para1 SYSTEM "para1.xml">
   <!ENTITY para2 SYSTEM "para2.xml">
   <!ENTITY para3 SYSTEM "para3.xml">
   ]>

   <html xmlns="http://www.w3.org/1999/xhtml">
     <head>
       <title>An Example XHTML 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`.

```
% xmllint --dropdtd --noent example.xml > example.html
```

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

3.7.3.2 Use Parameter Entities to Include Files

Note: You must have taken the previous steps first.

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

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" [
<!ENTITY % entities SYSTEM "entities.ent"> %entities;
]>

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>An Example XHTML 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.ent`, 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`.

```
% xmllint --dropdtd --noent example.xml > example.html
```

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

3.8 Marked Sections

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

Example 3-14. Structure of A Marked Section

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

As you would expect, being an XML 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 XGML context with `>`.

3.8.1 Marked Section Keywords

3.8.1.1 CDATA

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

When an XML 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 content model you will probably find most useful is `CDATA`.

`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.

> **Note:** When you use `CDATA` in examples of text marked up in XML, keep in mind that the content of `CDATA` is not validated. You have to check the included XML 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.

Example 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 XHTML.  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
```

```
      XHTML.  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.

Example 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, yet this usage is limited to entity files.

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 an entity file that defines general entities to include each chapter and guard these definitions with a parameter entity that can be set to either `INCLUDE` or `IGNORE` to control whether the entity is defined. After these conditional general entity definitions, place one more definition for each general entity to set them to an empty value. This technique makes use of the fact that entity definitions cannot be overridden but always the first definition takes effect. So you can control the inclusion of your chapter with the corrsponding parameter entity; if you set it to `INCLUDE`, the first general entity definition will be read and the second one will be ignored but if you set it to `IGNORE`, the first definition will be ignored and the second one will take effect.

Example 3-17. Using A Parameter Entity to Control a Marked Section

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

<![%electronic.copy;[
<!ENTITY chap.preface  SYSTEM "preface.xml">
]]>

<!ENTITY chap.preface "">
```

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

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

3.8.2 For You to Do. . .

1. Modify the `entities.ent` file to contain the following:

   ```
   <!ENTITY version "1.1">
   <!ENTITY % conditional.text "IGNORE">

   <![%conditional.text;[
   <!ENTITY para1 SYSTEM "para1.xml">
   ]]>

   <!ENTITY para1 "">

   <!ENTITY para2 SYSTEM "para2.xml">
   <!ENTITY para3 SYSTEM "para3.xml">
   ```

2. Normalize the `example.xml` file and notice that the conditional text is not present on the output document. Now if you set the parameter entity guard to `INCLUDE` and regenerate the normalized document, it will appear there again. Of course, this method makes more sense if you have more conditional chunks that depend on the same condition, for example, whether you are generating printed or online text.

3.9 Conclusion

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

Notes

1. *P*arameter entities use the *P*ercent symbol.

Chapter 4 XHTML Markup

4.1 Introduction

This chapter describes usage of the XHTML markup language used for the FreeBSD web site.

XHTML is the XML version of the HyperText Markup Language, the markup language of choice on the World Wide Web. More information can be found at http://www.w3.org/.

XHTML is used to mark up pages on the FreeBSD web site. It is usually not used to mark up other documentation, since DocBook offers a far richer set of elements from which to choose. Consequently, XHTML pages will normally only be encountered when writing for the web site.

HTML has gone through a number of versions. The XML-compliant version described here is called XHTML. The latest widespread version is XHTML 1.0, available in both *strict* and *transitional* variants.

The XHTML DTDs are available from the Ports Collection in `textproc/xhtml`. They are automatically installed as part of the `textproc/docproj` port.

> **Note:** This is *not* an exhaustive list of elements, since that would just repeat the documentation for XHTML. The aim is to list those elements most commonly used. Please post questions about elements or uses not covered here to the FreeBSD documentation project mailing list (http://lists.FreeBSD.org/mailman/listinfo/freebsd-doc).

> **Inline Versus 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.

4.2 Formal Public Identifier (FPI)

There are a number of XHTML FPIs, depending upon the version, or *level* of XHTML to which a document conforms. Most XHTML documents on the FreeBSD web site comply with the transitional version of XHTML 1.0.

```
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

4.3 Sectional Elements

An XHTML 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.

Example 4-1. Normal XHTML Document Structure

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
         <title>The Document's Title</title>
  </head>

  <body>

    ...

  </body>
</html>
```

4.4 Block Elements

4.4.1 Headings

XHTML has tags to denote headings in the 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.

Example 4-2. `<h1>`, `<h2>`, and Other Header Tags

Usage:

```
<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 XHTML 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<i>n</i>>` element should have the same element, but one further up the hierarchy, preceding it. Leaving gaps in the numbering is to be avoided.

Example 4-3. Bad Ordering of <h*n*> Elements

Usage:

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

<!-- Document introduction -->

<h3>Sub-section</h3>

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

4.4.2 Paragraphs

XHTML supports a single paragraph element, <p>.

Example 4-4. <p>

Usage:

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

4.4.3 Block Quotations

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

Example 4-5. <blockquote>

Usage:

```
<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>
```

4.4.4 Lists

XHTML 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.

Example 4-6. `<ul>` and `<ol>`

Usage:

```
<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>
```

Example 4-7. Definition Lists with `<dl>`

Usage:

```
<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>
```

4.4.5 Pre-formatted Text

Pre-formatted text can 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.

Example 4-8. <pre>

For example, the <pre> tags could be used 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, like an email message or program code.

4.4.6 Tables

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 the <p>element is not needed.

Example 4-9. Simple Use of <table>

Usage:

```
<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 or columns that should be spanned.

Example 4-10. Using `rowspan`

Usage:

```
<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>
```

Example 4-11. Using `colspan`

Usage:

```
<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>
```

Example 4-12. Using `rowspan` and `colspan` Together

Usage:

```
<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>
    <!-- Because the large cell on the left merges into
         this row, the first <td> will occur on its
         right -->

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

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

    <td>Bottom middle cell</td>

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

4.5 In-line Elements

4.5.1 Emphasizing Information

Two levels of emphasis are available in XHTML, `<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 should not be relied upon. According to best practices, webpages only hold structural and semantical information and stylesheets are later applied to them. Think of semantics, not formatting, when using these tags.

