Writing Documentation Using DocBook

A Crash Course

David Rugge

davidrugge@mindspring.com

Mark Galassi

rosalia@galassi.org

Éric Bischoff

ebischoff@nerim.net

Writing Documentation Using DocBook: A Crash Course

by David Rugge, Mark Galassi, and Éric Bischoff

This document is a first tutorial on how to write documentation in DocBook.

Table of Contents

1. Introduction 1

1.1. About this Booklet 1
1.2. Why DocBook? 2
1.3. Your World View 3
1.4. Markup based on content 4

2. Getting started 5

2.1. Presentation of the Tools 5
2.2. The Old Way: the DocBook-tools 5
2.3. The New Way: XSLTProc and FOP 7
2.4. My First DocBook File 8
2.5. Introducing the Style Sheets 9

3. Basic notions 11

3.1. Anatomy of a DocBook Tag 11
3.2. The Structure of a DocBook File 11

4. The Document Type Declaration 14

4.1. Using Entities for Shared Text 14
4.2. Using Entities to Include Other files 15
4.3. Identifying files with formal public IDs 15
4.4. Using Marked Sections to Handle Conditional Content 17

5. Meta Information 19

6. Lists 21

6.1. The simplelist 21
6.2. The itemizedlist 22
6.3. The orderedlist 22
6.4. The variablelist 23
6.5. The segmentedlist 24
6.6. qandaset 25
6.7. Procedures 25

7. Tables 27

8. Graphics 29

9. Links 32

10. Describing the Application’s Interface 33

10.1. Examples 33
10.2. GUI Interface Elements 34
10.3. Command Line Elements 36
10.4. Describing an API 38

11. Miscellaneous Useful Tags 40

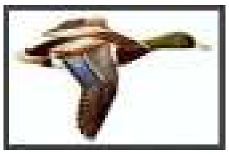
11.1. Labelling Tags 40
11.2. Formatting Tags 40
11.3. Warnings, Tips, and Notes 40

12. Where to Go Next 41

12.1. DocBook Resources 41

A. Licence 42

A.1. Free Documentation Licence 42



B. Emacs PSGML mode tips..50
Glossary..52

List of Tables

7-1. Mouse Mileage 28

Chapter 1. Introduction

1.1. About this Booklet

This booklet was meant as a tutorial that can give you an introduction on how to use all the power and effectiveness of DocBook. It aims at getting you introduced to this matter in the shortest delay, hence then name of "Crash-course to DocBook".

Even if you have never used either DocBook or other markup languages (like LinuxDoc) before, you should be able to become proficient in it just by reading through this guide and using the online (http://www.oasis-open.org/docbook/documentation/reference/html/docbook.html) or the paper (http://www.oreilly.com/catalog/docbook/index.html) version of [DocBook - The Definitive Guide] published by O’Reilly & Associates.

> **Note:** Please note that this crash course is designed to be used along with, not instead of, the DocBook Reference. There are a number of cases where it is much easier to refer to the reference rather than trying to rehash what it already covers. Use this guide to understand what DocBook is about and to have a general overview on how to use those tags.

This tutorial will teach you enough DocBook to write basic documentation. You will learn:

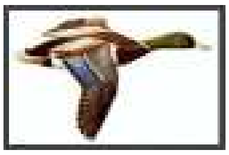
- What DocBook is about
- How to get the DocBook-Tools up and running
- The format of a DocBook tag
- How to structure your documents properly
- How to use lists and tables to organize data
- How to describe GUI elements and Unix Commands
- How to include graphics in your documentation
- How to link to URL’s and create cross-references

In appendix, one can find a short description of the Emacs psgml mode.

This material comes from the fusion of three documents:

- the "Introduction to DocBook" by Mark Galassi
- the "KDE crash-course to DocBook" by David Rugge
- parts of Eric Bischoff’s tutorial about DocBook

Many parts of this document were borrowed from the "DocBook 3.0 Reference" by Eve Maler of ArborText, Inc. and Terry Allen of Fujitsu Software Corporation. The parts of this document that



were borrowed from the Reference are Copyright © 1992, 1993, 1994, 1995, 1996, 1997 by HaL Computer Systems, Inc., O'Reilly & Associates, Inc., Fujitsu Software Corporation, and ArborText, Inc.

1.2. Why DocBook?

The DocBook format was designed by OASIS consortium specifically for technical documentation. It provides a rich set of tags to describe the content of your document.

Here is a number of key points that help understand what DocBook is:

Docbook is a markup language

> It is very similar to HTML in this respect. The tags give some structure to your document, and appear intermixed with the informational text.
>
> This pecular point makes it a revolution with respect to documentation translation, because the DTP phase (making the text look nice) is done once for all indirectly by tagging the original text. The translators only have to translate "in between the tags" and by pressing a single keystroke the translated output is generated.

It is made for technical documentation

> DocBook is perfectly suited for car engine parts documentation. However, it is strongly biased towards computer programs documentation.

It is maintained by an independant consortium

> The OASIS (http://www.oasis-open.org) consortium is in charge of maintaining and making this standard evolve through the DocBook Technical Committee. This is a guarantee of independance in front of proprietary software and standards.
>
> Major actors of the industry like Boeing or IBM are members of OASIS. Refer to the updated members list (http://www.oasis-open.org/html/members.htm) for further information.

Technically, DocBook is a SGML or XML DTD

> This means that one can take profit of the many SGML and XML aware tools. While DocBook as an XML implementation is quite recent, it has a long history as a SGML implementation.

DocBook is not a presentation language

> DocBook carefully cares about *not* specifying how the final documentation looks like. This allows the writer to concentrate on the organization and meaning of the document he or she writes. All the presentation issues are devolved to style sheets.
>
> This ensures all your documents have a consistent appearence, whoever should be the technical writer.

DocBook is customizable

> It is quite easy to customize the DTD to meet one's need thanks to its modular organization. But one must be aware that this must be done with respect to SGML/XML conventions and that it might introduce incompatibilities.
>
> If DocBook is used in conjunction with Norman Walsh's modular stylesheets, it is also possible to customize the way a DocBook file can be printed or put online too.

DocBook is comprehensive

> The large number of tags defined in DocBook guarantees that it can accomodate a wide range of situations and of processing expectations.
>
> This in turn makes it a bit difficult to learn, but one can manage writing documentation knowing only a limited set of tags and referring to the reference documentation when needed.

DocBook uses long and understable tags

> Example of such tags are <itemizedlist> or <literallayout>. This makes a DocBook text much easier to read than an HTML source for example. As a drawback, it can also become a bit tedious to type those long tags, but specialized modes in usual editors (like Emacs' psgml mode) can help out of this. One can use authoring tools as well.

DocBook does not ensure ascending compatibility between major releases

> While this might seem a drawback, in fact it is not, because it ensures a clean design even if wrong choices have been made previously by the DocBook Commitee at OASIS, and because documents written with different DTDs can coexist on a same computer system.

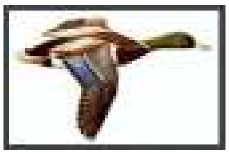
Some of those key points are discussed in more detail in the next sections.

1.3. Your World View

Most people who do word processing or typesetting use a WYSIWYG word processor or a typesetting system in which they type explicit markup instructions which tell the typesetter how to position text on the page (such as TeX and troff).

Both of these approaches suffer from a few serious problems. The biggest one is longevity of the document: eternal information (the profound things you type) is interspersed with information that will be obsolete (the typesetting information).

Another big problem with this old approach is lack of *structure*: the markup did not express content, but rather page layout. Let's say you are interested in indexing a bunch of papers written in TeX. It would be rather easy to index all occurances of boldface text, but that's not interesting at all! Instead, it would be really useful to index all function names in an API. With old typesetting approaches you would need artificially intelligent software that could understand the text and say "aha! this must be the definition of a function in the API".



