FreeBSD 文件計畫入門書

FreeBSD 文件計劃

FreeBSD 文件計畫入門書

by FreeBSD 文件計劃

Published $FreeBSD: doc/zh_TW.Big5/books/fdp-primer/book.sgml,v 1.6 2007/12/01 18:47:53 chinsan Exp $

感謝您參與FreeBSD 文件計劃(簡稱：FDP, FreeBSD Documentation Project)，您的點滴貢獻，都相當寶貴。

本入手書內容包括：如何開始著手翻譯的各項細節，以及會用到的一些好用工具(包括：必備工具、輔助工具)，以及文件計畫的宗旨。

本文件還在草稿，尚未完稿。未完成的章節，我們會在章節名稱旁邊加註『⋆』以作識別。

Table of Contents

List of Examples

序言

Shell 提示符號(Prompts)

下表顯示出一般帳號與root 的提示符號，在所有的文件例子中會用提示符號(prompt) ，來提醒您該用哪種帳號才對。

帳號	提示符號(Prompt)
普通帳號	`%`
`root`	#

書中所用的編排風格

下表為本書中所使用編排風格方式：

代表意義	舉例
指令	使用`ls -a` 來列出所有的檔案。
檔名	修改`.login` 檔。
螢幕上會出現的訊息	`You have mail.`
輸入指令後，螢幕上會出現的對應內容。	`% su Password:`
要參考的線上手冊(manual)	以su(1) 來切換帳號。
在講到帳號(user)、群組(group)的名稱的時候...	只有`root` 才可以做這件事。
語氣的强調	你『必須』這麼做才行。
打指令時，可替換的部份	要刪除檔案的話，請打`rm` *要刪除的檔名*
環境變數設定	`$HOME` 是指帳號的家目錄所在處。

『Note、Tip、Important、Warning、Example』的運用

以下文字是『注意』、『技巧』、『重要訊息』、『警告』、『範例』的運用。

Note: 表示需要注意的事項，其中包括您需要注意的事情，因為這些事情可能會影響到操作結果。

Tip: 提供可能對您有用或簡化操作方式的技巧說明。

Important: 表示要特別注意的事情。一般來說，它們會包括操作指令時需要加的額外參數。

Warning: 表示警告事項，比如如果您不注意則可能導致的損失。這些損失可能是對您或硬體造成實際傷害，也可能是無法估計的損害，例如一時疏忽而刪除重要檔案...。

Example 1. 這是舉例說明

這是舉例說明而已，通常包含應遵循的指令範例，或顯示某些特定動作所可能發生的結果。

感謝

在此要感謝Sue Blake, Patrick Durusau, Jon Hamilton, Peter Flynn, Christopher Maden 這些人的協助與閱讀初期草稿，並提供許多寶貴的潤稿意見與評論。

Chapter 1 概論

歡迎參與FreeBSD 文件計劃。維持優秀質量的文件對FreeBSD 的成功來說十分重要，而FreeBSD 文件計劃(以下皆以FDP 來代表FreeBSD Documentation Project 的縮寫) 則與這些文件撰寫、更新息息相關，因此您的點滴貢獻都是十分寶貴的。

本文件最主要的目的，就是清楚告訴您：『FDP 的架構有哪些』、『如何撰寫並提交文件給FDP』、『如何有效運用工具來協助撰稿』。

我們歡迎每個熱心的志士來加入FDP 行列。FDP 並不限定每月必須交出多少稿量，才能加入。您唯一須要作的就是訂閱FreeBSD documentation project 郵遞論壇 (http://lists.FreeBSD.org/mailman/listinfo/freebsd-doc) 。

讀完本份文件，您將會：

- 瞭解有哪些文件是由FDP 所維護的。
- 可以看懂FDP 所維護的SGML 原始文件。
- 知道如何來對文件作修改。
- 知道如何投稿自己的修改部份，並最後正式進入FreeBSD 文件內。

1.1 FreeBSD 文件的組成部分

FDP 總共負責FreeBSD 的4 種類別的文件：

線上手冊(manual)

英文版的系統manual 並不是由FDP 所撰寫的，因為它們是屬於base system 的部份。然而，FDP 可以(也曾這麼做過)修改這些文件，來讓這些文件寫得更清楚，甚至是勘正錯誤的地方。

翻譯團隊負責將系統的線上手冊翻譯為不同的語言。這些譯本都由FDP 維護。

FAQ

FAQ 主要是收集在各論壇或newsgroup 會常問到或有可能會問到的FreeBSD 相關問題與答案。(簡單講，就是『問答集』格式) 通常會擺在這裡面的問答格式，不會放太長的詳細內容。

使用手冊(Handbook)

使用手冊主要是給FreeBSD 使用者提供詳盡的線上參考資料。

Web site

FreeBSD 主要各項介紹方面的WWW 部份，歡迎逛逛http://www.FreeBSD.org/ (http://www.FreeBSD.org/index.html) 以及許多其他mirror 站。這網站是許多人第一次接觸FreeBSD 的地方。

這四個文件組成部分都可透過FreeBSD CVS tree 來取得。也就是說，這些文件的修改記錄對於任何人都是公開的，而且無論是誰都可以用像是**CSup**, **CVSup** 或**CTM** 將文件取出來(checkout)並放在自己機器上做備份或副本參考等用途。

此外，許多人會寫些教學文件或維護有關FreeBSD 內容的網站。(若作者同意的話)其中有些資料會保存在FreeBSD 正式CVS repository 內。而其他的文件，可能作者不希望被放在FreeBSD repository 內而另存他處。總之，FDP 會盡力提供這些文件的連結。

1.2 在開工之前...

本文假設您已經瞭解：

- 如何從FreeBSD CVS repository 更新自己電腦上的FreeBSD 文件部份(以**CVS** 或**CSup** 或**CVSup** 或是**CTM**) 或是用**CVSup** 來下載*checked-out* 的副本
- 如何用FreeBSD Ports 套件管理機制或pkg_add(1) 來下載、安裝軟體。

1.3 快速上手篇

若想先自行試試看，並有信心可以作得到，那麼就照下面步驟吧。

1. 安裝`textproc/docproj` 這個組合型port(meta-port)。

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

2. 下載FreeBSD `doc` tree 到本機上：無論是用CSup 或CVSup 的`checkout` 模式，或是複製完整的CVS repository 到本機上都可以。

   若想在本機上只跑最低限度的CVS repository 就好，那麼必須要checkout 出`doc/share` 以及`doc/en_US.ISO8859-1/share` 這兩個目錄才行。

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

   若硬碟空間還算可以的話，那可以把所有語系的doc 都check out 出來：

   ```
   % cvs checkout doc
   ```

3. 可依需要從repository 中checkout 出來你想修改某份現有的書籍或文章內容。若打算撰寫新書或新文章的話，可以參考現有的部分作為實例來做。

   舉例來說，若想寫篇新文章，內容是有關在FreeBSD 與Windows 2000 之間建立VPN 連線，那麼可以照類似下面這樣的作法：

   1. Check out `articles` 目錄：

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

   2. 複製現有的文章作為範本。在這個例子中，您打算決定把新文章放在`vpn-w2k` 的目錄下。

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

   若是要修改現有文章，像是FAQ(擺在`doc/en_US.ISO8859-1/books/faq`) ，那麼要從repository 中取出來(check out)：

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

4. 以編輯器來編寫`.sgml` 檔。
5. 以`lint` 當輔助參數，來快速檢測文件結構及連結有無錯誤，以下這個指令，實際上不會進行耗時的編書過程，只是先測試文件有無錯誤。

   ```
   % make lint
   ```

當編書的一切都就緒時，這時你可以用FORMATS 變數來指定產生的格式為哪一種。目前支援的格式共有：html, html-split, txt, ps, pdf, rtf 。所支援的格式列表最新版，可參考doc/share/mk/doc.docbook.mk 檔。請記得：在單一指令中，若要同時產生多種格式的話，應使用引號(quotes)來將這些格式括起來。