Example 4-13. `<em>` and `<strong>`

Usage:

```
<p><em>This</em> has been emphasized, while
  <strong>this</strong> has been strongly emphasized.</p>
```

4.5.2 Indicating Fixed-Pitch Text

Content that should be rendered in a fixed pitch (typewriter) typeface is tagged with `<tt>` (for “teletype”).

Example 4-14. `<tt>`

Usage:

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

4.5.3 Links

> **Note:** Links are also inline elements.

4.5.3.1 Linking to Other Documents on the Web

A link points to the URL of another document on the web. 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, typically by a different color or underlining.

Example 4-15. Using `<a href="...">`

Usage:

```
<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.

4.5.3.2 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*. Anchors are indicated with `<a>` and the `id` attribute instead of `href`.

Example 4-16. Using `<a id="...">`

Usage:

```
<p><a id="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 ID of the anchor after a # symbol.

Example 4-17. 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 #).

Example 4-18. 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>
```

Chapter 5 DocBook Markup

5.1 Introduction

This chapter is an introduction to DocBook as it is used for FreeBSD documentation. DocBook is a large and complex markup system, but the subset described here covers the parts that are most widely used for FreeBSD documentation. While a moderate subset is covered, it is impossible to anticipate every situation. Please post questions that this document does not answer to the FreeBSD documentation project mailing list (http://lists.FreeBSD.org/mailman/listinfo/freebsd-doc).

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 (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=docbook). As such, and unlike LinuxDoc and XHTML, DocBook is very heavily oriented towards markup that describes *what* something is, rather than describing *how* it should be presented.

The DocBook DTD is available from the Ports Collection in the `textproc/docbook-xml-450` port. It is automatically installed as part of the `textproc/docproj` port.

> **Formal Versus 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.

> **Inline Versus 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.

5.2 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.

> **Note:** 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 Subversion tree, as head/share/xml/freebsd.dtd (http://svnweb.FreeBSD.org/doc/head/share/xml/freebsd.dtd).

5.3 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 V4.2-Based Extension//EN"
```

5.4 Document Structure

DocBook allows structuring documentation in several ways. The FreeBSD Documentation Project uses two primary types of DocBook document: the book and the article.

Books are 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. For example, 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.

The nature of the document being written should be used to determine 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 (http://www.FreeBSD.org/docs.html) are all marked up as articles, while this document, the FreeBSD FAQ (http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/faq/index.html), and the FreeBSD Handbook (http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/handbook/index.html) are all marked up as books, for example.

5.4.1 Starting a Book

The content of a 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 is contained within <bookinfo>.

Example 5-1. 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>
```

5.4.2 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 is contained within `<articleinfo>`.

Example 5-2. 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>
```

5.4.3 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.

Example 5-3. 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.

Example 5-4. Empty Chapters

```
<chapter>
  <title>This is An Empty Chapter</title>

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

5.4.4 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>.

Example 5-5. 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>
```

Note: 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.

5.4.5 Subdividing Using <part> Elements

<part>s introduce another level 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>
```

5.5 Block Elements

5.5.1 Paragraphs

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

Almost all paragraphs in FreeBSD documentation use <para>. <formalpara> includes a <title> element, and <simpara> disallows some elements from within <para>. Stick with <para>.

Example 5-6. <para>

Usage:

```
<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.

5.5.2 Block Quotations

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

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

Example 5-7. <blockquote>

Usage:

```
<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:

A small excerpt from the US Constitution:

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

5.5.3 Tips, Notes, Warnings, Cautions, Important Information and Sidebars

Extra information may need to be separated from the main body of the text. Typically this is “meta” information of which the user should be aware.

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 loosely defined by the DocBook documentation, which 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.

Example 5-8. `<warning>`

Usage:

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

Appearance:

> **Warning:** Installing FreeBSD may make you want to delete Windows from your hard disk.

5.5.4 Lists and Procedures

Information often needs to be presented as lists, or as a number of steps that must be carried out in order to accomplish a particular goal.

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.

Example 5-9. `<itemizedlist>`, `<orderedlist>`, and `<procedure>`

Usage:

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

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

<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.

5.5.5 Showing File Samples

Fragments of a file (or perhaps a complete file) are shown by wrapping them 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.

Example 5-10. `<programlisting>`

Usage:

```
<para>When finished, the program will 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 finished, the program will look like this:

```
#include <stdio.h>

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

5.5.6 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 the 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.

Example 5-11. `<co>` and `<calloutlist>`

```
<para>When finished, the program will 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 finished, the program will 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.

5.5.7 Tables

Unlike HTML, DocBook does not need tables for layout purposes, as the stylesheet handles those issues. 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 there is 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.

Example 5-12. `<informaltable>`

Usage:

```
<informaltable 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.

Table borders can be suppressed by setting the `frame` attribute to `none` in the `<informaltable>` element. For example, `<informaltable frame="none">`.

Example 5-13. 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

5.5.8 Examples for the User to Follow

Examples for the user to follow are often necessary. Typically, these will consist of dialogs with the computer; the user types in a command, the user gets a response back, the user types 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. 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>`.

> **Note:** `&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 be displayed differently than system output text.

Example 5-14. `<screen>`, `<prompt>`, and `<userinput>`

Usage:

```
<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’
```

> **Note:** 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.

5.6 In-line Elements

5.6.1 Emphasizing Information

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 the document, no equivalent of HTML’s `<b>` and `<i>`. If the information being presented is important, then consider presenting it in `<important>` rather than `<emphasis>`.

Example 5-15. `<emphasis>`

Usage:

```
<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.

5.6.2 Quotations

To quote text from another document or source, or to denote a phrase that is used figuratively, use `<quote>`. Most of the markup tags available for normal text are also available from within a `<quote>`.

Example 5-16. Quotations

Usage:

```
<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.

5.6.3 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>`.

Example 5-17. Keys, Mouse Buttons, and Combinations

Usage:

```
<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 changes, 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 changes, type **Esc : q !**.

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

5.6.4 Applications, Commands, Options, and Cites

Both applications and commands are frequently referred to when writing documentation. The distinction between them is that an application is the name of a program or suite of programs that fulfill a particular task. A command is the filename of a program that the user can type and run at a command line.

It is often necessary to show some of the options that a command might take.

Finally, it is often useful 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>`.

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 FreeBSD documentation will usually include 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 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.