So your old world view of writing a document and having the main challenge be how to mark it up to look good on paper is a poor one. Your challenge should be how to mark your document up to emphasize *semantic content.*

1.4. Markup based on content

So how do you mark your documents such that useful information can be extracted and indexed? The approach in DocBook is to provide a very rich set of markup tags that all relate to the structure and nature of the document's content.

To give you a couple of examples of tags that could help with generating automatic indices: <attribution> and <command>. If you have a large body of documentation (for example, all Sun software and hardware is documented with DocBook) you can do a very easy search for any document that discusses a command called **mount**, or a quote attributed to Ken Thompson. On top of that, with such a structured search you would only find occurances of **mount** *when it is a command name*, and of Thompson *when he is the author of a quote*.

Now imagine for a moment what would happen if the entire World Wide Web used a rich content–based markup language instead of HTML: a search engine would give you the information you need without all the extra references which just happen to use those words casually. A search for `mount` on the web would almost certainly not find you references on the UNIX **mount** command.

So a rich markup language like DocBook is a good idea from many points of view, but it can also be difficult to use. DocBook has hundreds of tags (as opposed to just a few in HTML), so you might find the learning curve steep. That is true, and the only way around that is to write documentation on how to use DocBook!

On the other hand, once you are quite familiar with DocBook it will not slow you down too much to type in markup all the time. Keep in mind that most of the time a person is not writing, but rather worrying about meta–level problems with their document. If you use DocBook well you will spend a bit more time writing and a lot less time worrying about other issues like the layout on paper. (There is nothing you can do about it anyway!)

Chapter 2. Getting started

2.1. Presentation of the Tools

This section describes how to work with DocBook on Unix-like systems like Linux. If you unfortunately are working on some other operating system, you might need to gather and configure the needed tools by yourself or to buy a commercial solution.

The technology around DocBook evolves. In the past:

- DocBook started as a “SGML application”, just like HTML did.
- DocBook documents used being converted with the help of DSSSL style sheets.
- The PDF layout was accomplished through TeX typesetting engine.

Now, on the contrary:

- DocBook follows the syntax of XML, just like XHTML does.
- It is converted to other formats by XSLT style sheets.
- PDF layout being done through a XSL-FO engine.

The tools used to process DocBook files have to keep up with this evolution. We will successively present the DocBook-tools (SGML/DSSSL technology) and XSLTProc/FOP (XML/XSL technology).

> **Note:** There is another distribution of SGML/DSSL tools named "SGML-Tools (Lite)". More information is available at http://www.sgmltools.org.

2.2. The Old Way: the DocBook-tools

The DocBook-Tools consist of several packages that work together to convert DocBook SGML files into many other formats, and to perform other miscellaneous operations. The output formats include:

- HTML
- TeX and DVI
- PostScript
- RTF
- PDF
- Man pages
- TexInfo

> **Note:** We have said that the DocBook-tools were made for SGML DocBook. That's true, but they can also handle XML DocBook files.

The home page of the DocBook-Tools project is at http://sources.redhat.com/docbook-tools and the packages themselves can be obtained from ftp://sources.redhat.com/pub/docbook-tools/new-trials/ or one of its mirrors. Some commercial distributions have adopted the DocBook-Tools and provide them on their CD.

The DocBook-Tools include the following packages:

sgml-common
: basic SGML declarations and tools

jade
: a SGML and DSSSL stylesheets engine

jadetex
: a set of TeX macros used by the files generated by Jade

docbook-dtdXX-sgml
: DocBook's SGML DTD (there's one package per version of the DTD)

docbook-style-dsssl
: Norman Walsh's DSSSL stylesheets for DocBook

perl-SGMLSpm
: Interface between Perl and SGML

docbook-utils
: Helper shell scripts and perl utilities

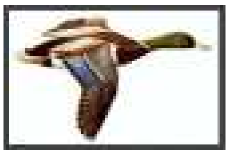
psgml
: Major mode for Emacs to edit SGML files

We will describe here only the RPM version of these packages. For other package formats, you will need to adapt the instructions below.

1. Download the packages
2. Install them

as in the following session example:

```
$ ncftp ftp://sources.redhat.com
ncftp> cd pub/docbook-tools/new-trials/RPMS
ncftp> mget i386/*.rpm
```



```
ncftp> mget noarch/*.rpm
ncftp> quit
$ su
Password: ultra-sucure
# rpm -ih sgml-common*.rpm
# rpm -ih jade*.rpm
# rpm -ih jadetex*.rpm
# rpm -ih docbook-dtd44-sgml*.rpm
# rpm -ih docbook-style-dsssl*.rpm
# rpm -ih perl-SGMLSpm*.rpm
# rpm -ih docbook-utils*.rpm
# exit
```

Note: The order of installing packages is important.

Note: When you are upgrading, rather than installing for the first time, the `rpm -ih` steps should be replaced with `rpm -Uh`.

You are now ready to edit SGML/DocBook documents, then convert them to other formats. To do that, you will use one of the following commands:

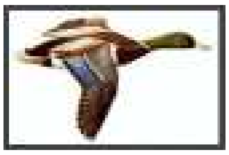
- **docbook2html** => HTML
- **docbook2ps** => PostScript
- **docbook2pdf** => PDF
- **docbook2rtf** => Rich Text Format
- etc...

2.3. The New Way: XSLTProc and FOP

Since the XML-based tools are much easier to use than the SGML-based tools, there is no software wrapper like the DocBook-tools around them. The basic utilities can be used directly.

The output formats are:

- HTML
- PDF
- MIF
- PCL
- PS
- SVG



RTF is planned by the experimental releases of FOP.

On a RPM-based system, you will need the following software packages:

docbook-dtdXX-xml
: DocBook's XML DTD (there's one package per version of the DTD)

docbook-style-xsl
: Norman Walsh's XSL stylesheets for DocBook

libxml2
: Daniel Veillard's XML library, needed by libxslt

libxslt
: Daniel Veillard's XSLT conversion engine

java-1_4_2-sun
: ... or any other Java Runtime Environment, needed by fop

fop
: XSL-FO processor

Most of these packages are usually preinstalled with your Linux distribution. For the other ones, use **rpm -ih *`packagename`***.

You are now ready to edit XML/DocBook documents, then convert them to other formats. To do that, you will need the following commands:

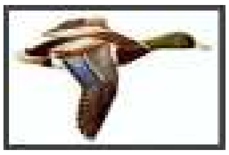
- **xsltproc**: DocBook to HTML or XSL-FO
- **fop**: XSL-FO to PDF, PS, etc.

2.4. My First DocBook File

First, you need a DocBook file to convert. Take any simple text editor you want and type (or cut'n'paste) the following lines:

Example 2-1. A minimal DocBook file

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN"
  "/usr/share/xml/docbook/schema/dtd/4.4/docbookx.dtd">

<book>
```



```
<bookinfo>
<title>Hello, world</title>
</bookinfo>

<chapter>
<title>Hello, world</title>

<para>This is my first DocBook file.</para>

</chapter>
</book>
```

Then save it under, say, *myfile.docbook*. If you are using SGML tools, the long pathname between quotes on the third line is not needed.

To convert this file from DocBook format to HTML format, use the command:

```
$ docbook2html myfile.docbook
```

or

```
$ xsltproc /usr/share/xml/docbook/stylesheet/nwalsh/current/html/docbook.xsl myfile.docbook -o myfil
```

according to your toolkit (DocBook-tools or XSLTProc).

Jade or xsltproc will chug away, and if your document has no errors you will get one or more HMTL files. Use a Web browser to look at them, `book1.htm`.

If you do get errors, read through the error log and correct errors from the beginning of the list first. Often, an early error such as an unclosed tag will cause a lot more errors to occur later in the document.

2.5. Introducing the Style Sheets

Note: This section is a bit outdated. It should describe the XSLT style sheets as well.

Stylesheets can tune the conversion in a way the resulting files have more clever names. Change the <book> and <chapter> tags in the above example as follows:

Example 2-2. The minimal DocBook file, with some attributes

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN">

<book lang="en">
<!-- Please remark the "lang" attribute here -->
```

```
<bookinfo>
<title>Hello, world</title>
</bookinfo>

<chapter id="introduction">
<title>Hello, world</title>

<para>This is my first DocBook file.</para>

</chapter>
</book>
```

The text between <!-- and --> above is a comment; use it to attract the attention of someone reading the DocBook source. It will never get processed.