舉例來說，若只要產生html 格式就好，那麼就打：

```
% make FORMATS=html
```

但若希望有html 及txt 格式的話，你可能要打兩次make(1) 指令才能完成：

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

其實，也可以用單一指令來完成：

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

6. 最後，以send-pr(1) 來提交修改的部份。

Chapter 2 工具

FDP 使用一堆工具來協助管理FreeBSD 文件、轉換文件格式等等。因此，若要進行FDP 工作的話，必須要學會這些工具才行。

這些工具都可以用Ports 或Packages 來安裝，以節省許多安裝的工夫。

您必須安裝這些工具，才能使用接下來各章節會介紹到的例子。這些工具的用法，會在後續相關章節談到。

> **建議安裝textproc/docproj:** 裝了`textproc/docproj` 可以更省時省力，它是個組合型的port(meta-port)，本身並非軟體，只是將一些常用工具組合起來而已。裝了這個port 之後，『應該』就會自動下載、安裝本章所會介紹到的工具了。若要處理中文的話，建議再裝`chinese/docproj` 會比較好。
>
> 在這些packages 當中，你可能會需要使用JadeTeX 這個macro 設定，一旦選擇使用該macro 的話，它會接著去裝TEX。由於TEX 算是個蠻大的套件，除非你需要輸出Postscript 或PDF 格式，否則就不必裝了。
>
> 所以請考慮是否要節省編譯時間、硬碟空間，以判定要不要裝JadeTeX (以及TEX) 了。若要一併裝起來的話：
>
> ```
> # make JADETEX=yes install
> ```
>
> 或是，不裝的話：
>
> ```
> # make JADETEX=no install
> ```
>
> 或者，也可以選擇`textproc/docproj-jadetex` 或是`textproc/docproj-nojadetex` 這兩個之一來裝，它們都是已事先設定`JADETEX` 變數的slave ports，都一樣會裝docproj 差別僅在於有沒有JadeTeX 而已。請注意：若只要輸出HTML 或ASCII 格式文件，那就不用裝**JadeTeX**，而若要輸出PostScript、PDF 格式，就需要裝TEX 才行。

2.1 必備工具

2.1.1 軟體

這些都是在進行FreeeBSD 文件計劃時所會需要用上的工具程式，而且可以用來轉換文件為HTML、plain text以及RTF 格式。這些相關套件在`textproc/docproj` 都已經全部收錄了。

Jade (`textproc/jade`)

DSSSL 規格的實作程式，可用來把標記語言的文件(marked up)轉換為其他格式，像是：HTML 及TEX。

Tidy (`www/tidy`)

HTML “pretty printer”，可用來把自動產生的HTML 內容整理得更易閱讀、以便日後維護。

Links (`www/links`)

文字操作模式的WWW 瀏覽器(browser)可以把HTML 檔轉為plain text 格式。

peps (`graphics/peps`)

文件中有些圖是存成EPS 格式的，這些必須要轉為PNG 格式，才能讓一般瀏覽器可以正常觀看。

2.1.2 DTD 及Entity

由於FDP 有用到許多DTD 跟Entity，因此在開工前，要裝上這些才行。

HTML DTD (`textproc/html`)

HTML 是用於WWW 的標記語言，且也是FreeBSD 網頁所使用的格式。

DocBook DTD (`textproc/docbook`)

DocBook 是專門用來製作技術文件的標示語言版本，FreeBSD 全部文件都是以DocBook 所寫成的。

ISO 8879 entities (`textproc/iso8879`)

在ISO 8879:1986 之中有19 個entity 被許多DTD 所大量使用，包括了數學符號、拉丁字母符號(尖重音等音節符號也是)以及希臘符號。

2.1.3 樣式表(Stylesheets)

這些樣式表都是用來轉換、重排文件的螢幕顯示、列印等效果處理

Modular DocBook 樣式表(`textproc/dsssl-docbook-modular`)

Modular DocBook 樣式表，是用來把DocBook 的標記語言文件轉換為其他格式，像是：HTML 或RTF。

2.2 輔助工具

不一定得裝下列的工具才行，但是，裝了之後會更容易進行各項工作，而且可輸出的格式也更具彈性。

2.2.1 軟體

JadeTeX 及**teTeX** (`print/jadetex` 及`print/teTeX`)

Jade 與**teTeX** 可用來把DocBook 格式文件轉為DVI, Postscript 及PDF 格式。安裝時請記得加上**JadeTeX** 這個macro，這樣才會順便裝上這兩個套件。

若無意把文件轉換更多格式的話(舉例：只要HTML, plain text, RTF 這些格式就夠的話) ，那麼就不用裝**JadeTeX** 與**teTeX**。如此一來可省下一些的編譯時間、安裝空間，因為**teTeX** 大約要至少30MB 空間。

> **Important:** 若決定要裝**JadeTeX** 以及**teTeX** 的話，那麼在裝完**JadeTeX** 之後，要記得設定**teTeX** 才行。`print/jadetex/pkg-message` 內有詳細介紹相關步驟。

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

這兩者編輯器都具有處理SGML DTD 標記文件的特殊模式。該模式提供一些指令，來簡化所需的打字次數，而且可以減少可能發生的錯誤。

不過，這些編輯器並不是必備的；任何文字編輯器都可以用來編輯標記語言文件。不過，你可以透過類似上述這樣的編輯器，來讓這些繁瑣作業更輕鬆有效率些。

若有推薦其他好用的處理SGML 文件程式，請來信讓Documentation Engineering Team <`doceng@FreeBSD.org`> 知道，如此一來，該軟體就會列入這裡介紹了。

Chapter 3 SGML Primer

FDP 文件幾乎都是以SGML 相關程式寫的。本章會介紹SGML 是什麼、如何閱讀、理解這些SGML 原稿，以及本文件中所運用的各項SGML 技巧。

本節部分靈感啓發來自Mark Galassi 的這篇Get Going With DocBook (http://nis-www.lanl.gov/~rosalia/mydocs/docbook-intro/docbook-intro.html)。

3.1 簡介

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

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

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

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

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

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

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

The extra information stored in the markup *adds value* to the document. Adding the markup to the document must typically be done by a person——after all, if computers could recognize the text sufficiently well to add the markup then there would be no need to add it in the first place. This *increases the cost* (i.e., the effort required) to create the document.

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

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

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

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

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

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

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

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

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

> **Note:** This processing simply confirms that the choice of elements, their ordering, and so on, conforms to that listed in the DTD. It does *not* check that you have used *appropriate* markup for the content. If you tried to mark up all the filenames in your document as function names, the parser would not flag this as an error (assuming, of course, that your DTD defines elements for filenames and functions, and that they are allowed to appear in the same place).

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

3.2 Elements, tags, and attributes

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

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

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

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

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

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

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

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

Example 3-1. Using an element (start and end tags)

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

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

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

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

Example 3-2. Using an element (start tag only)

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

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

<hr>

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

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

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 DTD 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 SGML) refers to "the <p> tag" they mean the literal text consisting of the three characters <, p, and >. But the phrase "the <p> element" refers to the whole element.
>
> This distinction *is* very subtle. But keep it in mind.

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

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

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

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

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. If you need to include quotes (") within an attribute then use single quotes around the attribute value.

Example 3-5. Single quotes around attributes

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

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

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

3.2.1 For you to do. . .

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

1. Download and install `textproc/docproj` from the FreeBSD ports system. This is a *meta-port* that should download and install all of the programs and supporting files that are used by the Documentation Project.
2. Add lines to your shell startup files to set `SGML_CATALOG_FILES`. (If you are not working on the English version of the documentation, you will want to substitute the correct directory for your language.)

Example 3-6. `.profile`, for sh(1) and bash(1) users

```
SGML_ROOT=/usr/local/share/sgml
SGML_CATALOG_FILES=${SGML_ROOT}/jade/catalog
SGML_CATALOG_FILES=${SGML_ROOT}/iso8879/catalog:$SGML_CATALOG_FILES
SGML_CATALOG_FILES=${SGML_ROOT}/html/catalog:$SGML_CATALOG_FILES
SGML_CATALOG_FILES=${SGML_ROOT}/docbook/4.1/catalog:$SGML_CATALOG_FILES
SGML_CATALOG_FILES=/usr/doc/share/sgml/catalog:$SGML_CATALOG_FILES
SGML_CATALOG_FILES=/usr/doc/en_US.ISO8859-1/share/sgml/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/sgml
setenv SGML_CATALOG_FILES ${SGML_ROOT}/jade/catalog
setenv SGML_CATALOG_FILES ${SGML_ROOT}/iso8879/catalog:$SGML_CATALOG_FILES
setenv SGML_CATALOG_FILES ${SGML_ROOT}/html/catalog:$SGML_CATALOG_FILES
setenv SGML_CATALOG_FILES ${SGML_ROOT}/docbook/4.1/catalog:$SGML_CATALOG_FILES
setenv SGML_CATALOG_FILES /usr/doc/share/sgml/catalog:$SGML_CATALOG_FILES
setenv SGML_CATALOG_FILES /usr/doc/en_US.ISO8859-1/share/sgml/catalog:$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.sgml`, and enter the following text:

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

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

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

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

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

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

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

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

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