Example 5-18. Applications, Commands, and Options

Usage:

```
<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.1;, and &man.newaliases.1;
  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(1), and newaliases(1) 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`.

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

5.6.5 Files, Directories, Extensions

To refer to the name of a file, a directory, or a file extension, use `<filename>`.

Example 5-19. `<filename>`

Usage:

```
<para>The XML source for the Handbook in English is
  found in <filename class="directory">/usr/doc/en_US.ISO8859-1/books/handbook/</filename>.  The
  file is called <filename>book.xml</filename> in that
  directory.  There is also a <filename>Makefile</filename>
  and a number of files with a <filename>.ent</filename>
  extension.</para>
```

Appearance:

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

5.6.6 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.

To include the name of a program from the FreeBSD Ports Collection in the document, use the `<filename>` tag with the `role` attribute set to `package`. Since ports can be installed in any number of locations, only include the category and the port name; do not include `/usr/ports`.

Example 5-20. `<filename>` Tag with `package` Role

Usage:

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

Appearance:

Install the `net/wireshark` port to view network traffic.

5.6.7 Devices

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

There are two names for devices: the device name as it appears in `/dev`, or the name of the device as it appears in the kernel. For this latter course, use `<devicename>`.

Sometimes there is no choice. Some devices, such as network cards, do not have entries in `/dev`, or the entries are markedly different from their kernel device names.

Example 5-21. `<devicename>`

Usage:

```
<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, network 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, network 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`.

5.6.8 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.

Identification information for networked computers (hosts) can be marked up 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`. The hostname can be explicitly specified 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 (CIDR notation).

`role="mac"`

The text is an Ethernet MAC address, expressed as a series of 2 digit hexadecimal numbers separated by colons.

Example 5-22. `<hostid>` and Roles

Usage:

```
<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 <acronym>IP</acronym> 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 <acronym>MAC</acronym> address uniquely identifies
  every network card in existence.  A typical
  <acronym>MAC</acronym> 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`.

5.6.9 Usernames

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

To refer to a specific username, such as `root` or `bin`, use `<username>`.

Example 5-23. `<username>`

Usage:

```
<para>To carry out most system administration functions
  requires logging in as <username>root</username>.</para>
```

Appearance:

To carry out most system administration functions requires logging in as `root`.

5.6.10 Describing `Makefiles`

> **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 `Makefiles`, `<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.

Example 5-24. `<maketarget>` and `<makevar>`

Usage:

```
<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`.

5.6.11 Literal Text

Literal text, or text which should be entered verbatim, is often needed in documentation. This is text that is excerpted from another file, or which should be copied exactly as shown from the documentation into another file.

Some of the time, `<programlisting>` will be sufficient to denote this text. But `<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>`.

Example 5-25. `<literal>`

Usage:

```
<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.

5.6.12 Showing Items That the User *Must* Fill In

There will often be times when the user is shown what to do, or referred to a file or command line, but cannot simply copy the example provided. Instead, they must supply 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.

Example 5-26. `<replaceable>`

Usage:

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

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.

Usage:

```
<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*.

5.6.13 Quoting System Errors

System errors generated by FreeBSD are marked with `<errorname>`. This indicates the exact error that appears.

Example 5-27. `<errorname>`

Usage:

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

Appearance:

```
Panic: cannot mount root
```

5.7 Images

> **Important:** Image support in the documentation is currently extremely experimental. The mechanisms described here are unlikely to change, but that is not guaranteed.
>
> Installation of the `graphics/ImageMagick` port is required. It is used to convert between the different image formats. This port is *not* in the `textproc/docproj` meta port, it must be installed by hand.
>
> The best example of what follows in practice is the `doc/en_US.ISO8859-1/articles/vm-design/` document. If the description that follows is unclear, 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.

5.7.1 Image Formats

Two image formats are currently supported. Which to choose will depend on the nature of the image.

Images that are primarily vector based, such as network diagrams, time lines, and similar, should be in EPS (Encapsulated Postscript) format. These images have a `.eps` extension.

For bitmaps, such as screen captures, use the PNG (Portable Network Graphic) format. These images have the `.png` extension.

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

Use the appropriate format for each image. It is to be expected that 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*.

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

5.7.2 Image 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>`.

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

There are two circumstances where this can happen.

- When the reader is viewing the documentation in HTML. In this case, each image will need associated alternate text to show the user, typically while 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 make things easier to understand. Suppose there is an image called `fig1.png` that is to be included 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>` contains a `<literallayout>` element, where the `class` attribute is set to `monospaced`. This is an opportunity to demonstrate 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>` contains a single `<phrase>` element. The contents of this phrase will become the `alt` attribute for the image when this document is converted to HTML.

5.7.3 Image `Makefile` Entries

Images must be listed in the `Makefile` in the `IMAGES` variable. This variable must contain the names of all the *source* images. For example, if there are three figures, `fig1.eps`, `fig2.png`, `fig3.png`, then the `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 the source document, you only need to list the image files *you* provided.

5.7.4 Images and Chapters in Subdirectories

Be careful when separating documentation into smaller files in different directories (see Section 3.7.1).

Suppose there is 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, it is suggested to place those images in each chapter's subdirectory (`chapter1/`, `chapter2/`, and `chapter3/`).

However, doing this requires including the directory names in the `IMAGES` variable in the `Makefile`, *and* including the directory name in the `<imagedata>` element in the document document.

For example, if the book has `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 will work.

5.8 Links

Note: Links are also in-line elements.

5.8.1 `id` Attributes

Most DocBook elements accept an `id` attribute to give that part of the document a unique name. The `id` can be used as a target for a crossreference or link.

Any portion of the document that will be a link target must have an `id` attribute. Assigning an `id` to all chapters and sections, even if there are no current plans to link to them, is a good idea. These `id`s can be used as unique anchor reference points by anyone referring to the HTML version of the document.

Example 5-28. `id` on Chapters and Sections

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

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

  <sect1 id="introduction-moredetails">
    <title>More Details</title>

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

Use descriptive values for `id` names. The values must be unique within the entire document, not just in a single file. In the example, the subsection `id` is constructed by appending text to the chapter `id`. This ensures that the `id`s are unique. It also helps both reader and anyone editing the document to see where the link is located within the document, similar to a directory path to a file.

To allow the user to jump into a specific portion of the document, even in the middle of a paragraph or an example, use `<anchor>`. This element has no content, but takes an `id` attribute.

Example 5-29. `<anchor>`

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

5.8.2 Crossreferences with `xref`

`<xref>` provides the reader with a link to jump to another section of the document. The target `id` is specified in the `linkend` attribute, and `<xref>` generates the link text automatically.