Now use the style sheet provided with the docbook-utils:

```
$ cp /usr/share/sgml/docbook/utils*/docbook-utils.dsl .
$ docbook2html -d docbook-utils.dsl#html myfile.docbook
```

Now the files should go to a `HTML` directory, and be named `index.html` and `introduction.html`, instead of having names like `book1.htm`. The main file will always be named `index.html` and the chapters like <chapter id="introduction"> will go to files named after the id attribute. This change has been accomplished through style sheet magic.

> **Note:** Use #print instead of #html to specify the right part of the style sheet to use if you try them with some command like **docbook2pdf** instead of **docbook2html**.

In fact the style sheets are a very powerful tool. They enable you to get rid of problems like "I want it to look like this". If you come to such questions while writing a DocBook file, then it means that something is going wrong in your approach of the things.

If you get a look to the style sheet file named `docbook-utils.dsl`, you’ll see that it is written in a cryptic language named DSSSL, that looks really like some LISP. This unfortunately means that some good programming knowledge is often required to tune the stylesheets.

More information on how to customize the stylesheets can be found on http://www.nwalsh.com/docbook/dsssl/doc/. More information on DSSSL can be obtained on http://www.jclark.com/dsssl/ (http://www.jclark.com/dsssl).

Chapter 3. Basic notions

3.1. Anatomy of a DocBook Tag

A DocBook tag consists of an element and attributes. For example, <chapter id="introduction"> contains the element chapter and the attribute id. The element modifies the text within the markup, and the element's attributes modify the element. For example, the chapter element says that all text included within the starting and closing tags should be treated as a chapter, while the id attribute labels the chapter so it can be linked to or used as a file name when DocBook is translated to another format.

Most DocBook tags contain a common set of attributes. The common attributes that you will be using most often are lang, which specifies the language of the data inside the tag, and id, which labels the tag.

> **Important:** All (or nearly all) DocBook tags must have a start and an end tag. If you read through the DocBook Reference you will notice that not all DocBook tags are required to have both start and end tags. Omitting the ending tags where they are not required is not usually a good idea because it will work only with SGML DocBook and not with XML DocBook.
>
> Also, make sure to follow the proper case of the tag because XML is picky about the case of tags.

For detailed information about tags and a list of all available tags, see the DocBook Reference (http://www.oasis-open.org/docbook/documentation/reference/html/docbook.html) and the DocBook Quick Reference (http://www.docbook.org/tdg/en/html/quickref.html). Those guides currently address the version 3.1 of DocBook but they can be applied with little modifications to earlier or latter versions.

3.2. The Structure of a DocBook File

The tags covered in this section are listed below.

book - Book
article - Article
refentry - Equivalent of a man page
chapter - Chapter of a book or an article
sect1 ... sect5 - Sections and subsections of a chapter
title - Text of a heading or the title of a block-oriented element
para - Paragraph

Example 3-1. Chapters and sections

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN">

<book id="hello-world" lang="en">
```

```
<bookinfo>
<title>Hello, world</title>
</bookinfo>

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

<para>This is the introduction. It has two sections</para>

<sect1 id="about-this-book">
<title>About this book</title>

<para>This is my first DocBook file.</para>

</sect1>

<sect1 id="work-in-progress">
<title>Warning</title>

<para>This is still under construction.</para>

</sect1>

</chapter>
</book>
```

The above example shows a skeleton of the structural tags. The first line is the DTD declaration which indicates which DTD to use to process this document (namely DocBook version 4.4). This information will be described in more detail in the Document Type Declaration section.

Next comes the content model, which is <book> here. You can also use <article>, which is more lightweight than <book>, or <refentry> which is the equivalent of a UNIX man page.

Note the use of the lang attribute in the <book> tag. The language attribute should always be used to make it easy to determine what language in which the document is written.

After the <book> tag comes the meta information for the document which is encapsulated within the <bookinfo> tag. This information will be described in more detail in the Meta Information section.

Then come the chapters of your book, which may contain one or more section tags (<sect1> - <sect5>). Human-readable (not numerical) ID attributes for <chapter> and <sect> tags are required for two reasons:

- Labelling all of the sections of your document allows you to easily cross-reference your document with hyperlinks.
- Jade uses the ID’s of Chapters to name the output files, so if you do not include ID’s for all your Chapters, the file names will be different each time the docs are updated, which wastes space in CVS.

Chapters and sections must contain at least a <title> and an empty <para> tag. The place where certain elements can occur, cannot occur or must occur is defined by the DocBook DTD, and is covered in detail by the Reference Guide.

Content in DocBook is contained within a <para> tag, which is very similar to the <p> tag in HTML and LinuxDoc except that it must always have a closing </para> tag. Each time there can be a line break in some text (like in a list item), it means that the text will have to be enclosed in <para> tags.

Let’s summarise and extend what we have seen so far. A book will be structured in the following way:

```
book
  meta information
  chapter
    sect1
      sect2
    sect1
  chapter
    sect1
  appendix
    sect1
  appendix
    sect1
    ...
  glossary
```

An article will be structured in the following way:

```
article
  meta information
  sect1
  sect1
    sect2
  sect1
  ...
```

Chapter 4. The Document Type Declaration

In "real life", the document type declaration (the very first lines of your DocBook file) is slightly more complicated than the single line presented in the document skeleton in the previous section.

We'll examine in the following sections why it makes sense to add declarations here.

4.1. Using Entities for Shared Text

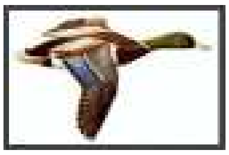
Example 4-1. Entities used to share text

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN" [
  <!ENTITY cereals "<productname>Frobozz Cerals</productname>">
  <!ENTITY orange-juice "<productname>Hourmade Orange Juice</productname>">
]>

<book id="marketing-study" lang="en">

<bookinfo>
<title>Marketing study about &cereals;</title>
</bookinfo>

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

<para>
This study describes the expected impact of the
new &cereals; product, and examines the opportunity
of an advertising campain combined with the &orange-juice;.
</para>

</chapter>
</book>
```

You find there "entity definitions" that makes &cereals; a synonymous for the "Frobozz Cereals" product name, marked up as <productname>Frobozz Cereals</productname>.

The example will look something like this when converted:

This study describes the expected impact of the new Frobozz Cerals product, and examines the opportunity of an advertising campain combined with the Hourmade Orange Juice.