```
% nsgmls -s example.sgml
```

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

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

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

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

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

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

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

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

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

4. Put the `title` element back in.

3.3 The DOCTYPE declaration

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

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

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

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

That line contains a number of different components.

`<!`

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

`DOCTYPE`

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

`html`

Names the first element that will appear in the document.

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

Lists the Formal Public Identifier (FPI) for the DTD that this document conforms to. Your SGML parser will use this to find the correct DTD when processing this document.

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

`>`

Returns to the document.

3.3.1 Formal Public Identifiers (FPIs)

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

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

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

`Owner`

This indicates the owner of the FPI.

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

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

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

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

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

Keyword

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

Description

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

Language

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

3.3.1.1 `catalog` files

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

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

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

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

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

3.3.1.2 `SGML_CATALOG_FILES`

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

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

Typically, you will want to include the following files:

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

You should already have done this.

3.3.2 Alternatives to FPIs

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

The syntax for this is slightly different:

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

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

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

3.4 Escaping back to SGML

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

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

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

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

3.5 註解

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

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

Example 3-8. SGML generic comment

```
<!-- 測試註解 -->
<!-- 這是註解 -->

<!-- 這也是註解     -->

<!-- 要寫多行註解的話，
     這是其中之一的方式 -->

<!-- 要寫多行註解，    --
```

```
-- 也可以這樣子用 -->
```

Use 2 dashes: There is a problem with producing the Postscript and PDF versions of this document. The above example probably shows just one hyphen symbol, - after the <! and before the >.

You *must* use two -, *not* one. The Postscript and PDF versions have translated the two - in the original to a longer, more professional *em-dash*, and broken this example in the process.

The HTML, plain text, and RTF versions of this document are not affected.

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

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

Example 3-9. Erroneous SGML comments

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

     THIS IS OUTSIDE THE COMMENT!

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

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

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

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

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

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

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

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

3.5.1 For you to do. . .

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

3.6 Entities

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

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

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

3.6.1 General Entities

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

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

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

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

You can also use general entities to enter characters that you could not otherwise include in an SGML document. For example, < and & cannot normally appear in an SGML document. When the SGML parser sees the < symbol it assumes that a tag (either a start tag or an end tag) is about to appear, and when it sees the & symbol it assumes the next text will be the name of an entity.

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

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

Example 3-10. Defining general entities

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

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

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

3.6.2 Parameter entities

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

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

Example 3-11. Defining parameter entities

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!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.sgml`.

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

   <html>
     <head>
       <title>An example HTML 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 `nsgmls`.

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

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

4. The solution is to *normalize* your document using an SGML normalizer. The normalizer reads in valid SGML and outputs equally valid SGML which has been transformed in some way. One of the ways in which the

normalizer transforms the SGML is to expand all the entity references in the document, replacing the entities with the text that they represent.

You can use `sgmlnorm` to do this.

```
% sgmlnorm example.sgml > example.html
```

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

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

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

3.7 Using entities to include files

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

3.7.1 Using general entities to include files

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

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

Example 3-12. Using general entities to include files

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

<html>
  <!-- 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.sgml`, `chapter2.sgml`, and so on) *must not* start with a DOCTYPE declaration. This is a syntax error.

3.7.2 Using parameter entities to include files

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

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

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

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

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

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.sgml">
<!ENTITY chapter.2 SYSTEM "chapter2.sgml">
<!ENTITY chapter.3 SYSTEM "chapter3.sgml">
```

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

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!-- 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>
  &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.sgml`, `para2.sgml`, and `para3.sgml`.

   Put content similar to the following in each file:

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

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

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

```
<!ENTITY para2 SYSTEM "para2.sgml">
<!ENTITY para3 SYSTEM "para3.sgml">
]>

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

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

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

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

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

4. Load `example.html` into your web browser, and confirm that the `para`*n*`.sgml` 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.sgml` so that it looks like this:

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

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

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

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

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

   ```
   <!ENTITY version "1.1">
   ```

```
<!ENTITY para1 SYSTEM "para1.sgml">
<!ENTITY para2 SYSTEM "para2.sgml">
<!ENTITY para3 SYSTEM "para3.sgml">
```

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

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

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

3.8 Marked sections

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

Example 3-14. Structure of a marked section

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

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

The first square bracket begins to delimit the marked section.

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

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

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

3.8.1 Marked section keywords

3.8.1.1 CDATA, RCDATA

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

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

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

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

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

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

This is particularly useful if you are including some verbatim text that contains lots of < and & characters. While you could go through the text ensuring that every < is converted to a `<` and every & is converted to a `&`, it can be easier to mark the section as only containing CDATA. When the SGML parser encounters this it will ignore the < and & symbols embedded in the content.

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

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 HTML.  The surrounding text (<para> and
  <programlisting>) are from DocBook.</para>

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

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

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

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

3.8.1.2 `INCLUDE` and `IGNORE`

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

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. Remember that parameter entities can only be used in SGML contexts, and the keyword of a marked section *is* an SGML context.

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

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

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

Example 3-17. Using a parameter entity to control a marked section

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

...

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

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

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

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

3.8.2 For you to do. . .

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

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

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

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

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

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

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

2. Normalize this file using sgmlnorm(1) and examine the output. Notice which paragraphs have appeared, which have disappeared, and what has happened to the content of the CDATA marked section.
3. Change the definition of the `text.output` entity from `INCLUDE` to `IGNORE`. Re-normalize the file, and examine the output to see what has changed.

3.9 Conclusion

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

Notes

1. *P*arameter entities use the *P*ercent symbol.
2. This is a shame. Imagine all the problems and hacks (such as Server Side Includes) that could be avoided if they did.

Chapter 4 SGML Markup

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

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

This is *not* an exhaustive list of elements, since that would just reiterate the documentation for each language. The aim of this section is to list those elements more likely to be useful to you. If you have a question about how best to markup a particular piece of content, please post it to the FreeBSD documentation project 郵遞論壇 (http://lists.FreeBSD.org/mailman/listinfo/freebsd-doc).

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

4.1 HTML

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

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

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

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

4.1.1 Formal Public Identifier (FPI)

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

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

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

4.1.2 Sectional elements

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

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

Example 4-1. Normal HTML document structure

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

  <body>

    ...

  </body>
</html>
```

4.1.3 Block elements

4.1.3.1 Headings

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

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

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

Example 4-2. `<h1>`, `<h2>`, etc.

Use:

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

<!-- Document introduction goes here -->

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

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

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

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

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

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

Generally, an HTML page should have one first level heading (`<h1>`). This can contain many second level headings (`<h2>`), which can in turn contain many third level headings. Each `<h<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 <hn> elements

Use:

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

<!-- Document introduction -->

<h3>Sub-section</h3>

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

4.1.3.2 Paragraphs

HTML supports a single paragraph element, <p>.

Example 4-4. <p>

Use:

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

4.1.3.3 Block quotations

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

Example 4-5. <blockquote>

Use:

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

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

4.1.3.4 Lists

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

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

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

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

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

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

Use:

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

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

  <li>Second item</li>

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

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

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

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

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

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

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

Use:

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

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

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

  <dt>Term 2</dt>

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

  <dt>Term 3</dt>
```

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

4.1.3.5 Pre-formatted text

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

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

Example 4-8. <pre>

You could use <pre> to mark up an email message:

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

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

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

  Comments appreciated.

  N</pre>
```

Keep in mind that < and & still are recognized as special characters in pre-formatted text. This is why the example shown had to use < instead of <. For consistency, > was used in place of >, too. Watch out for the special characters that may appear in text copied from a plain-text source, e.g., an email message or program code.

4.1.3.6 Tables

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

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

Example 4-9. Simple use of <table>

Use:

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

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

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

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

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

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

Example 4-10. Using `rowspan`

Use:

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

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

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

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

Example 4-11. Using `colspan`

Use:

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

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

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

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

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

Use:

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

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

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

  <tr>
    <!-- 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>
```

4.1.4 In-line elements

4.1.4.1 Emphasizing information

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

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

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

Use:

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

4.1.4.2 Bold and italics

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

Example 4-14. `<b>` and `<i>`

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

4.1.4.3 Indicating fixed pitch text

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

Example 4-15. `<tt>`

Use:

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

4.1.4.4 Content size

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

1. Use `<big>` and `<small>` around the content you wish to change size. These tags can be nested, so `<big><big>This is much bigger</big></big>` is possible.
2. Use `<font>` with the `size` attribute set to `+1` or `-1` respectively. This has the same effect as using `<big>` or `<small>`. However, the use of this approach is deprecated.
3. Use `<font>` with the `size` attribute set to a number between 1 and 7. The default font size is `3`. This approach is deprecated.

Example 4-16. `<big>`, `<small>`, and `<font>`

The following fragments all do the same thing.

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

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

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

4.1.5 Links

Note: Links are also in-line elements.

4.1.5.1 Linking to other documents on the WWW

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

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

Example 4-17. Using `<a href="...">`

Use:

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

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

4.1.5.2 Linking to other parts of documents

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

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

Example 4-18. Using `<a name="...">`

Use:

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

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

Example 4-19. Linking to a named part of another document

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

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

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

Example 4-20. Linking to a named part of the same document

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

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

4.2 DocBook

DocBook was originally developed by HaL Computer Systems and O'Reilly & Associates to be a DTD for writing technical documentation [1]. Since 1998 it is maintained by the DocBook Technical Committee (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=docbook). As such, and unlike LinuxDoc and HTML, DocBook is very heavily oriented towards markup that describes *what* something is, rather than describing *how* it should be presented.

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

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

4.2.1 FreeBSD extensions

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

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

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

> **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 CVS tree, as doc/share/sgml/freebsd.dtd (http://www.FreeBSD.org/cgi/cvsweb.cgi/doc/share/sgml/freebsd.dtd).

4.2.2 Formal Public Identifier (FPI)

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

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

4.2.3 Document structure

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

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

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

Chapters and sections contain the remainder of the content.

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

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

The FreeBSD tutorials (http://www.FreeBSD.org/docs.html) are all marked up as articles, while this document, the FreeBSD FAQ (http://www.FreeBSD.org/doc/zh_TW.Big5/books/faq/index.html), and the FreeBSD Handbook (http://www.FreeBSD.org/doc/zh_TW.Big5/books/handbook/index.html) are all marked up as books.

4.2.3.1 Starting a book

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

This additional information should be contained within <bookinfo>.

Example 4-21. Boilerplate <book> with <bookinfo>

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

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

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

```
    <releaseinfo>$FreeBSD$</releaseinfo>

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

  ...