Example 5-30. Using <xref>

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

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

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

The link text will be generated automatically, looking like (*emphasized* text indicates the link text):

> More information can be found in *Chapter 1, Introduction.*
>
> More specific information can be found in *Section 1.1, "More Details"*.

The link text is generated automatically from the chapter and section number and title elements.

> **Note:** <xref> cannot link to an id attribute on an <anchor> element. The <anchor> has no content, so the <xref> cannot generate the link text.

5.8.3 Linking to the Same Document or Other Documents on the Web

The link elements described here allow the writer to define the link text. It is very important to use descriptive link text to give the reader an idea of where the link will take them. Remember that DocBook can be rendered to multiple types of media. The reader may be looking at a printed book or other form of media where there are no links. If the link text is not descriptive enough, the reader may not be able to locate the linked section.

5.8.3.1 Links to the Same Document

<link> is used to create a link within the same document. The target id is specified in the linkend attribute. This element wraps content, which is used for the link text.

Example 5-31. Using <link>

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

```
<para>More information can be found in the
  <link linkend="introduction">sample introduction</link>.</para>

<para>More specific information can be found in the
  <link linkend="introduction-moredetails">sample introduction with more details</link> section.</
```

This output will be generated (*emphasized* text is used to show the link text):

> More information can be found in the *sample introduction.*
>
> More specific information can be found in the *sample introduction with more details* section.

> **Note:** <link> can be used to include links to the id of an <anchor> element, since the <link> content defines the link text.

5.8.3.2 Linking to Other Documents on the Web

The `<ulink>` is used to link to external documents on the web. 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.

Example 5-32. `<ulink>` to a FreeBSD Documentation Web Page

Link to the book or article URL entity. To link to a specific chapter in a book, add a slash and the chapter file name, followed by an optional anchor within the chapter. For articles, link to the article URL entity, followed by an optional anchor within the article. URL entities can be found in `doc/share/xml/urls.ent`.

Usage for book links:

```
<para>Read the <ulink
    url="&url.books.handbook;/svn.html#svn-intro">SVN
    introduction</ulink>, then pick the nearest mirror from
  the list of <ulink
    url="&url.books.handbook;/subversion-mirrors.html">Subversion
    mirror sites</ulink>.</para>
```

Appearance:

Read the SVN introduction (http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/handbook/svn.html#svn-intro), then pick the nearest mirror from the list of Subversion mirror sites (http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/handbook/subversion-mirrors.html).

Usage for article links:

```
<para>Read this <ulink url="&url.articles.bsdl-gpl;">article
    about the BSD license</ulink>, or just the <ulink
    url="&url.articles.bsdl-gpl;#intro">introduction</ulink>.</para>
```

Appearance:

Read this article about the BSD license (http://www.FreeBSD.org/doc/en_US.ISO8859-1/articles/bsdl-gpl), or just the introduction (http://www.FreeBSD.org/doc/en_US.ISO8859-1/articles/bsdl-gpl#intro).

Example 5-33. `<ulink>` to a FreeBSD Web Page

Usage:

```
<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 (http://www.FreeBSD.org/index.html) instead.

Example 5-34. `<ulink>` to an External Web Page

Usage:

```
<para>Wikipedia has an excellent reference on
  <ulink
    url="http://en.wikipedia.org/wiki/GUID_Partition_Table">GUID
    Partition Tables</ulink>.</para>
```

Appearance:

Wikipedia has an excellent reference on GUID Partition Tables (http://en.wikipedia.org/wiki/GUID_Partition_Table).

Notes

1. A short history can be found under http://www.oasis-open.org/docbook/intro.shtml#d0e41 (http://www.oasis-open.org/docbook/intro.shtml#d0e41).
2. There are other types of list element in DocBook, but we are not concerned with those at the moment.

Chapter 6 Stylesheets

XML 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 XSLT, XSL FO or CSS.

We use XSLT stylesheets to transform DocBook into XHTML and then we apply CSS formatting to XHTML pages. Currently, the printable output is rendered with legacy DSSSL stylesheets but this may probably change in the future.

6.1 CSS

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

6.1.1 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 XHTML files. The project-wide CSS file is copied from `doc/share/misc/docbook.css` when documents are converted to XHTML, and is installed automatically.

Chapter 7 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.

7.1 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 XML 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.Big5/`. 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.

7.2 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 XHTML 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 XHTML 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 `man`*`n`* 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.

7.3 Document Specific Information

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

7.3.1 The 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.

7.3.1.1 Physical Organization

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

> **Note:** 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 mailing list (http://lists.FreeBSD.org/mailman/listinfo/freebsd-doc).

7.3.1.1.1 Makefile

The Makefile defines some variables that affect how the XML 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.

7.3.1.1.2 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.

7.3.1.1.3 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="kernelconfig">
...
</chapter>
```

Then it will be called `chapter.xml` in the `kernelconfig` directory. In general, the entire contents of the chapter will be held in this file.

When the XHTML version of the Handbook is produced, this will yield `kernelconfig.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. Now, it is possible to include images in each chapter. Images for each Handbook chapter are stored within `share/images/books/handbook`. Note that localized version of these images should be placed in the same directory as the XML sources for each chapter. 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`.

> **Important:** 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 XML 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 XML 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.

Chapter 8 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 XML 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.

8.1 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.

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

8.2 Understanding `Makefile`s 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`.

8.2.1 Subdirectory `Makefile`s

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.

8.2.2 Documentation `Makefiles`

These `Makefiles` 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.

> **Note:** We covered optional variable assignments in the previous section.

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

8.3 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.