Proceeding like this has a number of advantages:

- It spares the hassle of typing several times the very same thing
- It allows to centralized the changes to commonly used sentences into one single place
- If the entity name is chosen with caution, it makes the source text more legible

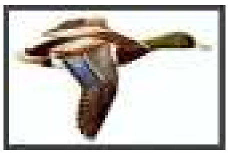
4.2. Using Entities to Include Other files

The document type declaration is also the right place to include other files. The documentation usually tend to become very big, so it's a good idea to split it into several files that are included into a single main file like this:

Example 4-2. Entities used to include other files

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN" [
  <!ENTITY introduction SYSTEM "introduction.docbook">
  <!ENTITY marketing SYSTEM "marketing-plan.docbook">
  <!ENTITY advertising SYSTEM "advertising-campain.docbook">
  <!ENTITY conclusion SYSTEM "conclusion.docbook">
  <!ENTITY cereals "<productname>Frobozz Cerals</productname>">
  <!ENTITY orange-juice "<productname>Hourmade Orange Juice</productname>">
]>

<book id="marketing-study" lang="en">

<bookinfo>
<title>Marketing study about &cereals;</title>
</bookinfo>

&introduction;
&marketing;
&advertising;
&conclusion;

</book>
```

Of course, you must create files named `introduction.docbook`, `marketing-plan.docbook`, etc, that will hold your chapters, to make this example work.



4.3. Identifying files with formal public IDs

> **Note:** This is quite advanced technology. Feel free to skip it if you want to be told about more basic topics.

You can always identify files to include by their name and path information (a system identifier). But there are several reasons why you may want to use public IDs, or FPIs, instead. The biggest reason is to make it easier to move files without having to change your documents. Another reason is to make it easier to exchange your files with other people or groups where the directories on their system may be different. FPIs also allow you or your company to claim ownership of important information, such as your DTD.

With FPIs, you identify a file by an abstract name in your document and then supply the location of that file in a catalog, sometimes called a mapping file or entity manager. The catalog is another, separate file from your document.

If a file that is used in your document moves, you simply change its location in the catalog rather than changing the location in your document or any other document that uses it. If you exchange files with someone else, or simply move the files to a new computer with different directories, you only have to change the location information once in the catalog.

FPIs must have a specific structure. Two slashes are used to mark the separation between each part of the structure, such as:

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

Registration

  The first character indicates whether the FPI is formally registered (+) or not (-) with an ISO approved registration service. If you define your own FPIs and don’t register them, use the hyphen.

Owner

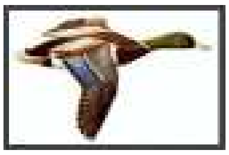
  The owner of the file is the second part of the FPI. This can be a company, an organization, or a person.

Keyword

  There are several keywords that indicate the type of information in the file. Some of the most common keywords are DTD, ELEMENT, 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. For a DTD like DocBook, it's the language of the markup text, not the language of the document itself!

To use those Formal Public Identifiers in your document, replace the SYSTEM keyword with PUBLIC in the Document Type Declaration at the beginning of your file. Then, for the systems to know where to find the file, you will neeed to establish a correspondance between the FPI and the real filename in a "catalog". Look into the files named "catalog" under `/usr/share/sgml` to see how this works.

4.4. Using Marked Sections to Handle Conditional Content

Note: This is quite advanced technology. Feel free to skip it if you want to be told about more basic topics.

Please be aware that marked sections are not supported by XML. Skip this section too if you are using XML.

Sometimes you need to have different versions of the content for different purposes. There are several ways to do this using SGML, one of which is called marked sections. A simple example of conditional content might be the description of keys used in a software program where they appear in boxes in the printed manual but are blue inside brackets on the web site or CD. Rather than have two separate versions of the conventions for each output of the manual, you can use marked sections to keep both variations in the same document.

There are different types of marked sections, but the types that allow you to control conditional content are ignore/include sections. These markers act like on/off switches to allow content to be included or ignored in different situations.

The marker for an include marked section looks like this:

```
<![INCLUDE [ keys are boxed, such as <key>F1</key> ]]>
```

while an ignore marked section looks like this:

```
<![IGNORE [ keys are blue inside brackets, such as <key>F1</key> ]]>
```

INCLUDE and IGNORE are the keywords that tell the SGML system what to include or skip. As this example shows, marked sections can contain both text and tags as long as the tags within the markers are balanced (if a tag starts inside a marker, then it ends inside the same marker).

In this example, you leave the markers for the print version as INCLUDE and the markers for the electronic version as IGNORE when you print a master copy. When you create the electronic book or HTML for the web site, you change the markers for the print version to IGNORE and the markers for the electronic version to INCLUDE. This works just fine, unless you have several different sections you need to include or ignore together - it's cumbersome to change each one manually and you can easily make a mistake.

So instead, you can define parameter entities under any names you want and then change the entities to turn the include/ignore switches. To do this, you add parameter entity declarations in the internal subset at the top of your master SGML document. For example:

```
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook V4.4//EN" [
...
<!ENTITY % hardcopy "INCLUDE">
<!ENTITY % softcopy "IGNORE">
...]>
```

You then use the names for each marked section, like this:

```
<![%hardcopy; [ keys are boxed, such as <key>F1</key> ]]>
<![%softcopy; [ keys are blue inside brackets, such as <key>F1</key> ]]>
```

To print the master copy, you leave the entity declarations as they are shown above. The SGML system interprets each %hardcopy; it finds as INCLUDE and includes those marked sections. The %softcopy; is interpreted as IGNORE and those sections are skipped. When you're ready to produce the electronic version, you only have to change the entity declarations at the top of the file, like this:

```
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook V4.4//EN" [
...
<!ENTITY % hardcopy "IGNORE">
<!ENTITY % softcopy "INCLUDE">
...]>
```

With this single change, the electronic versions are included and the printed versions are skipped.

Marked sections can be simple, but they are not always the best choice to manage conditional text. They are best if you use them sparingly and in very clear situations - it's easy to figure out when to use them and when to change the INCLUDE/IGNORE switches. Some of the problems that they can create include:

- Marked sections can be nested (a marked section inside another marked section), but this can confuse your SGML system and may not produce the effect you want. For example, SGML systems can't properly handle an included marked section inside an ignored marked section.
- If you use lots of differently named sections, it's easy to lose track of the content and can make your SGML document invalid if some required structures are set to IGNORE.
- Finally, they are not supported in XML. Moving to XML could be difficult if you use marked sections a lot.

Chapter 5. Meta Information

The tags covered in this section are listed below.

bookinfo - Metainformation for a book
title - Text of a heading or the title of a block-oriented element
authorgroup - Wrapper for author information
author - Author of a document
authorinitials - Initials or other identifier for the author of a revision or comment
firstname - Given name
othername - Name component that is not a firstname, surname, or lineage
surname - Family name
keywordset - Set of terms describing the content of a document
keyword - Term describing the content of a document
releaseInfo - Information about a particular version of a document
revhistory - Revisions to a document
revision - Entry in revhistory, describing some revision made to the text
revnumber - Number of a revision
revremark - Description of a revision
abstract - Document summary
date - Date of publication or revision of a document

Example 5-1. DocBook metainformation

```
<bookinfo>
<title>The Twiddle Handbook</title>
<authorgroup>
<author>
<firstname>George</firstname>
<othername>N.</othername>
<surname>Ugnacious</surname>
</author>
</authorgroup>

<date>03/04/1999</date>
<releaseinfo>1.01.00</releaseinfo>

<abstract>
<para>
<application>twiddle</application> is an application specially
designed to do nothing you would ever want.
</para>
</abstract>

<keywordset>
<keyword>twiddle</keyword>
<keyword>sample application</keyword>
</keywordset>

</bookinfo>
```

The <bookinfo> tag contains all of the meta information for your document.

It is information that describes your document, as opposed as its contents that are in the various chapters. This allows to:

- retrieve a specific among many others on your local hard disk
- print some information on the front page, under the control of the style sheets
- have it retrieved by search engines if you put it online in HTML format
- etc ...

Chapter 6. Lists

DocBook lists are very similar to their counterparts in HTML except that DocBook contains several more types of lists for specialized purposes.

The tags covered in this chapter are listed below.

simplelist - List of single words or short phrases
member - Member of a simplelist
itemizedlist - List in which each entry is marked with a bullet, dash, or other dingbat
listitem - Wrapper for the elements of items in an itemizedlist or orderedlist
orderedlist - List in which each entry is marked with a sequentially incremented label
variablelist - List in which each entry is composed of sets of one or more terms with associated listitems
varlistentry - Wrapper for term and its associated listitem in a variablelist
term - Hanging term attached to a listitem within a varlistentry in a variablelist
segmentedlist - List of sets of information
segtitle - Title that pertains to one seg in each seglistitem
seglistitem - List item in a segmentedlist
seg - Component of a segmentedlist
qandaset - A question-and-answer set
qandaentry - A question/answer set within a qandaset
question - A question in a qandaset
answer - An answer to a question posed in a qandaset
procedure - List of operations to be performed
step - Part of a procedure
substeps - Wrapper for steps within steps

6.1. The simplelist

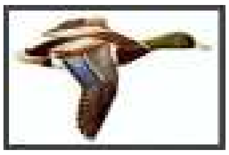
The easiest of all the lists to use is the simplelist. It is designed for lists of short phrases (like a grocery list) and only requires two tags for building the list as you can see in the example below. The <member> tag can only contain inline content, so a simplelist cannot contain other lists.

Example 6-1. A simplelist