</book>
```

4.2.3.2 Starting an article

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

This additional information should be contained within `<articleinfo>`.

Example 4-22. Boilerplate `<article>` with `<articleinfo>`

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

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

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

    <releaseinfo>$FreeBSD$</releaseinfo>

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

  ...

</article>
```

4.2.3.3 Indicating chapters

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

Example 4-23. A simple chapter

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

  ...
</chapter>
```

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

Example 4-24. Empty chapters

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

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

4.2.3.4 Sections below chapters

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

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

Example 4-25. Sections in chapters

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

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

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

    ...
  </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>

        ...
      </sect3>
    </sect2>

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

      ...
    </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.

4.2.3.5 Subdividing using <part>s

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

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

  <chapter>
    <title>Overview</title>

    ...
  </chapter>

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

    ...
  </chapter>

  <chapter>
    <title>History</title>

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

4.2.4 Block elements

4.2.4.1 Paragraphs

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

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

Example 4-26. <para>

Use:

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

Appearance:

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

4.2.4.2 Block quotations

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

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

Example 4-27. <blockquote>

Use:

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

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

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

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

Appearance:

Preamble to the Constitution of the United States

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

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

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

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

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

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

Example 4-28. `<warning>`

Use:

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

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

4.2.4.4 Lists and procedures

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

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

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

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

Example 4-29. `<itemizedlist>`, `<orderedlist>`, and `<procedure>`

Use:

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

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

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

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

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

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

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

Appearance:

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

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

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

4.2.4.5 Showing file samples

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

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

Example 4-30. `<programlisting>`

Use:

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

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

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

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

Appearance:

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

```
#include <stdio.h>

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

4.2.4.6 Callouts

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

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

Example 4-31. `<co>` and `<calloutlist>`

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

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

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

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

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

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

Appearance:

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

```
#include <stdio.h> ❶

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

❶ Includes the standard IO header file.

❷ Specifies that `main()` returns an int.

❸ The `printf()` call that writes `hello, world` to standard output.

4.2.4.7 Tables

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

In general terms (and see the DocBook documentation for more detail) a table (which can be either formal or informal) consists of a `<table>` element. This contains at least one `<tgroup>` element, which specifies (as an attribute) the number of columns in this table group. Within the tablegroup you can then have one `<thead>` element, which contains elements for the table headings (column headings), and one `<tbody>` which contains the body of the table.

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

Example 4-32. `<informaltable>`

Use:

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

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

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

Appearance:

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

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

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

Example 4-33. Tables where `frame="none"`

Appearance:

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

4.2.4.8 Examples for the user to follow

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

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

`<screen>`

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

Within `<screen>`, white space is significant.

`<prompt>`, `&prompt.root;` and `&prompt.user;`

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

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

> **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 probably be displayed differently to the user.

Example 4-34. `<screen>`, `<prompt>`, and `<userinput>`

Use:

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

Appearance:

```
% ls -1
foo1
```

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

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.

4.2.5 In-line elements

4.2.5.1 Emphasizing information

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

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

Example 4-35. `<emphasis>`

Use:

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

Appearance:

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

4.2.5.2 Quotations

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

Example 4-36. Quotations

Use:

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

Appearance:

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

4.2.5.3 Keys, mouse buttons, and combinations

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

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

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

Example 4-37. Keys, mouse buttons, and combinations

Use:

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

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

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

Appearance:

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

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

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

4.2.5.4 Applications, commands, options, and cites

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

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

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

Mark up application names with `<application>`.

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

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

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

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

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

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

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

...

]>
```

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

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

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

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

Example 4-38. Applications, commands, and options.

Use:

```
<para><application>Sendmail</application> is the most
  widely used Unix mail application.</para>

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

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

Appearance:

Sendmail is the most widely used Unix mail application.

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

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

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

4.2.5.5 Files, directories, extensions

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

Example 4-39. `<filename>`

Use:

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

Appearance:

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

4.2.5.6 The name of ports

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

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

Example 4-40. `<filename>` tag with `package` role

Use:

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

Appearance:

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

4.2.5.7 Devices

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

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

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

Example 4-41. `<devicename>`

Use:

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

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

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

Appearance:

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

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

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

4.2.5.8 Hosts, domains, IP addresses, and so forth

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

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

No role attribute, or `role="hostname"`

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

`role="domainname"`

> The text is a domain name, such as `FreeBSD.org` or `ngo.org.uk`. There is no hostname component.

`role="fqdn"`

> The text is a Fully Qualified Domain Name, with both hostname and domain name parts.

`role="ipaddr"`

> The text is an IP address, probably expressed as a dotted quad.

`role="ip6addr"`

> The text is an IPv6 address.

`role="netmask"`

> The text is a network mask, which might be expressed as a dotted quad, a hexadecimal string, or as a `/` followed by a number.

`role="mac"`

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

Example 4-42. `<hostid>` and roles

Use:

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

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

<para>When adding an IP alias to an interface (using
  <command>ifconfig</command>) <emphasis>always</emphasis> use a
```

```
  netmask of <hostid role="netmask">255.255.255.255</hostid>
  (which can also be expressed as <hostid
    role="netmask">0xffffffff</hostid>.</para>

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

Appearance:

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

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

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

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

4.2.5.9 Usernames

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

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

Example 4-43. `<username>`

Use:

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

Appearance:

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

4.2.5.10 Describing `Makefile`s

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

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

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

Example 4-44. <maketarget> and <makevar>

Use:

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

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

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

Appearance:

Two common targets in a Makefile are all and clean.

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

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

4.2.5.11 Literal text

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

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

On these occasions, use <literal>.

Example 4-45. <literal>

Use:

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

Appearance:

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

4.2.5.12 Showing items that the user *must* fill in

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

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

Example 4-46. `<replaceable>`

Use:

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

Appearance:

```
% man command
```

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

Use:

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

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

Appearance:

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

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

4.2.5.13 Quoting system errors

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

Example 4-47. `<errorname>`

Use:

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

Appearance:

```
Panic: cannot mount root
```

4.2.6 Images

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

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

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

4.2.6.1 Image formats

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

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

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

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

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

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

4.2.6.2 Markup

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

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

There are two circumstances where this can happen.

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

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

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

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

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

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

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

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

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

4.2.6.3 `Makefile` entries

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

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

or

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

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

4.2.6.4 Images and chapters in subdirectories

You must be careful when you separate your documentation into smaller files (see Section 3.7.1) in different directories.

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

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

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

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

  ...

</mediaobject>
```

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

The `Makefile` must contain:

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

Then everything should just work.

4.2.7 Links

Note: Links are also in-line elements.

4.2.7.1 Linking to other parts of the same document

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

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

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

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

Example 4-48. `id on chapters and sections`

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

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

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

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

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

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

Example 4-49. `<anchor>`

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

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

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

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

Example 4-50. Using <xref>

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

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

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

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

More information can be found in *Chapter One*.

More specific information can be found in *the section called Sub-sect 1*.

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

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

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

Example 4-51. Using <link>

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

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

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

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

More information can be found in *the first chapter*.

More specific information can be found in *this* section.

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

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

4.2.7.2 Linking to documents on the WWW

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

Example 4-52. `<ulink>`

Use:

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

Appearance:

Of course, you could stop reading this document and go to the FreeBSD home page (http://www.FreeBSD.org/index.html) instead.