8.3.1 `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"
```

8.3.1.1 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.

> **Note:** `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.

8.3.1.2 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.

8.3.2 `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.

8.3.2.1 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.

8.3.2.2 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.

8.3.2.2.1 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.

8.3.2.3 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.

8.3.2.4 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.

Chapter 9 The Website

9.1 Preparation

Use a disk with sufficient free space. A full copy of the documentation and web site files takes over 700 MB. Allowing a full gigabyte provides some breathing room. This space will hold the XML tools, the documentation tree, temporary build space and the installed web pages.

> **Note:** Make sure the documentation ports are updated to the latest version. See the Handbook section on ports (http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/handbook/ports.html#ports-using) for more information.

9.1.1 Using `svn`

`svn` is needed to check out the documentation and web site files from the `doc` Subversion repository. `svn` can be installed with pkg_add(1) or from the FreeBSD Ports Collection by running:

```
# cd /usr/ports/devel/subversion
# make install clean
```

To check out the source files for the FreeBSD web site and the rest of the documentation, run:

```
% svn checkout https://svn0.us-east.FreeBSD.org/doc/head/ ~/doc
```

svn0.us-east.FreeBSD.org (https://svn0.us-east.FreeBSD.org/) is a public `SVN` server. Select the closest mirror and verify the mirror server certificate from the list of Subversion mirror sites (http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/handbook/svn-mirrors.html).

After the checkout completes, the current version of the FreeBSD documentation, including the web site files, will be present in `~/doc`.

9.2 Build the Web Pages

Having obtained the documentation and web site source files, the web site can be built. In this example, the build directory is *`~/doc`* and all the required files are already in place.

The web site is built from the `en_US.ISO8859-1/htdocs` subdirectory of the document tree directory, `~/doc` in this example. Change to the build directory and start the build by executing `make all`.

```
% cd ~/doc/en_US.ISO8859-1/htdocs
% make all
```

> **Tip:** The web site build uses the `INDEX` from the Ports Collection and may fail if that file or `/usr/ports` is not present. The simplest approach is to install the Ports Collection (http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/handbook/ports.html#ports-tree).

9.3 Install the Web Pages

Run `make install`, setting `DESTDIR` to the target directory for the web site files. The files will be installed in `$DESTDIR/data`, which is expected to be the web server's document root.

This installation is run as the `root` user because the permissions on the web server directory will not allow files to be installed by an unprivileged user. In this example, the web site files were built by user `jru` in their home directory, `/usr/home/jru/doc`.

```
# cd /home/jru/doc/en_US.ISO8859-1/htdocs
# env DESTDIR=/usr/local/www make install
```

The install process will not delete any old or outdated files that existed previously in the same directory. If a new copy of the site is built and installed every day, this command will find and delete all files that have not been updated in three days.

```
# find /usr/local/www -ctime 3 -delete
```

9.4 Environment Variables

`ENGLISH_ONLY`

If set and not empty, only the English documents will be built or installed. All translations will be ignored. E.g.:

```
# make ENGLISH_ONLY=YES all install
```

To unset the variable and build all pages, including translations, set `ENGLISH_ONLY` to an empty value:

```
# make ENGLISH_ONLY="" all install clean
```

`WEB_ONLY`

If set and not empty, only the HTML pages from the `en_US.ISO8859-1/htdocs` directory will be built or installed. All other directories within `en_US.ISO8859-1` (Handbook, FAQ, Tutorials) will be ignored. E.g.:

```
# make WEB_ONLY=YES all install
```

`WEB_LANG`

If set, build or install only for the languages specified by this variable inside the *~/doc* directory. All other languages except English will be ignored. E.g.:

```
# make WEB_LANG="el_GR.ISO8859-7 es_ES.ISO8859-1 hu_HU.ISO8859-2 nl_NL.ISO8859-1" all install
```

`WEB_ONLY`, `WEB_LANG`, and `ENGLISH_ONLY` are make(1) variables and can be set in `/etc/make.conf`, `Makefile.inc`, as environment variables on the command line, or in dot files.

Chapter 10 Translations

This is the FAQ for people translating the FreeBSD documentation (FAQ, Handbook, tutorials, manual pages, and others) to different languages.

It is *very* heavily based on the translation FAQ from the FreeBSD German Documentation Project, originally written by Frank Gründer <elwood@mc5sys.in-berlin.de> and translated back to English by Bernd Warken <bwarken@mayn.de>.

The FAQ is maintained by the Documentation Engineering Team <doceng@FreeBSD.org>.

1. Why a FAQ?

More and more people are approaching the freebsd-doc mailing list and volunteering to translate FreeBSD documentation to other languages. This FAQ aims to answer their questions so they can start translating documentation as quickly as possible.

2. What do i18n and l10n mean?

i18n means internationalization and l10n means localization. They are just a convenient shorthand.

i18n can be read as “i” followed by 18 letters, followed by “n”. Similarly, l10n is “l” followed by 10 letters, followed by “n”.

3. Is there a mailing list for translators?

Yes. Different translation groups have their own mailing lists. The list of translation projects (http://www.freebsd.org/docproj/translations.html) has more information about the mailing lists and web sites run by each translation project.

4. Are more translators needed?

Yes. The more people work on translation the faster it gets done, and the faster changes to the English documentation are mirrored in the translated documents.

You do not have to be a professional translator to be able to help.

5. What languages do I need to know?

Ideally, you will have a good knowledge of written English, and obviously you will need to be fluent in the language you are translating to.

English is not strictly necessary. For example, you could do a Hungarian translation of the FAQ from the Spanish translation.

6. What software do I need to know?

It is strongly recommended that you maintain a local copy of the FreeBSD Subversion repository (at least the documentation part). This can be done by running:

```
% svn checkout https://svn0.us-east.FreeBSD.org/doc/head/ head
```

svn0.us-east.FreeBSD.org (https://svn0.us-east.FreeBSD.org/) is a public SVN server. Select the closest mirror and verify the mirror server certificate from the list of Subversion mirror sites (http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/handbook/svn-mirrors.html).

> **Note:** This will require the `devel/subversion` package to be installed.

You should be comfortable using **svn**. This will allow you to see what has changed between different versions of the files that make up the documentation.

For example, to view the differences between revisions `r33733` and `r33734` of `en_US.ISO8859-1/books/fdp-primer/book.xml`, run:

```
% svn diff -r33733:33734 en_US.ISO8859-1/books/fdp-primer/book.xml
```

7. How do I find out who else might be translating to the same language?

The Documentation Project translations page (http://www.FreeBSD.org/docproj/translations.html) lists the translation efforts that are currently known about. If others are already working on translating documentation to your language, please do not duplicate their efforts. Instead, contact them to see how you can help.

If no one is listed on that page as translating for your language, then send a message to the FreeBSD documentation project mailing list (http://lists.FreeBSD.org/mailman/listinfo/freebsd-doc) in case someone else is thinking of doing a translation, but has not announced it yet.

8. No one else is translating to my language. What do I do?

Congratulations, you have just started the “FreeBSD *your-language-here* Documentation Translation Project”. Welcome aboard.

First, decide whether or not you have got the time to spare. Since you are the only person working on your language at the moment it is going to be your responsibility to publicize your work and coordinate any volunteers that might want to help you.

Write an email to the Documentation Project mailing list, announcing that you are going to translate the documentation, so the Documentation Project translations page can be maintained.

If there is already someone in your country providing FreeBSD mirroring services you should contact them and ask if you can have some webspace for your project, and possibly an email address or mailing list services.

Then pick a document and start translating. It is best to start with something fairly small—either the FAQ, or one of the tutorials.

9. I have translated some documentation, where do I send it?

That depends. If you are already working with a translation team (such as the Japanese team, or the German team) then they will have their own procedures for handling submitted documentation, and these will be outlined on their web pages.

If you are the only person working on a particular language (or you are responsible for a translation project and want to submit your changes back to the FreeBSD project) then you should send your translation to the FreeBSD project (see the next question).

10. I am the only person working on translating to this language, how do I submit my translation?

or

We are a translation team, and want to submit documentation that our members have translated for us.

First, make sure your translation is organized properly. This means that it should drop into the existing documentation tree and build straight away.

Currently, the FreeBSD documentation is stored in a top level directory called `head/`. Directories below this are named according to the language code they are written in, as defined in ISO639 (`/usr/share/misc/iso639` on a version of FreeBSD newer than 20th January 1999).

If your language can be encoded in different ways (for example, Chinese) then there should be directories below this, one for each encoding format you have provided.

Finally, you should have directories for each document.

For example, a hypothetical Swedish translation might look like:

```
head/
    sv_SE.ISO8859-1/
                      Makefile
                      htdocs/
                            docproj/
                      books/
                            faq/
                                Makefile
                                book.xml
```

`sv_SE.ISO8859-1` is the name of the translation, in *`lang`*`.`*`encoding`* form. Note the two Makefiles, which will be used to build the documentation.

Use tar(1) and gzip(1) to compress up your documentation, and send it to the project.

```
% cd doc
% tar cf swedish-docs.tar sv_SE.ISO8859-1
% gzip -9 swedish-docs.tar
```

Put `swedish-docs.tar.gz` somewhere. If you do not have access to your own webspace (perhaps your ISP does not let you have any) then you can email Documentation Engineering Team <`doceng@FreeBSD.org`>, and arrange to email the files when it is convenient.

Either way, you should use send-pr(1) to submit a report indicating that you have submitted the documentation. It would be very helpful if you could get other people to look over your translation and double check it first, since it is unlikely that the person committing it will be fluent in the language.

Someone (probably the Documentation Project Manager, currently Documentation Engineering Team <doceng@FreeBSD.org>) will then take your translation and confirm that it builds. In particular, the following things will be looked at:

1. Do all your files use RCS strings (such as "ID")?
2. Does `make all` in the `sv_SE.ISO8859-1` directory work correctly?
3. Does `make install` work correctly?

If there are any problems then whoever is looking at the submission will get back to you to work them out.

If there are no problems your translation will be committed as soon as possible.

11. Can I include language or country specific text in my translation?

We would prefer that you did not.

For example, suppose that you are translating the Handbook to Korean, and want to include a section about retailers in Korea in your Handbook.

There is no real reason why that information should not be in the English (or German, or Spanish, or Japanese, or . . .) versions as well. It is feasible that an English speaker in Korea might try to pick up a copy of FreeBSD whilst over there. It also helps increase FreeBSD's perceived presence around the globe, which is not a bad thing.

If you have country specific information, please submit it as a change to the English Handbook (using send-pr(1)) and then translate the change back to your language in the translated Handbook.

Thanks.

12. How should language specific characters be included?

Non-ASCII characters in the documentation should be included using SGML entities.

Briefly, these look like an ampersand (&), the name of the entity, and a semi-colon (;).

The entity names are defined in ISO8879, which is in the ports tree as `textproc/iso8879`.

A few examples include:

Entity: é
Appearance: é
Description: Small "e" with an acute accent

Entity: É
Appearance: É
Description: Large "E" with an acute accent

Entity: ü
Appearance: ü
Description: Small "u" with an umlaut

After you have installed the iso8879 port, the files in `/usr/local/share/xml/iso8879` contain the complete list.

13. Addressing the reader

In the English documents, the reader is addressed as “you”, there is no formal/informal distinction as there is in some languages.

If you are translating to a language which does distinguish, use whichever form is typically used in other technical documentation in your language. If in doubt, use a mildly polite form.

14. Do I need to include any additional information in my translations?

Yes.

The header of the English version of each document will look something like this:

```
<!--
     The FreeBSD Documentation Project

     $FreeBSD: head/en_US.ISO8859-1/books/faq/book.xml 38674 2012-04-14 13:52:52Z $
-->
```

The exact boilerplate may change, but it will always include a $FreeBSD$ line and the phrase `The FreeBSD Documentation Project`. Note that the $FreeBSD part is expanded automatically by Subversion, so it should be empty (just `$FreeBSD$`) for new files.

Your translated documents should include their own $FreeBSD$ line, and change the `FreeBSD Documentation Project` line to `The FreeBSD` *`language`* `Documentation Project`.

In addition, you should add a third line which indicates which revision of the English text this is based on.

So, the Spanish version of this file might start:

```
<!--
     The FreeBSD Spanish Documentation Project

     $FreeBSD: head/es_ES.ISO8859-1/books/faq/book.xml 38826 2012-05-17 19:12:14Z hrs $
     Original revision: r38674
-->
```

Chapter 11 Writing Style

11.1 Tips

Technical documentation can be improved by consistent use of several principes. Most of these can be classified into three goals: *be clear*, *be complete*, and *be concise*. These goals can conflict with each other. Good writing consists of a balance between them.

11.1.1 Be Clear

Clarity is extremely important. The reader may be a novice, or reading the document in a second language. Strive for simple, uncomplicated text that clearly explains the concepts.

Avoid flowery or embellished speech, jokes, or colloquial expressions. Write as simply and clearly as possible. Simple text is easier to understand and translate.

Keep explanations as short, simple, and clear as possible. Avoid empty phrases like “in order to”, which usually just means “to”. Avoid potentially patronizing words like “basically”. Avoid Latin terms like “i.e.” or “cf.”, which may be unknown outside of academic or scientific groups.

Write in a formal style. Avoid addressing the reader as “you”. For example, say “copy the file to `/tmp`” rather than “you can copy the file to `/tmp`”.

Give clear, correct, *tested* examples. A trivial example is better than no example. A good example is better yet. Do not give bad examples, identifiable by apologies or sentences like “but really it should never be done that way”. Bad examples are worse than no examples. Give good examples, because *even when warned not to use the example as shown*, the reader will usually just use the example as shown.

Avoid *weasel words* like “should”, “might”, “try”, or “could”. These words imply that the speaker is unsure of the facts, and create doubt in the reader.

Similarly, give instructions as imperative commands: not “you should do this”, but merely “do this”.

11.1.2 Be Complete

Do not make assumptions about the reader’s abilities or skill level. Tell them what they need to know. Give links to other documents to provide background information without having to recreate it. Put yourself in the reader’s place, anticipate the questions they will ask, and answer them.

11.1.3 Be Concise

While features should be documented completely, sometimes there is so much information that the reader cannot easily find the specific detail needed. The balance between being complete and being concise is a challenge. One approach is to have an introduction, then a “quick start” section that describes the most common situation, followed by an in-depth reference section.

11.2 Guidelines

To promote consistency between the myriad authors of the FreeBSD documentation, some guidelines have been drawn up for authors to follow.

Use American English Spelling

There are several variants of English, with different spellings for the same word. Where spellings differ, use the American English variant. “color”, not “colour”, “rationalize”, not “rationalise”, and so on.

> **Note:** The use of British English may be accepted in the case of a contributed article, however the spelling must be consistent within the whole document. The other documents such as books, web site, manual pages, etc. will have to use American English.

Do not use contractions

Do not use contractions. Always spell the phrase out in full. “Don’t use contractions” would be wrong.

Avoiding contractions makes for a more formal tone, is more precise, and is slightly easier for translators.

Use the serial comma

In a list of items within a paragraph, separate each item from the others with a comma. Separate the last item from the others with a comma and the word “and”.

For example, look at the following:

> This is a list of one, two and three items.

Is this a list of three items, “one”, “two”, and “three”, or a list of two items, “one” and “two and three”?

It is better to be explicit and include a serial comma:

> This is a list of one, two, and three items.

Avoid redundant phrases

Try not to use redundant phrases. In particular, “the command”, “the file”, and “man command” are probably redundant.

These two examples show this for commands. The second example is preferred.

Use the command `svn` to update your sources.

Use `svn` to update your sources.

These two examples show this for filenames. The second example is preferred.

... in the filename `/etc/rc.local`...

... in `/etc/rc.local`...

These two examples show this for manual references. The second example is preferred (the second example uses `<citerefentry>`).

See `man csh` for more information.

See csh(1).

Two spaces at the end of sentences

Always use two spaces at the end of sentences, as this improves readability, and eases use of tools such as **Emacs**.

While it may be argued that a capital letter following a period denotes a new sentence, this is not the case, especially in name usage. “Jordan K. Hubbard” is a good example; it has a capital `H` following a period and a space, and there certainly is not a new sentence there.

For more information about writing style, see Elements of Style (http://www.bartleby.com/141/), by William Strunk.

11.3 Style Guide

To keep the source for the documentation consistent when many different people are editing it, please follow these style conventions.

11.3.1 Letter Case

Tags are entered in lower case, `<para>`, *not* `<PARA>`.

Text that appears in SGML contexts is generally written in upper case, `<!ENTITY...>`, and `<!DOCTYPE...>`, *not* `<!entity...>` and `<!doctype...>`.

11.3.2 Acronyms

Acronyms should generally be spelled out the first time they appear in a document, as in: “Network Time Protocol (NTP)”. After the acronym has been defined, you should generally use the acronym only (not the whole term, unless it makes more sense contextually to use the whole term). Usually, acronyms are defined only one per document. But if you prefer, you can also define them the first time they appear in each chapter.

All acronyms should be enclosed in `<acronym>` tags, with a `role` attribute with the full term defined. This allows a link to the glossary to be created, and for mouseovers to be rendered with the fully expanded term.

11.3.3 Indentation

Each file starts with indentation set at column 0, *regardless* of the indentation level of the file which might contain this one.

Opening tags increase the indentation level by 2 spaces. Closing tags decrease the indentation level by 2 spaces. Blocks of 8 spaces at the start of a line should be replaced with a tab. Do not use spaces in front of tabs, and do not

add extraneous whitespace at the end of a line. Content within elements should be indented by two spaces if the content runs over more than one line.

For example, the source for this section looks something like:

```
+--- This is column 0
V
<chapter>
  <title>...</title>

  <sect1>
    <title>...</title>

    <sect2>
      <title>Indentation</title>

      <para>Each file starts with indentation set at column 0,
        <emphasis>regardless</emphasis> of the indentation level of the file
        which might contain this one.</para>

      ...
    </sect2>
  </sect1>
</chapter>
```

If you use **Emacs** or **XEmacs** to edit the files then `sgml-mode` should be loaded automatically, and the **Emacs** local variables at the bottom of each file should enforce these styles.

Vim users might want to configure their editor with:

```
augroup sgmledit
  autocmd FileType sgml set formatoptions=cq2l " Special formatting options
  autocmd FileType sgml set textwidth=70       " Wrap lines at 70 columns
  autocmd FileType sgml set shiftwidth=2       " Automatically indent
  autocmd FileType sgml set softtabstop=2      " Tab key indents 2 spaces
  autocmd FileType sgml set tabstop=8          " Replace 8 spaces with a tab
  autocmd FileType sgml set autoindent         " Automatic indentation
augroup END
```

11.3.4 Tag Style

11.3.4.1 Tag Spacing

Tags that start at the same indent as a previous tag should be separated by a blank line, and those that are not at the same indent as a previous tag should not:

```
<article lang='en'>
  <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>
```

11.3.4.2 Separating Tags

Tags like `<itemizedlist>` which will always have further tags inside them, and in fact do not take character data themselves, are always on a line by themselves.

Tags like `<para>` and `<term>` do not need other tags to contain normal character data, and their contents begin immediately after the tag, *on the same line*.

The same applies to when these two types of tags close.

This leads to an obvious problem when mixing these tags.

When a starting tag which cannot contain character data directly follows a tag of the type that requires other tags within it to use character data, they are on separate lines. The second tag should be properly indented.

When a tag which can contain character data closes directly after a tag which cannot contain character data closes, they co-exist on the same line.

11.3.5 White Space Changes

When committing changes, *do not commit changes to the content at the same time as changes to the formatting*.

This is so that the teams that convert the documentation to other languages can quickly see what content has actually changed in your commit, without having to decide whether a line has changed because of the content, or just because it has been refilled.

For example, if you have added two sentences to a paragraph, such that the line lengths on the paragraph now go over 80 columns, first commit your change with the too-long line lengths. Then fix the line wrapping, and commit this second change. In the commit message for the second change, be sure to indicate that this is a whitespace-only change, and that the translation team can ignore it.

11.3.6 Non-Breaking Space

Avoid line breaks in places where they look ugly or make it difficult to follow a sentence. Line breaks depend on the width of the chosen output medium. In particular, viewing the HTML documentation with a text browser can lead to badly formatted paragraphs like the next one:

```
Data capacity ranges from 40 MB to 15
GB.  Hardware compression ...
```

The general entity ` ` prohibits line breaks between parts belonging together. Use non-breaking spaces in the following places:

- between numbers and units:

  ```
  57600 bps
  ```

- between program names and version numbers:

  ```
  FreeBSD 4.7
  ```

- between multiword names (use with caution when applying this to more than 3-4 word names like “The FreeBSD Brazilian Portuguese Documentation Project”):

  ```
  Sun Microsystems
  ```

11.4 Word List

This list of words shows the correct spelling and capitalization when used in FreeBSD Documentation. If a word is not on this list, ask about it on the FreeBSD documentation project mailing list (http://lists.FreeBSD.org/mailman/listinfo/freebsd-doc).

Word	XML Code
CD-ROM	`<acronym>CD-ROM<acronym>`
DoS (Denial of Service)	`<acronym>DoS<acronym>`
email	
file system	
IPsec	
Internet	
manual page	
mail server	
name server	
Ports Collection	Ports Collection
read-only	
Soft Updates	
UNIX®	`&unix;`
web server	

Chapter 12 Using `sgml-mode` with Emacs

Recent versions of **Emacs** or **XEmacs** (available from the Ports Collection) contain a very useful package called PSGML (can be installed from `editors/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)))
```

Chapter 13 See Also

This document is deliberately not an exhaustive discussion of XML, the DTDs listed, and the FreeBSD Documentation Project. For more information about these, you are encouraged to see the following web sites.

13.1 The FreeBSD Documentation Project

- The FreeBSD Documentation Project web pages (http://www.FreeBSD.org/docproj/index.html)
- The FreeBSD Handbook (http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/handbook/index.html)

13.2 XML

- W3C's XML page SGML/XML web page (http://www.w3.org/XML/)

13.3 HTML

- The World Wide Web Consortium (http://www.w3.org/)
- The HTML 4.0 specification (http://www.w3.org/TR/REC-html40/)

13.4 DocBook

- The DocBook Technical Committee (http://www.oasis-open.org/docbook/), maintainers of the DocBook DTD
- DocBook: The Definitive Guide (http://www.docbook.org/), the online documentation for the DocBook DTD
- The DocBook Open Repository (http://docbook.sourceforge.net/) contains DSSSL stylesheets and other resources for people using DocBook

13.5 The Linux Documentation Project

- The Linux Documentation Project web pages (http://www.tldp.org/)

Appendix A. Examples

This appendix contains example XML files and command lines you can use to convert them from one output format to another. If you have successfully installed the Documentation Project tools then you should be able to use these examples directly.

These examples are not exhaustive—they do not contain all the elements you might want to use, particularly in your document's front matter. For more examples of DocBook markup you should examine the XML source for this and other documents, available in the **svn** `doc` repository, or available online starting at http://svnweb.FreeBSD.org/doc/.

To avoid confusion, these examples use the standard DocBook 4.1 DTD rather than the FreeBSD extension. They also use the stock stylesheets distributed by Norm Walsh, rather than any customizations made to those stylesheets by the FreeBSD Documentation Project. This makes them more useful as generic DocBook examples.

A.1 DocBook <book>

Example A-1. DocBook <book>

```
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.2//EN"
  "http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd">

<book lang='en'>
  <bookinfo>
    <title>An Example Book</title>

    <author>
      <firstname>Your first name</firstname>
      <surname>Your surname</surname>
      <affiliation>
        <address><email>foo@example.com</email></address>
      </affiliation>
    </author>

    <copyright>
      <year>2000</year>
      <holder>Copyright string here</holder>
    </copyright>

    <abstract>
      <para>If your book has an abstract then it should go here.</para>
    </abstract>
  </bookinfo>

  <preface>
    <title>Preface</title>

    <para>Your book may have a preface, in which case it should be placed
      here.</para>
  </preface>
```

```
  <chapter>
    <title>My First Chapter</title>

    <para>This is the first chapter in my book.</para>

    <sect1>
      <title>My First Section</title>

      <para>This is the first section in my book.</para>
    </sect1>
  </chapter>
</book>
```

A.2 DocBook `<article>`

Example A-2. DocBook `<article>`

```
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.2//EN"
  "http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd">

<article lang=’en’>
  <articleinfo>
    <title>An Example Article</title>

    <author>
      <firstname>Your first name</firstname>
      <surname>Your surname</surname>
      <affiliation>
        <address><email>foo@example.com</email></address>
      </affiliation>
    </author>

    <copyright>
      <year>2000</year>
      <holder>Copyright string here</holder>
    </copyright>

    <abstract>
      <para>If your article has an abstract then it should go here.</para>
    </abstract>
  </articleinfo>

  <sect1>
    <title>My First Section</title>

    <para>This is the first section in my article.</para>

    <sect2>
      <title>My First Sub-Section</title>
```

```
      <para>This is the first sub-section in my article.</para>
    </sect2>
  </sect1>
</article>
```

A.3 Producing Formatted Output

This section assumes that you have installed the software listed in the `textproc/docproj` port, either by hand, or by using the port. Further, it is assumed that your software is installed in subdirectories under `/usr/local/`, and the directory where binaries have been installed is in your `PATH`. Adjust the paths as necessary for your system.

A.3.1 Using Jade

Example A-3. Converting DocBook to HTML (One Large File)

```
% 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 the standard output (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.

Example A-4. Converting DocBook to HTML (Several Small Files)

```
% 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.

Example A-5. Converting DocBook to Postscript

The source XML file must be converted to a TEX file.

```
% jade -V tex-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
```

Example A-6. Converting DocBook to PDF

The first part of this process is identical to that when converting DocBook to Postscript, using the same `jade` command line (Example A-5).

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.