```
<simplelist>
<member>Apples</member>
<member>Oranges</member>
<member>Bananas</member>
<member>Grapefruit</member>
<member>Black Beans</member>
</simplelist>
```

When converted, a simplelist will look something like this:

Apples



Oranges
Bananas
Grapefruit
Black Beans

6.2. The itemizedlist

An itemizedlist is similar to the simplelist except that each entry contains a paragraph instead of just a short phrase, allowing you to put more varied content in your list. ItemizedLists can contain other lists.

Example 6-2. An itemizedlist

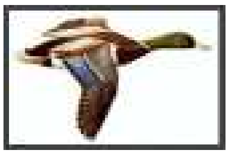
```
<itemizedlist>
<listitem><para>Apples - my favorite fruit.</para></listitem>
<listitem><para>Oranges - yummy, but sticky.</para></listitem>
<listitem><para>Bananas - they ripen too quickly!</para></listitem>
<listitem><para>Grapefruit - great when eaten in halves.</para></listitem>
<listitem><para>Black Beans - go well with rice.</para></listitem>
</itemizedlist>
```

The example will look something like this when converted:

- Apples - my favorite fruit.
- Oranges - yummy, but sticky.
- Bananas - they ripen too quickly!
- Grapefruit - great when eaten in halves.
- Black Beans - go well with rice.

6.3. The orderedlist

The orderedlist is like the itemizedlist except that each listitem is numbered or lettered. The numeration attribute specifies what kind of numbering will be used and can be one of the following values: arabic, upperalpha, loweralpha, upperroman, lowerroman. There are several other attributes that control the appearance of an orderedlist, see the DocBook Reference for details. orderedlists can contain other lists.



Example 6-3. An orderedlist

```
<orderedlist numeration="arabic">
<listitem><para>Wake up.</para></listitem>
<listitem><para>Eat Breakfast.</para></listitem>
<listitem><para>Take a shower.</para></listitem>
<listitem><para>Contemplate my navel.</para></listitem>
<listitem><para>Go to Sleep.</para></listitem>
</orderedlist>
```

The example will look something like this when converted:

1. Wake up.
2. Eat Breakfast.
3. Take a shower.
4. Contemplate my navel.
5. Go to Sleep.

6.4. The variablelist

The variablelist is similar to an HTML definition list. It is used when you have a list of terms and definitions. The variablelist consists of several tags: <varlistentry>, which is used to group related terms together, <term>, which contains the term, and <listitem>, which contains the decription of the term.

Example 6-4. A variablelist

```
<variablelist>
<varlistentry>
<term>Black Beans</term>
<listitem><para>My favorite black bean recipe is black bean
soup, but they also go well with rice.</para></listitem>
</varlistentry>
<varlistentry>
<term>Apples</term>
<term>Bananas</term>
<listitem><para>You can eat them straight, but they also go
well in salads and in desserts.</para></listitem>
</varlistentry>
</variablelist>
```

When converted, the example variablelist will look something like this:

Black Beans

    My favorite black bean recipe is black bean soup, but they also go well with rice.

Apples
Bananas

    You can eat them straight, but they also go well in salads and in desserts.

6.5. The segmentedlist

segmentedlists are used to list information in distinct fields like the contents of an address book. The name of each field is put inside of a <segtitle> tag. Then, use the <seglistitem> tag to start and end each set of data. The actual data is put in the <seg> tag.

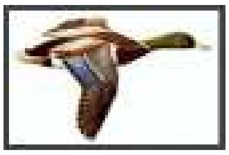
Example 6-5. A segmentedlist

```
<segmentedlist>
<segtitle>Name</segtitle>
<segtitle>Occupation</segtitle>
<segtitle>Favorite Food</segtitle>
<seglistitem>
<seg>Tux</seg>
<seg>Linux mascot</seg>
<seg>Herring</seg>
</seglistitem>
<seglistitem>
<seg>Konqui</seg>
<seg>The KDE Dragon</seg>
<seg>Gnomes</seg>
</seglistitem>
</segmentedlist>
```

This silly example looks something like this when converted:

Name: Tux
Occupation: Linux mascot
Favorite Food: Herring

Name: Konqui
Occupation: The KDE Dragon
Favorite Food: Gnomes



6.6. qandaset

The qandaset is a specialized list designed specifically to deal with sets of questions and answers, like you would see in a FAQ. Each set of questions and answers are contained within a <qandaentry> tag. The <question> and <answer> tags contain the questions and answers respectively.

Example 6-6. A qandaset

```
<qandaset>
<qandaentry>
<question>
<para>What are little boys made of?</para>
</question>
<answer>
<para>Snips and snails and puppy dog tails.</para>
</answer>
</qandaentry>
<qandaentry>
<question>
<para>What are little girls made of?</para>
</question>
<answer>
<para>Sugar and spice and everything nice.</para>
</answer>
</qandaentry>
</qandaset>
```

The qandaset looks something like this when converted:

1. What are little boys made of?

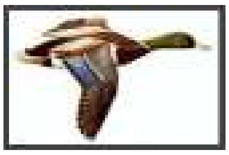
Snips and snails and puppy dog tails.

2. What are little girls made of?

Sugar and spice and everything nice.

6.7. Procedures

Procedure lists are a specialized orderedlist used for listing step-by-step procedures like you would find in a recipe or Linux HowTo.



Example 6-7. A procedure list

```
<procedure>
<title>Waking Up</title>
<para>This is what you must do to awaken.</para>
<step performance="required">
<para>
Bring yourself to a hypnopompic state, either from an ongoing dream or by use of
your internal clock. You may feel unable to move, but you will no longer be
dreaming. </para>
<para>Now you are ready for real-world readjustment.</para>
<substeps>
<step performance="optional">
<para>Roll over.</para>
</step>
<step performance="required">
<para>Squint out of one eye.</para>
</step>
</substeps>
</step>
<step performance="required">
<para>Yawn and rise from your bed.
</para>
</step>
</procedure>
```

The above example would look something like this when converted:

Waking Up

This is what you must do to awaken.

1. Bring yourself to a hypnopompic state, either from an ongoing dream or by use of your internal clock. You may feel unable to move, but you will no longer be dreaming.

   Now you are ready for real-world readjustment.

   a. Roll over.

   b. Squint out of one eye.

2. Yawn and rise from your bed.

Chapter 7. Tables

The tags covered in this section are listed below.

table - Table in a document
informaltable - Untitled table
thead - Heading row of a table
tfoot - Footer row of a table
tgroup - Wrapper for part of a table that contains an array along with its formatting information
tbody - Wrapper for the rows of a table or informaltable
row - Row in a tbody, thead, or tfoot
entry - Cell in a table
entrytbl - Subtable appearing as a table cell

Tables are used to organize data into a columnar format with optional titles, headers, and footers. DocBook tables come in two varieties: the table, which requires a title, and the informaltable, which does not have a title. All the other characteristics of these two table types are the same.

A table consists of formatting information and data entries. There are quite a few attributes that can be adjusted to tweak the display of your data. This tutorial will only cover the basic formatting attributes. For more details, you should refer to the DocBook Reference.

Tables begin with the <table> or <informaltable> tag. Next, define a title using the <title> tag if you are using a regular table. Finally, we get to the <tgroup> tag which contains all of the header, footer, and row information. You can have more than one tgroup if you wish to change formatting options for a section of the table. The <tgroup> tag has a number of optional formatting parameters, but the COLS attribute, which specifies the number of columns, is required. The <thead>, <tfoot>, and <tbody> contain the data in your table. Data in the thead appears at the top of the table, tbody appears in the middle, and tfoot appears at the end of the table.

Data in a table is contained in rows and entries, labelled with the <row> and <entry> tags respectively. Use the <row> tag to begin a row, an <entry> tag for each item of data, and a closing <row> tag to end that row. If you want to embed a table within a table, you must use the <entrytbl> tag.

Example 7-1. A table