Notes

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

Chapter 5 * Stylesheets

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

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

5.1 * DSSSL

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

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

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

5.2 CSS

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

5.2.1 The Web site (HTML documents)

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

5.2.2 The DocBook documents

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

Chapter 6 Structuring documents under `doc/`

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

1. make it easy to automate converting the document to other formats;
2. promote consistency between the different documentation organizations, to make it easier to switch between working on different documents;
3. make it easy to decide where in the tree new documentation should be placed.

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

6.1 The top level, `doc/`

There are two types of directory under `doc/`, each with very specific directory names and meanings.

Directory: `share/`
Meaning: Contains files that are not specific to the various translations and encodings of the documentation. Contains subdirectories to further categorize the information. For example, the files that comprise the make(1) infrastructure are in `share/mk`, while the additional SGML support files (such as the FreeBSD extended DocBook DTD) are in `share/sgml`.

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.

6.2 The *`lang.encoding/`* directories

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

Directory: `articles`
Contents: Documentation marked up as a DocBook `<article>` (or equivalent). Reasonably short, and broken up into sections. Normally only available as one HTML file.

Directory: `books`
Contents: Documentation marked up as a DocBook `<book>` (or equivalent). Book length, and broken up into chapters. Normally available as both one large HTML file (for people with fast connections, or who want to print it easily from a browser) and as a collection of linked, smaller files.

Directory: `man`
Contents: For translations of the system manual pages. This directory will contain one or more `mann` directories, corresponding to the sections that have been translated.

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

6.3 Document specific information

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

6.3.1 The Handbook

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

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

6.3.1.1 Physical organization

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

> **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 郵遞論壇 (http://lists.FreeBSD.org/mailman/listinfo/freebsd-doc).

6.3.1.1.1 *`Makefile`*

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

6.3.1.1.2 *`book.sgml`*

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

6.3.1.1.3 *`directory/chapter.sgml`*

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

For example, if one of the chapter files contains:

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

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

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

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

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

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

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

Chapter 7 The Documentation Build Process

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

After you have finished reading this chapter you should:

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

7.1 The FreeBSD Documentation Build Toolset

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

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

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

7.2 Understanding Makefiles in the Documentation tree

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

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

7.2.1 Subdirectory Makefiles

These `Makefile`s usually take the form of:

```
SUBDIR =articles
SUBDIR+=books

COMPAT_SYMLINK = en
```

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

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

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

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

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

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

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

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

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

7.2.2 Documentation Makefiles

These 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.sgml

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 .sgml 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.

7.3 FreeBSD Documentation Project make includes

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

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

7.3.1 doc.project.mk

By inspection:

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

PREFIX?=        /usr/local
PRI_LANG?=      en_US.ISO8859-1

.if defined(DOC)
.if ${DOCFORMAT} == "docbook"
.include "doc.docbook.mk"
.endif
.endif

.include "doc.subdir.mk"
.include "doc.install.mk"
```

7.3.1.1 Variables

DOCFORMAT and MAINTAINER are assigned default values, if these are not set by the document make file.

PREFIX is the prefix under which the documentation building tools are installed. For normal package and port installation, this is /usr/local.

PRI_LANG should be set to whatever language and encoding is natural amongst users these documents are being built for. US English is the default.

> **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.

7.3.1.2 Conditionals

The .if defined(DOC) line is an example of a **make** conditional which, like in other programs, defines behavior if some condition is true or if it is false. defined is a function which returns whether the variable given is defined or not.

.if ${DOCFORMAT} == "docbook", next, tests whether the DOCFORMAT variable is "docbook", and in this case, includes doc.docbook.mk.

The two .endifs close the two above conditionals, marking the end of their application.

7.3.2 doc.subdir.mk

This is too long to explain by inspection, you should be able to work it out with the knowledge gained from the previous chapters, and a little help given here.

7.3.2.1 Variables

- SUBDIR is a list of subdirectories that the build process should go further down into.
- ROOT_SYMLINKS is the name of directories that should be linked to the document install root from their actual locations, if the current language is the primary language (specified by PRI_LANG).
- COMPAT_SYMLINK is described in the Subdirectory Makefile section.

7.3.2.2 Targets and macros

Dependencies are described by *target*: *dependency1 dependency2* ... tuples, where to build target, you need to build the given dependencies first.

After that descriptive tuple, instructions on how to build the target may be given, if the conversion process between the target and its dependencies are not previously defined, or if this particular conversion is not the same as the default conversion method.

A special dependency .USE defines the equivalent of a macro.

```
_SUBDIRUSE: .USE
.for entry in ${SUBDIR}
        @${ECHO} "===> ${DIRPRFX}${entry}"
        @(cd ${.CURDIR}/${entry} && \
        ${MAKE} ${.TARGET:S/realpackage/package/:S/realinstall/install/} DIRPRFX=${DIRPRFX}${entry}/
.endfor
```

In the above, _SUBDIRUSE is now a macro which will execute the given commands when it is listed as a dependency.

What sets this macro apart from other targets? Basically, it is executed *after* the instructions given in the build procedure it is listed as a dependency to, and it does not adjust .TARGET, which is the variable which contains the name of the target currently being built.

```
clean: _SUBDIRUSE
        rm -f ${CLEANFILES}
```

In the above, clean will use the _SUBDIRUSE macro after it has executed the instruction rm -f ${CLEANFILES}. In effect, this causes clean to go further and further down the directory tree, deleting built files as it goes *down*, not on the way back up.

7.3.2.2.1 Provided targets

- install and package both go down the directory tree calling the real versions of themselves in the subdirectories (realinstall and realpackage respectively).
- clean removes files created by the build process (and goes down the directory tree too). cleandir does the same, and also removes the object directory, if any.

7.3.2.3 More on conditionals

- exists is another condition function which returns true if the given file exists.
- empty returns true if the given variable is empty.
- target returns true if the given target does not already exist.

7.3.2.4 Looping constructs in make (.for)

.for provides a way to repeat a set of instructions for each space-separated element in a variable. It does this by assigning a variable to contain the current element in the list being examined.

```
_SUBDIRUSE: .USE
.for entry in ${SUBDIR}
        @${ECHO} "===> ${DIRPRFX}${entry}"
        @(cd ${.CURDIR}/${entry} && \
        ${MAKE} ${.TARGET:S/realpackage/package/:S/realinstall/install/} DIRPRFX=${DIRPRFX}${entry}/
.endfor
```

In the above, if SUBDIR is empty, no action is taken; if it has one or more elements, the instructions between .for and .endfor would repeat for every element, with entry being replaced with the value of the current element.

Chapter 8 建構Website

8.1 事前準備

請先準備約200MB 空間，這些是要用來放SGML 工具程式、CVS tree、臨時編譯用的空間，以及編譯好的網頁存放空間。若事先已有裝SGML 工具程式、CVS tree 的話，那麼只需頂多約100MB 空間即可。

> **Note:** 請確認一下你的相關文件製作所會用到的ports 都是最新版！若不清楚所裝的版本為何，那麼就先以pkg_delete(1) 指令來移除舊版，接著才去裝port。舉例來說，若已裝的是jade-1.1，但是我們目前需要的卻是jade-1.2，那麼先用下列方式來移除舊版：
>
> ```
> # pkg_delete jade-1.1
> ```

接著，就是設定CVS repository。需要至少www, doc, ports 這三樣CVS tree(當然還要加上CVSROOT)。請參閱CVSup 簡介 (http://www.FreeBSD.org/doc/zh_TW.Big5/books/handbook/synching.html#CVSUP) 以瞭解如何來mirror a CVS tree 或部分CVS tree。

最低需求的cvsup collections 為：`www`, `doc-all`, `cvs-base` 以及`ports-base`。

剛講的這些需要約105MB 空間。

而完整的CVS tree - 包括`src`, `doc`, `www` 以及`ports` - 目前約為940MB。

8.2 Build the web pages from scratch

1. 先建立要編譯的目錄(至少要有60MB 空間)，並切換到該目錄。

   ```
   # mkdir /var/tmp/webbuild
   # cd /var/tmp/webbuild
   ```

2. 從CVS tree 內checkout 相關的SGML 檔。

   ```
   # cvs -R co www doc
   ```

3. 切到`www/en` 目錄，然後打make(1) `all` 來產生網頁。

   ```
   # cd en
   # make all
   ```

8.3 在你的網頁伺服器上安裝網頁

1. 如果你已經離開`en` 這個目錄，請切換回這個目錄中。

   ```
   # cd path/www/en
   ```

2. 執行make(1) `install` ，並將`DESTDIR` 設定為你想安裝檔案的目錄名稱。

   ```
   # make DESTDIR=/usr/local/www install
   ```

3. 如果你之前已經在相同的目錄中安裝了這些網頁，安裝過程並不會刪除任何既有或過期的網頁。舉例來說，如果你每日建構和安裝新的網頁副本，這個指令將會搜尋並刪除在三天內沒有更新的檔案。

   ```
   # find /usr/local/www -ctime 3 -print0 | xargs -0 rm
   ```

8.4 環境變數

CVSROOT

設定CVS tree 的位置，此為必備條件。

```
# CVSROOT=/home/ncvs; export CVSROOT
```

ENGLISH_ONLY

如果設定這個環境變數，而且值不為空白，makefiles 將只會建構和安裝英文文件。所以將會略過其他的各國翻譯。例如：

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

如果你想要取消變數ENGLISH_ONLY 以及建構所有的頁面並包括翻譯，只要將變數ENGLISH_ONLY 的值設定成空白即可。

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

WEB_ONLY

如果有設定這個變數的話，makefiles 將只會從www 目錄建構及安裝HTML 頁面。所有從doc 目錄下的文件全部都會被忽略(Handbook, FAQ, Tutorials)。例如：

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

NOPORTSCVS

如果設了這個變數，makefiles 就不會從ports cvs repository 取出檔案。取而代之會從/usr/ports (或是PORTSBASE 所設定的值) 內複製檔案。

CVSROOT 是環境變數。你必須直接使用指令或是在dot files (如：~/.profile) 中設定這個環境變數。

WEB_ONLY、ENGLISH_ONLY 及NOPORTSCVS 都是makefile 變數。你可以在/etc/make.conf、Makefile.inc 中設定這些變數，作法就像是用命令列或使用dot files 來設定環境變數一般。

Chapter 9 翻譯時的常見問題

本章是翻譯FreeBSD 文件(包含：FAQ, Handbook, tutorials, manual pages等)的常見問題(FAQ)。

本文件*主要*是以FreeBSD 德文翻譯計劃的翻譯FAQ 為母本而來的，原始撰稿者為Frank Gründer <elwood@mc5sys.in-berlin.de> ，並由Bernd Warken <bwarken@mayn.de> 再翻譯回英文版。

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

1. FAQ 的目的是?

隨著越來越多人參與freebsd-doc 郵遞論壇，而且希望將FreeBSD 文件翻譯為各種語言版本。我們希望這份FAQ 能儘可能為這些參與翻譯者提供快速的解惑。

2. i18n 跟l10n 是什麼呢？

i18n 是internationalization 的簡寫，而l10n 則是localization 的簡寫。這些都是為了書寫方便而用的簡寫。

i18n 就是開頭為“i” 後面有18 個字母，最後接“n”。同理，l10n 則是開頭為“l” 後面有10 個字母，最後接“n”。

3. 有專門給譯者參與討論的mailing list 嗎？

有啊，不同的語系翻譯者都各自有自屬的mailing lists。這份翻譯計劃清單 (http://www.freebsd.org/docproj/translations.html) 有列出各翻譯計劃的詳細mailing lists 及相關網站。

4. 需要更多人一起參與翻譯嗎？

當然囉，越多人參與翻譯，那麼就能夠越快翻完，而且英文版文件若有增減、更新的話，各翻譯版也可以儘快同步囉。

不一定得是專業譯者，才能參與翻譯的。

5. 有要求哪些語言能力呢？

理論上，必須要對英文非常熟稔，而且很明顯地，對想翻譯的語言必須要能運用自如。

英文並非一定要會的。比如說，可以把西班牙文(Spanish)的FAQ 翻譯為匈牙利文(Hungarian)。

6. 該學會哪些程式的使用呢？

強烈建議在自己機器上也建立FreeBSD CVS repository 的備份(至少文件部分)，可以用**CTM** 或**CVSup** 都可以。Handbook 中的"更新、升級FreeBSD" 一章內有提到如何使用這些程式。

此外，需要熟悉**CVS** 用法。如此一來，你可以查閱不同版本之間的差異處。

[XXX To Do(尚未撰稿，仍待補充) -- 寫份上手說明(tutorial)來介紹如何以CVSup 取得文件部分，以及察看不同版本之間的差異。]

7. 要怎麼找出來還有誰要跟我一起翻譯的呢？

文件計劃的翻譯 (http://www.FreeBSD.org/docproj/translations.html) 這列了目前已知的各翻譯者成果，如果已經有其他人也在做跟你一樣的翻譯工作，那麼請不要重複浪費人力，請與他們聯繫看看還有哪些地方可以幫上忙的。

若上面並未列出你母語的翻譯，或是也有人要翻譯但還未公開宣布的話，那麼就寄信到FreeBSD documentation project 郵遞論壇 (http://lists.FreeBSD.org/mailman/listinfo/freebsd-doc) 吧。

8. 都沒人翻譯為我所使用的語言，該怎麼辦？

恭喜啊，你剛好踏上“FreeBSD *你的母語* 文件翻譯計劃” 的啓程之路，歡迎上船。

首先呢，先判斷是否有妥善規劃時間，因為你只有一個人在翻而已，因此，相關翻譯成果的公布、與其他可能會幫忙的志工們聯繫這些工作都是你的職責所在。

寫信到FreeBSD documentation project 郵遞論壇 (http://lists.FreeBSD.org/mailman/listinfo/freebsd-doc) 向大家宣布你正準備要翻譯，然後文件計劃的翻譯部分就會更新相關資料

若你的國家已經有人提供FreeBSD 的mirror(映設) 服務的話，那麼就先跟他們聯繫，並詢問你是否在上面可以有網頁空間來放相關計劃資料，以及是否可以有提供email 帳號或mailing list 服務。

然後，就開始翻文件囉，一開始翻譯的時候，先找些篇幅較短的文件會比較容易些——像是FAQ 啦，或是如何上手之類的說明文章。

9. 已經翻好一些文件了，該寄到哪呢？

這要看情況而定。若你是在翻譯團隊內做的話(像是日本、德國)，他們會有自己內部流程來決定翻譯文件怎麼送，這些大致流程會在他們網頁上面有寫。

若你是某語系的唯一翻譯者(或你是負責某翻譯計劃，並想把成果回饋給FreeBSD 計劃) ，那麼你就應該把自己的翻譯成果寄給FreeBSD 計劃。(細節請看下個問題)

10. 我是該語系的唯一翻譯者，該怎麼把翻譯成果寄出去呢？

或者

我們是翻譯團隊，該怎麼把我們成員翻譯成果寄出去呢？

首先，請先確定你的翻譯成果組織條理分明，並可正確編譯，也就是說：把它擺到現有文件架構內是可以正確編譯成功的。

目前，FreeBSD 文件都是放在最上層的`doc/` 目錄內。而該目錄下的則依其語系來做分類命名的，依照ISO639 定義(`/usr/share/misc/iso639` 的這個FreeBSD 版本比1999/01/20 還新)。

若你這個語系可能會有不同編碼方式(像是：中文) 那麼就應該會像下面這樣，來依你所使用的編碼方式細分。

最後，你應該建立好各文件的目錄了。

舉例來說，假設有瑞典文(Swedish)版的翻譯，那麼應該會長像：

```
doc/
```

```
sv_SE.ISO8859-1/
                Makefile
                books/
                      faq/
                          Makefile
                          book.sgml
```

`sv_SE.ISO8859-1` 是依照*語系(lang).編碼(encoding)* 的規則來建立的譯名。請注意：其中有兩個`Makefiles` 檔，它們是用來編書的。

然後請用tar(1) 與gzip(1) 來把你的翻譯文件壓縮起來，並寄到本計劃來。

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

接著，把`swedish-docs.tar.gz` 放到網頁空間上，若你沒有自己網頁空間的話(ISP不提供) ，那麼可以該檔寄到Documentation Engineering Team <`doceng@FreeBSD.org`> 來。

還有，記得用send-pr(1) 以正式通知大家；你已經寄出翻譯文件了，還有，若有人可以幫忙檢閱、複審文件的話，對翻譯品質較好，因為這也有助於提升翻譯品質的流暢度。

最後，會有人(可能是文件計劃總管，或是Documentation Engineering Team <`doceng@FreeBSD.org`> 成員) 會檢閱你的翻譯文件，並確認是否可正常編譯。此外，他們會特別注意下列幾點：

1. 你的檔案是否都有用RCS tag (像是"ID" 之類的)？
2. `sv_SE.ISO8859-1` 是否可以順利`make all` 編譯呢？
3. `make install` 是否結果有正確？

若有問題的話，那麼檢閱者會叮嚀你，來讓這些翻譯成果可以正確使用。

若沒問題的話，那麼就會很快把你的翻譯成果commit 進去了。

11. 可以加入某語系或某國家才有的東西到翻譯內容內嗎？

我們希望不要這麼做。

舉例來說，假設你正準備把Handbook 翻譯為韓文版，並希望把韓國零售處也加到你翻譯的Handbook 韓文版內。

我們想不出來有啥原因，為什麼不把這些資訊提供給英文版呢？(或是德文、西班牙文、日文等...) 因為，有可能英語讀者跑去韓國時，會想買FreeBSD 相關產品。此外，這也可以提升FreeBSD 的可見度，很顯然的，這並不是件壞事啊。

若你有某國才有的資料，請(用send-pr(1))提供給英文版Handbook 以作為修訂，然後再把英文版的修訂部分，翻為你要翻譯的Handbook 吧。

感恩，謝謝。

12. 要怎麼把該語系特有的字元寫進去翻譯內容呢？

文件內所有的非ASCII(Non-ASCII) 字元，都要使用SGML entities 才能寫進去。

簡單來說，長相一開頭會是& 符號(&)，然後是該entity 名稱，最後接上分號(;)。

這些entity 名稱都是ISO8879 所制訂的，而port tree 內則在`textproc/iso8879`。

以下舉一些例子：

Entity名稱: é
實際樣子: é
介紹: 小“e”，並帶尖、重音(acute accent)

Entity名稱: É
實際樣子: É
介紹: 大“E”，並帶尖、重音(acute accent)

Entity名稱: ü
實際樣子: ü
介紹: 小“u”，並帶日耳曼語系中的母音變化(umlaut)

在裝了iso8879 這個port 之後，就可以在`/usr/local/share/sgml/iso8879` 找到這些的詳細列表。

13. 如何稱呼讀者呢？

在英文文件內，讀者都是以“you” 來稱呼，而有些語言並沒有正式/非正式的區隔。

若你所要翻的語言可以區別這些差異，那麼請用該語系在一般技術文件上所使用的稱呼吧。如果容易造成困惑的話，那麼請改用較中性的稱呼來取代。

14. 翻譯成果內要不要附上一些其他訊息呢？

當然要。

每份英文版原稿的開頭，通常會有像下面的內容：