```
<table>
<title>Mouse Mileage</title>
<tgroup cols="3">
<thead>
<row>
<entry>Month</entry>
<entry>Week</entry>
<entry>Feet Traveled</entry>
</row>
</thead>
<tfoot>
<row>
```

```
<entry>Total</entry>
<entry></entry>
<entry>1753</entry>
</row>
</tfoot>
<tbody>
<row>
<entry>August</entry>
<entry>1</entry>
<entry>987</entry>
</row>
<row>
<entry>August</entry>
<entry>2</entry>
<entry>657</entry>
</row>
<row>
<entry>August</entry>
<entry>3</entry>
<entry>109</entry>
</row>
</tbody>
</tgroup>
</table>
```

The above example would look something like this when converted:

Table 7-1. Mouse Mileage

Month	Week	Feet Traveled
August	1	987
August	2	657
August	3	109
Total		1753

Chapter 8. Graphics

Below is a list of tags related to graphical objects:

screeninfo - Information about how a screenshot was produced
screenshot - Representation of what the user sees or might see on a computer screen
mediaobject - A picture, a sound, a text, that can be encoded in several different formats at the same time, not rendered
inlinemediaobject - A picture, a sound, a text, that can be encoded in several different formats at the same time, to be re
imagedata - One of the formats encoding the image, such as EPS when printing and PNG when displaying online

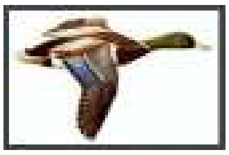
Documentation for graphical applications demands screenshots, pictures of icons and buttons, and other graphical elements. DocBook has tags to support screenshots, graphics, and inline graphics. The examples below contain the same picture as a screenshot and as an inline graphic.

Example 8-1. An inline media object

```
<para>
Here are a bunch of rectangles
<inlinemediaobject>
<imageobject> <imagedata fileref="rectangles.eps" format="EPS" /> </imageobject>
<imageobject> <imagedata fileref="rectangles.png" format="PNG" /> </imageobject>
<textobject> <phrase>A bunch of rectangles</phrase> </textobject>
</inlinemediaobject>
</para>
```

The <inlinemediaobject> tag is displayed alongside text. The fileref attribute of the <imagedata> tag contains the name of the graphic file, the format attribute contains the type of the graphic file, and the optional align attribute changes the alignment of the graphic. The example will look something like this when converted:

Here are a bunch of rectangles



Example 8-2. A screenshot

```
<screenshot>
<screeninfo>Colored Rectangles</screeninfo>
<mediaobject>
<imageobject> <imagedata fileref="rectangles.eps" format="EPS" /> </imageobject>
<imageobject> <imagedata fileref="rectangles.png" format="PNG" /> </imageobject>
<textobject> <phrase>A bunch of rectangles</phrase> </textobject>
<caption>
<para>Here are a bunch of rectangles</para>
</caption>
</mediaobject>
</screenshot>
```

The <screenshot> tag designates a screenshot with the <screeninfo> tag providing a textual description of the screen shot. The heart of the screenshot is the <mediaobject> tag which points to a graphic file as explained in the first example. A screen shot looks something like this when converted:

Here are a bunch of rectangles

Chapter 9. Links

These are the tags covered in this section:

anchor - Spot in text
email - Email address in an address
link - Hypertext link
ulink - A link that addresses its target by means of a URL, a Uniform Resource Locator

Linking in DocBook is very similar to linking in HTML. The <link> tag is used to link to any element with an ID in a document, and can also be used to link to other local files as well. The <ulink> tag is used when you need to link to a URL. The <email> tag is a specialized form of the ulink tag used for email addresses. <anchor> is used to mark a spot in the text that you want to reference later with a Link.

Example 9-1. Many kinds of links

```
<para id="mylink">
This is a paragraph that will be linked to with a link tag. Oh, and by
the way, check out <ulink url="http://www.kde.org">my favorite web
site</ulink> while you are here. <anchor id="anotherlink"/> Kilroy was
here.</para>
<para>
The above paragraph can be located <link linkend="mylink">here</link>.
My email address is <email>konqui@kde.org</email></para>
```

Note that Link uses the linkend attribute and ulink uses the URL attribute for their link targets. The above example would be displayed something like this when converted:

This is a paragraph that will be linked to with a link tag. Oh, and by the way, check out my favorite web site (http://www.kde.org) while you are here. Kilroy was here.

The above paragraph can be located here. My email address is <`konqui@kde.org`>

Chapter 10. Describing the Application’s Interface

10.1. Examples

These are the tags covered in this section:

literallayout - Wrapper for lines set off from the main text that are not tagged as screens, examples, or programlisting, i
example - Example of a computer program or related information
informalexample - Untitled example
programlisting - Listing of all or part of a program
screen - Text that a user sees or might see on a computer screen

There are many situations where you must include examples of source code, commands, or GUI actions in your documentation. DocBook has many tags to support these needs. Whenever you want to include examples in your document, just put an <example> or <informalexample> tag around the example text or graphic.

Example 10-1. An example

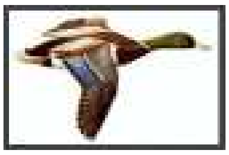
```
<example>
<title>A BASIC Example</title>
<programlisting>
10 PRINT "HELLO WORLD"
20 GOTO 10
</programlisting>
</example>
```

In this first example, we have a listing of a simple BASIC program. The code contained in the <programlisting> tag is displayed with the spacing and line breaks intact which is very useful for code examples and similar situations where you must preserve the literal formatting. The LiteralLayout and Screen tags work in the same way, but are used to indicate different types of content. screen contains output that would appear on the screen, while literallayout is used for any other text that must be rendered with line breaks and tabs.

The example would look something like this when converted:

Example 10-2. A programlisting

```
10 PRINT "HELLO WORLD"
20 GOTO 10
```



One problem can occur with the LiteralLayout, ProgramListing, and Screen tags: all text is rendered literally, but DocBook tags are still interpreted as tags and not text. What do you do when you need to show text without having your tags interpreted? The answer is to use <![CDATA [ ]]>, which labels the text contained within the inner brackets as character data that should not be interpreted by the SGML parser. Any text within the brackets will remain as-is after the conversion, so the example above will successfully reproduce its tags.

Example 10-3. CDATA usage

```
<example>
<title>some markup</title>
<screen>
<![CDATA[
<para>This is a DocBook example.</Para>
 ]]>

</screen>
</example>
```

This is what the markup example would look like when converted:

Example 10-4. Some markup