```
<!--
     The FreeBSD Documentation Project

     $FreeBSD: doc/en_US.ISO8859-1/books/fdp-primer/translations/chapter.sgml,v 1.5 2000/07/07 18:38
-->
```

實際上的內容可能稍有不同，但每份原稿都會附上$FreeBSD$ 這一行以及`The FreeBSD Documentation Project` 宣告。請注意：$FreeBSD 開頭的這行是會由CVS 隨著每次異動而自動更改的，所以，新檔案的話請保持原狀(也就是只要寫`$FreeBSD$` 就好了)。

翻譯文件中，必須都要有$FreeBSD$ 這行，並且把`FreeBSD Documentation Project` 這行改為`The FreeBSD` *`你的語系`* `Documentation Project`。

此外，還必須加上第三行來指出你所翻譯的，到底是以英文版原稿的哪一版本為母本所做的翻譯。

因此呢，西班牙文版(Spanish)的檔案開頭應該是長像這樣：

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

```
     $FreeBSD: doc/es_ES.ISO8859-1/books/fdp-primer/translations/chapter.sgml,v 1.3 1999/06/24 19:12
     Original revision: 1.11
-->
```

Chapter 10 文件的撰寫風格

由於FreeBSD 文件是由眾多作者所維護的，為了保持寫作風格的一貫性，於是就產生較有共識的寫作規則，請各位記得要遵守。

使用美式英語

同一個字在不同種類的英語會有著不同的拼法。遇到拼字不同的情況，請採用美式英語拼法。像是：請改用“color”，而非“colour”。請改用“rationalize”，而非“rationalise” 等等類似字彙。

> **Note:** 若文章採用英式英語也可以接受，但必須全篇文章都採用同一拼法才行。而文件的其他部份，像是書、網頁、manual 說明等則必須採用美式英語。

不要用簡寫

請不要簡寫(contraction)。請務必將完整的字寫出來。比如：“Don’t use contractions” 這句有用到簡寫，就要避免。

正式書面寫法避免簡寫的原因，乃是因為如此一來字句意思較精準，且對譯者會比較輕鬆些。

正確使用serial comma 以及頓號

英文段落通常會逗號(,)作為該句所提到的各項目的語氣區隔。並且會在最後一個提到的項目時，先加上逗號再接上“and”，最後才是最後的項目。

舉個例子，看看下面這句：

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

那麼這一句到底是有三個項目(“one”、“two” 、“three”)呢？或者是只有兩個項目(“one”、“two and three”)呢？

因此較妥的方式是以serial comma 的方式，才能正確表達語意：

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

然而，在翻譯過程中，建議把逗號(,)部份改為頓號(、)，並且“and” 的部份可略而不翻，以免語意頓塞。

避免使用贅詞

請試著避免使用贅詞(redundant phrase)。尤其是“這個指令”、“這個檔案”、“man 指令” 這幾個通常都是不必要的贅詞。

以指令(command)方面舉例，比較妥當的用法是第二句的例子：

使用`cvsup` 指令來更新原始碼。

使用`cvsup` 來更新原始碼。

以檔案(filename)方面舉例，比較妥當的用法是第二句的例子：

... 在這個/etc/rc.local 檔案...

... 在/etc/rc.local 檔...

以man(manual)方面舉例，比較妥當的用法是第二句(有用到SGML <citerefentry> 標籤)：

請打man csh 指令以參閱詳情說明。

詳情請參閱csh(1)。

每句後面加上兩個空白

為了使文章更易閱讀，以及讓**Emacs** 之類的工具容易運用，請在每一完整句子後面加上兩個空白。

不過，句號(.)後面有接大寫字母，並不一定表示前一個句點所在處就是完整句子，尤其是名字部份常常會有這現象。像是“Jordan K. Hubbard” 這人名就是很好的例證：句號後面接空白，然後是大寫的H，然而這肯定並不是兩段句子。

撰寫風格的相關細節，可參閱William Strunk 所寫的Elements of Style (http://www.bartleby.com/141/)。

10.1 Style guide

由於Handbook 是由眾多作者所維護，為了保持寫作風格的一貫性，請遵守下列撰寫風格。

10.1.1 大小寫

Tag 的部份都是用小寫字母，譬如是用<para> ，*而非* <PARA>。

而SGML 內文則是用大寫字母表示，像是：<!ENTITY...> 及<!DOCTYPE...>，*而不是* <!entity...> 及<!doctype...>。

10.1.2 縮寫字

縮寫字(acronym)通常在書中第一次提到時，必須同時列出完整拼法，比如："Network Time Protocol (NTP)"。定義縮寫字之後，應該儘量只使用該縮寫字(而非完整詞彙，除非使用完整詞彙可以更能表達語意)來表達即可。通常每本書只會第一次提到時，才會列出完整詞彙，但若您高興也可以在每章第一次提到時又列出完整詞彙。

此外，同一縮寫字在前三次使用時，須使用<acronym> 標籤，並把完整詞彙附在role 屬性內做說明。如此一來就會建立詞彙表，並且當滑鼠移至該縮寫字上方時，就會顯示完整詞彙。

10.1.3 縮排

無論 檔案縮排設定為何，每個檔案一開始的縮排(indentation)都是從0 縱列開始

未完的標籤會以多兩個空白來增加縮排，結尾的標籤則少兩個空白來縮減縮排。若已達8 個空白，則以tab 取代之。此外，在tab 前面不要再用空白，也不要在每行後面加上空白。每個tag 的內文若超過一行的話，則接下來的就多兩個空白以做縮排。

舉個例子，這節所用的寫法大致是下面這樣：

```
+--- 這是 0 縱列
V
<chapter>
  <title>...</title>

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

    <sect2>
      <title>縮排</title>

      <para><emphasis>無論</emphasis> 檔案縮排設定為何，
        每個檔案一開始的縮排(indentation)都是從 0 縱列開始。</para>

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

若用**Emacs** 或**XEmacs** 來編輯這檔，那麼會自動進入`sgml-mode` 模式，然後就會強制使用每個檔案最下方的環境設定。

Vim 愛用者也可以用下列設定來調整：

```
augroup sgmledit
  autocmd FileType sgml set formatoptions=cq2l " 特殊格式選項
  autocmd FileType sgml set textwidth=70       " 在 70 縱列處即自動換行
  autocmd FileType sgml set shiftwidth=2       " 自動縮排 2 個空白
  autocmd FileType sgml set softtabstop=2      " 按 Tab 會自動轉為兩個空白縮排
  autocmd FileType sgml set tabstop=8          " 把 8 個空白轉為 tab
  autocmd FileType sgml set autoindent         " 自動縮排
augroup END
```

10.1.4 Tag 風格

10.1.4.1 Tag 空行

同一縮排等級的標籤要以空一行來做區隔，而不同縮排等級的則不必。比如：

```
<article>
  <articleinfo>
    <title>NIS</title>

    <pubdate>October 1999</pubdate>

    <abstract>
```

```
      <para>...
        ...
        ...</para>
    </abstract>
  </articleinfo>

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

    <para>...</para>
  </sect1>

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

    <para>...</para>
  </sect1>
</article>
```