```
<para>This is a DocBook example.</para>
```

10.2. GUI Interface Elements

Accel - Keycap used with a meta key to activate a graphical user interface
KeyCap - Text printed on a physical key on a computer keyboard, not necessarily the same thing as a KeyCode
KeyCode - Computer’s numeric designation of a key on a computer keyboard
KeyCombo - Combination of input actions
KeySym - Key symbol name, which is not necessarily the same thing as a Keycap
MenuChoice - Menu selection or series of such selections
MouseButton - Conventional name of a mouse button
Interface - Element of a graphical user interface
InterfaceDefinition - Full or short name of a formal specification of a graphical user interface
GUIButton - Text on a button in a graphical user interface
GUIIcon - Graphic and, or, text appearing as a icon in a graphical user interface
GUILabel - Text in a graphical user interface
GUIMenu - Name of a menu in a graphical user interface
GUIMenuItem - Name of a terminal menu item in a graphical user interface
GUISubmenu - Name of a submenu in a graphical user interface
Action - Function invoked in response to a user event

One could almost say that there are too many tags in DocBook for describing GUI elements. Most of the tags listed above can be used in a variety of contexts, but a few, such as <keycap>, must be used within other tags. The example and explanation below will not cover all of the tags listed above. This list is for your convenience since the DocBook Reference does not group tags by their function.

All of the GUI tags can be used within the context of a regular paragraph. So if I wanted to talk about the Trash icon or the Empty Trash button, I would just use the <guiicon> and <guibutton> tags like this: <guiicon>Trash</guiicon> icon, <guibutton>Empty Trash</guibutton> button. Note that all GUI tags may also contain inline graphics.

Below is a more complicated example of GUI tag usage.

Example 10-5. guimenu and shortcut

```
<variablelist>
<varlistentry>
<term><menuchoice>
<shortcut>
<keycombo><keycap>Ctrl</keycap><keycap>n</keycap></keycombo>
</shortcut>
<guimenu>File</guimenu>
<guimenuitem>New</guimenuitem>
</menuchoice></term>
<listitem><para><action>Creates a new document</action></para></listitem>
</varlistentry>
<varlistentry>
<term><menuchoice>
<shortcut>
<keycombo><keycap>Ctrl</keycap><keycap>s</keycap></keycombo>
</shortcut>
<guimenu>File</guimenu>
<guimenuitem>Save</guimenuitem>
</menuchoice></term>
<listitem><para><action>Saves the document</action></para></listitem>
</varlistentry>
<varlistentry>
<term><menuchoice>
<shortcut>
<keycombo><keycap>Ctrl</keycap><keycap>q</keycap></keycombo>
</shortcut>
<guimenu>File</guimenu>
<guimenuitem>Quit</guimenuitem>
</menuchoice></term>
<listitem><para><action>Quits</action> application>Kapp</application></para></listitem>
</varlistentry>
</variablelist>
```

I hope your eyes haven't glazed over at the sight of all those tags! The most complicated part of this example is the <Shortcut> tag which labels keyboard shortcuts for menuitems. Shortcut contains either a KeyCombo or a single KeyCap that contains the key or group of keys the use would press to invoke that menuitem from the keyboard. It is important to use the KeyCombo and KeyCap tags within the Shortcut tag because it is incorrect to use character data (the Ctrl-q text for example) within a shortcut.

Other tags worth mentioning from the example are menuchoice, action, and application. menuchoice labels a menu choice and should contain the shortcut (if any) the name of the menu in GUIMenu, and the name of the menuitem in guimenuitem. Action simply labels a phrase that describes what the menuitem (or other interface element) does. application is a tag used to label the names of applications.

The example would look something like this when converted:

File⟶New (**Ctrl-n**)

Creates a new document

File⟶Save (**Ctrl-s**)

Saves the document

File⟶Quit (**Ctrl-q**)

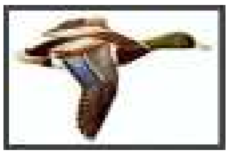
Quits Kapp

10.3. Command Line Elements

The following tags are used to label elements of a command:

type - Classification of a value
literal - Literal string, used in-line, that is part of data in a computer
userinput - Data entered by the user
symbol - Name that is replaced by a value before processing
replaceable - Content that may be replaced in a synopsis or command line
filename - Name of a file, possibly including pathname
prompt - Character indicating the start of an input field in a computer display.
paramdef - Data type information and the name of the parameter this information applies to
parameter - Part of an instruction to a computer
option - Option for a computer program command
envar - Environmental variable
command - Executable program, or the entry a user makes to execute a command
cmdsynopsis - Synopsis for a command
arg - Argument in a cmdsynopsis
computeroutput - Data presented to the user by a computer
funcsynopsis - Syntax summary for a function definition
funcsynopsisinfo - General information on how to use the function
funcprototype - Prototype of the function
funcdef - Name and return value of the function
function - Name of the function
paramdef - Name and type of a function parameter
parameter - Name of a function parameter

There are two situations in which you want to describe a command: showing an example of a



command typed on the command line and a detailed description of all of the arguments and options to a command like you would see in a man page.

DocBook supports both of these contexts with the <command> and <cmdsynopsis> tags.

Example 10-6. A command and its output

```
<screen>
<prompt>bash$</prompt> <command>twiddle <replaceable>myfile</replaceable>
</command>
twiddling myfile.....done!
</screen>
```

Would appear as:

```
bash$ twiddle -c 1 myfile

twiddling myfile.....done!
```

The command tag can also be used within a paragraph to mark the name of a command. For example:

```
The <command>twiddle</command> command is used to twiddle
files. Twiddled files will be marked with the .twid extension, so if I <command>twiddle<
<replaceable>myfile</replaceable> then it will become
<replaceable>myfile.twid</replaceable>. Errors are written to the
file <filename>twiddle.err</filename>.
```

The **twiddle** command is used to twiddle files. Twiddled files will be marked with the .twid extension, so if I **twiddle** *`myfile`* then it will become *`myfile.twid`*. Errors are written to the file `twiddle.err`.

The <prompt> tag is simply used to label the prompt in a command line. Replaceable labels text that should be replaced by the user. In the example, `myfile` is just an arbitrary name for a file since we don't know and don't care what the name of the file is, we just want to show how the command is used. If a filename in a command is known, use the <filename> tag instead.

Marking up a cmdsynopsis is a bit more difficult. Here is an example from the DocBook Reference:

Example 10-7. A commandsynopsis

```
<cmdsynopsis>
   <!-- This is a synopsis for the command foo.
        The options -a and -x are optional and exclusive
        The option -c takes a cheese and is optional and repeatable
        The options -t and -k are referred to in another fragment
        The options -i, -j, and -k are required and exclusive
        The option -f takes a filename and is required
        The -t and -k options specify the kind of milk and mold in an
            optional and repeatable group
    -->
    <command>foo</command>
    <group>
      <arg>-a</arg>
      <arg>-x</arg>
    </group>
    <group>
    <arg rep="repeat">-c <replaceable>cheese</replaceable></arg>
    <synopfragmentref linkend="cheesetype">cheesetype</synopfragmentref>
    </group>
    <group choice="req">
      <arg>-i</arg>
      <arg>-j</arg>
      <arg>-k</arg>
    </group>
    <arg choice="req">-f <replaceable>filename</replaceable></arg>
    <synopfragment id="cheesetype">
      <group rep="repeat">
         <arg>-t <replaceable>milk</replaceable></arg>
         <arg>-k <replaceable>mold</replaceable></arg>
      </group>
    </synopfragment>
  </cmdsynopsis>
```

Which looks like this:

foo [-a | -x] [-c *cheese*❶*cheesetype*] {-i | -j | -k} {-f *filename*}

❶ [-t *milk* | -k *mold*]...

A cmdsynopsis contains one command, groups of related args, independent args, and synopfragments. The <arg> labels arguments to the command. arg has two attributes: choice and rep. choice is used to indicate whether the tag is optional (the default), required (req), or to be displayed without any decoration (plain). The <group> tag is used to group together related args. synopfragment is the most complicated of the cmdsynopsis tags. It is used to provide a more detailed description of options for an argument. A synopfragment consists of two parts: the synopfragment, which contains the additional Args, and the synopfragmentref which points to the detailed description.

10.4. Describing an API

DocBook has a rather detailed way of marking up descriptions of function behaviour. The tag that introduces it is <funcsynopsis>. Here is an example:

Example 10-8. Describing a function in a C library API