10.1.4.2 標籤的分行

像是<itemizedlist> 這類的標籤事實上本身不含任何文字資料，必須得由其他標籤來補充內文。這類的標籤會獨用一整行。

另外，像是<para> 及<term> 這類的標籤並不需搭配其他標籤，就可附上文字資料，並且在標籤後面的*同一行* 內即可立即寫上這些內文。

當然，這兩類的標籤結尾時也是跟上面道理相同。

不過，當上述這兩種標籤混用時，會有很明顯的困擾。

當第一類標籤的後面接上第二類標籤的話，那麼要把這兩類標籤各自分行來寫。後者標籤的段落，也是需要做適當縮排調整。

而第二類標籤結尾時，可以與第一類標籤的結尾放在同一行。

10.1.5 空白的更改

在commit 修改時，*請別在修改內容的同時，也一起更改編排格式*。

如此一來，像是Handbook 翻譯團隊才能迅速找出你改了哪些內容，而不用費心思去判斷該行的改變，是由於格式重排或者內容異動。

舉例說明，若要在某段加上兩個句子，如此一來該段落的某行勢必會超出80 縱列，這時請先commmit 修改。接著，再修飾過長行落的換行，然後再次commit 之。而第二次的commit 紀錄，請明確說明這只是whitespace-only (修改空白而已) 的更改，如此一來，翻譯團隊就可以忽略第二次commit 了。

10.1.6 Nonbreaking space

請避免一些情況下的斷行：造成版面醜醜的、或是須連貫表達的同一句子。斷行的情況會隨所閱讀的工具不同而有所不同。尤其是透過純文字瀏覽器來看HTML 時會更明顯看到類似下面這樣不好的編排段落：

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

請使用` `以避免同句子之間的斷行，以下示範如何使用nonbreaking spaces：

- 在數字與單位之間：

  ```
  57600 bps
  ```

- 在程式名稱與版號之間：

  ```
  FreeBSD 4.7
  ```

- multiword 之間(使用時請小心，像是“The FreeBSD Brazilian Portuguese Documentation Project” 這類由三到四個字所組成的，則不用加。)：

  ```
  Sun Microsystems
  ```

10.2 詞彙表

以下為FreeBSD 文件計劃編排時所採用的小型詞彙表。若找不到要找的詞彙，請參閱O’Reilly word list (http://www.oreilly.com/oreilly/author/stylesheet.html)。

- 2.2.X
- 4.X-STABLE
- CD-ROM
- DoS *(Denial of Service)*
- Ports Collection
- IPsec
- Internet
- MHz
- Soft Updates
- Unix
- disk label
- email
- file system
- manual page
- mail server
- name server
- null-modem
- web server

Chapter 11 Using `sgml-mode` with Emacs

Recent versions of **Emacs** or **XEmacs** (available from the ports collection) contain a very useful package called PSGML. Automatically invoked when a file with the `.sgml` 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/sgml/catalog")))
(add-hook 'psgml-mode-hook
  '(lambda () (local-psgml-mode-hook)))
```

Chapter 12 他山之石

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

12.1 The FreeBSD Documentation Project

- The FreeBSD Documentation Project web pages (http://www.FreeBSD.org/docproj/index.html)
- The FreeBSD Handbook (http://www.FreeBSD.org/doc/zh_TW.Big5/books/handbook/index.html)

12.2 SGML

- The SGML/XML web page (http://www.oasis-open.org/cover/), a comprehensive SGML resource
- Gentle introduction to SGML (http://etext.virginia.edu/bin/tei-tocs?div=DIV1&id=SG)

12.3 HTML

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

12.4 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.

12.5 The Linux Documentation Project

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

Appendix A. 範例

本附錄收錄一些SGML 檔範例，以及用來轉換格式的相關指令。若已成功安裝文件計畫工具包的話，那麼就可以直接照下面範例來使用。

這些例子並不是很詳細——並未包括你可能想用的元件，尤其像是你文件的前頁(正文前的書頁，包括扉頁、序言、目錄等) 若需參考更多DocBook 標記語言文件的話，那麼可以透過**CSup**、**CVSup** 程式來抓`doc` tree 部分，以察看本文件或其他文件的SGML 原稿。或者，也可以線上瀏覽http://www.FreeBSD.org/cgi/cvsweb.cgi/doc/。

為了避免不必要的困擾，這些例子採用標準的DocBook 4.1 DTD 而非FreeBSD 額外的DTD。同時並採用Norm Walsh 氏的樣式表(stylesheets)，而非FreeBSD 文件計劃有自行改過的樣式表。在一般使用DocBook 的例子，這樣子會比較簡化環境，而不會造成額外困擾。

A.1 DocBook `<book>`

Example A-1. DocBook `<book>`

```
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook V4.1//EN">

<book>
  <bookinfo>
    <title>樣本書的例子</title>

    <author>
      <firstname>名(first name)</firstname>
      <surname>姓(surname)</surname>
      <affiliation>
        <address><email>foo@example.com</email></address>
      </affiliation>
    </author>

    <copyright>
      <year>2000</year>
      <holder>相關版權字樣</holder>
    </copyright>

    <abstract>
      <para>該書若有摘要，就寫在這邊。</para>
    </abstract>
  </bookinfo>

  <preface>
    <title>序言</title>

    <para>該書若有序言，就放在這邊。</para>
  </preface>

  <chapter>
    <title>第一章</title>
```

```
    <para>這是這本書的第一章。</para>

    <sect1>
      <title>第一節</title>

      <para>這本書的第一節。</para>
    </sect1>
  </chapter>
</book>
```

A.2 DocBook <article>

Example A-2. DocBook <article>

```
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook V4.1//EN">

<article>
  <articleinfo>
    <title>文章樣本</title>

    <author>
      <firstname>名(first name)</firstname>
      <surname>姓(surname)</surname>
      <affiliation>
        <address><email>foo@example.com</email></address>
      </affiliation>
    </author>

    <copyright>
      <year>2000</year>
      <holder>相關版權字樣</holder>
    </copyright>

    <abstract>
      <para>該文章若有摘要，就寫在這邊。</para>
    </abstract>
  </articleinfo>

  <sect1>
    <title>第一節</title>

    <para>該文章的第一節。</para>

    <sect2>
      <title>第一小節(sub-section)</title>

      <para>該文章的第一小節(sub-section)</para>
    </sect2>
  </sect1>
```

```
</article>
```

A.3 Producing formatted output

本節有些前提，假設：已經有裝`textproc/docproj`上面所安裝各軟體，無論它們是用port 方式安裝或是手動安裝。此外，假設所裝的軟體都放在`/usr/local/` 下的子目錄，並且所安裝的相關執行檔，都有裝在你的`PATH` 環境變數內的目錄。如有必要的話，請依你的系統環境而調整相關路徑。

A.3.1 使用Jade

Example A-3. 轉換DocBook 為HTML (完整模式)

```
% jade -V nochunks \  ❶
    -c /usr/local/share/sgml/docbook/dsssl/modular/catalog \ ❷
    -c /usr/local/share/sgml/docbook/catalog \
    -c /usr/local/share/sgml/jade/catalog \
    -d /usr/local/share/sgml/docbook/dsssl/modular/html/docbook.dsl \❸
    -t sgml ❹ file.sgml > file.html ❺
```

❶ Specifies the `nochunks` parameter to the stylesheets, forcing all output to be written to STDOUT (using Norm Walsh's stylesheets).

❷ Specifies the catalogs that Jade will need to process. Three catalogs are required. The first is a catalog that contains information about the DSSSL stylesheets. The second contains information about the DocBook DTD. The third contains information specific to Jade.

❸ Specifies the full path to the DSSSL stylesheet that Jade will use when processing the document.

❹ Instructs Jade to perform a *transformation* from one DTD to another. In this case, the input is being transformed from the DocBook DTD to the HTML DTD.

❺ Specifies the file that Jade should process, and redirects output to the specified `.html` file.

Example A-4. 轉換DocBook 為HTML (章節模式)

```
% jade \
    -c /usr/local/share/sgml/docbook/dsssl/modular/catalog \ ❶
    -c /usr/local/share/sgml/docbook/catalog \
    -c /usr/local/share/sgml/jade/catalog \
    -d /usr/local/share/sgml/docbook/dsssl/modular/html/docbook.dsl \❷
    -t sgml ❸ file.sgml 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. 轉換DocBook 為Postscript(PS) 格式

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

```
% jade -Vtex-backend \ ❶
    -c /usr/local/share/sgml/docbook/dsssl/modular/catalog \ ❷
    -c /usr/local/share/sgml/docbook/catalog \
    -c /usr/local/share/sgml/jade/catalog \
    -d /usr/local/share/sgml/docbook/dsssl/modular/print/docbook.dsl \❸
    -t tex ❹ file.sgml
```

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. 轉換DocBook 為PDF 格式

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.