```
<funcsynopsis>
<funcsynopsisinfo>#include <stdlib.h></funcsynopsisinfo>
<funcprototype>
<funcdef>double <function>atof</function></funcdef>
<paramdef>const char *<parameter>nptr</parameter></paramdef>
</funcprototype>
</funcsynopsis>
```

Here is how it looks:

```
#include <stdlib.h>
double atof(const char *nptr);
```

Chapter 11. Miscellaneous Useful Tags

11.1. Labelling Tags

Application - Name of a software program
Markup - String of formatting Markup in text, which it is desired to represent literally

The application tag labels the name of an application. The <markup> tag is used to label marked up text, such as HTML or TeX tags.

11.2. Formatting Tags

emphasis - Emphasized text
subscript - Subscript
superscript - Superscript

These tags are fairly self-explanatory. *This is an emaphasized sentence with* $^{\text{superscripted}}$ *and* $_{\text{subscripted}}$ *text.*

11.3. Warnings, Tips, and Notes

The following tags are used to set off paragraphs from the rest of the text.

caution - Admonition set off from the text
important - Admonition set off from the text
note - Message to the user, set off from the text
tip - Suggestion to the user, set off from the text
warning - Admonition set off from the text

All of the above tags wrap around paragraphs like in this example:

```
<warning><para>Danger, Will Robinson!</para></warning>
```

The example would look something like this:

Warning

Danger, Will Robinson!

Chapter 12. Where to Go Next

There are several useful parts of DocBook that are not discussed in this Tutorial. These include Callouts, Indexes, Glossaries, and Reference Pages. If you need to use any of these features, you should consult the DocBook Reference and/or purchase a DocBook book.

12.1. DocBook Resources

Below are a list of resources that will help you acquire and get started with DocBook.

- DocBook technical committee (http://www.oasis-open.org/docbook/) page at OASIS
- DocBook - the Definitive Guide (http://www.oreilly.com/catalog/docbook/chapter/book/docbook.html) online book
- DocBook 3.1 Quick Reference (http://www.oreilly.com/catalog/docbook/chapter/book/quickref.html)
- Norman Walsh's DocBook Site (http://nwalsh.com/docbook/)
- Jorge Godoy's DocBook HOWTO (http://docbook.godoy.homeip.net:81/sgml/docbook/howto/)
- Mark Galassi's DocBook Intro (http://nis-www.lanl.gov/~rosalia/mydocs/docbook-intro.html)
- FreeBSD Documentation Primer (http://www.freebsd.org/doc/en_US.ISO8859-1/books/fdp-primer/index.html)
- DocBook Install mini-HOWTO (http://www.tldp.org/HOWTO/mini/DocBook-Install/index.html) - useful if you want to compile yourself your tools
- Bob DuCharme's PSGML tips (http://www.snee.com/bob/sgmlfree/emcspsgm.html) for emacs

Appendix A. Licence

This document is released under Free Documentation licence; the terms of this licence are detailed below.

A.1. Free Documentation Licence

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

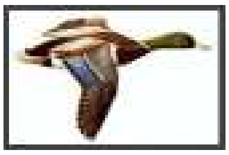
3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the

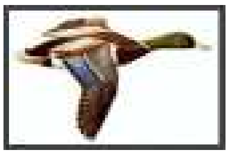


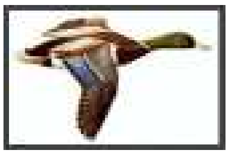
Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document,





If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix B. Emacs PSGML mode tips

There are a few SGML editors. A well known one is emacs with PSGML mode. It has emacs-style completion *on tags*.

Here are a few tips on how to use this mode. First install the "psgml" package, then load a DocBook file into Emacs.

> **Note:** Emacs with PSGML mode does not support filenames ending in .docbook. Your file name has to end in .sgml or .xml like *`myfile.sgml`*.

If you type "C-c C-e" it will prompt you for an element, and offer as completions only the valid elements at that point.

Once it inserts the element, it inserts it with any required following elements along with a comment saying which ones you could put later on.

As an example, I just went to a DocBook buffer and typed

```
C-c C-e variab<SPACE BAR><RETURN>
```

and it inserted this text in the buffer:

```
  <variablelist>
    <varlistentry>
      <term></term>
      <listitem>
      <!-- one of (epigraph authorblurb abstract highlights comment bridgehead anchor sidebar procedure msgset table figure example equation informaltable informalexample informalequation graphicco graphic blockquote address simpara para formalpara funcsynopsis cmdsynopsis synopsis screenshot screenco screen programlistingco programlisting literallayout warning tip note important caution variablelist simplelist segmentedlist orderedlist itemizedlist glosslist calloutlist) -->
      </listitem>
    </varlistentry>
  </variablelist>
```

Another example:

```
C-c C-e i<SPACE BAR>
```

and it shows me the following completions:

```
Click mouse-2 on a completion to select it.
In this buffer, type RET to select the completion near point.

Possible completions are:
important                    indexterm
informalequation             informalexample
informaltable                itemizedlist
```

Glossary

ASCII

(American Standard Code for Information Interchange) This standard character encoding scheme is used extensively in data transmission.

ANSI

(American National Standards Institute) This group is the U.S. member organization that belongs to the ISO, the International Organization for Standardization.

attribute

An attribute provides more information about an element such as classification level, unique reference identifiers, or formatting information.

CCITT Group 4

(International Consultative Committee on Telegraphy and Telephony) This CALS standard for raster graphics incorporates tiling, which divides a large image into smaller tiles. You can exchange graphic files in CCITT/4 format in a compressed state so they take up much less file space.

CITIS

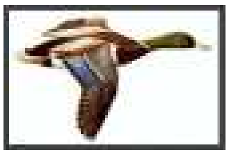
(Contractor Integrated Technical Information Service) As part of CALS Phase II, CITIS is a draft functional specification for services. DoD acquisition managers designed CITIS as a plan to gain access to product-related digital technical information.

CGM

(Computer Graphics Metafile) CGM is one of the CALS standard formats for representing 2–D technical illustrations. CGM is an object-oriented graphic format.

DSSSL

(Document Style Semantics and Specification Language) Style sheets language for SGML documents derived from Scheme language and normalized under the number ISO/IEC 10179:1996.



DTD

(Document Type Definition) A DTD is the formal definition of the elements, structures, and rules for marking up a given type of SGML or XML document. You can store a DTD at the beginning of a document or externally in a separate file.

EDI

(Electronic Data Interchange) This is a set of computer interchange standards for business documents such as invoices, bills, and purchase orders.

element

An element is a piece of data within a document that may contain either text or other subelements such as a paragraph, a chapter, and so on.

element declaration

A statement in the DTD defining an element and declaring the order in which it may appear in the document and what other elements it may include.

entity

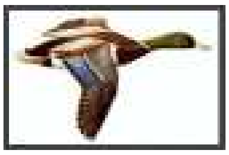
An entity is a self-contained piece of data that can be referenced as a unit. You can refer to an entity by a symbolic name in the DTD or the document. An entity can be a string of characters, a symbol character (unavailable on a standard keyboard), a separate text file, or a separate graphic file.

entity declaration

A statement in the DTD or document that assigns an SGML or XML name to an entity so you can reference it.

FOSI

(Formatting Output Specification Instance) A FOSI is used for formatting SGML documents for printing and other outputs. It is a separate file that contains formatting information for each element in a document.



HTML

(HyperText Markup Language) This is the format of files published on the World Wide Web. HTML is an application of SGML; to author in HTML using SGML-based authoring software, you simply need the HTML DTD.

IGES

(Initial Graphics Exchange Specification) The IGES standard for engineering, product design, and manufacturing drawings is one of the CALS standard graphics formats.

Internet

The Internet is a worldwide communications network originally developed by the U.S. Department of Defense as a distributed system with no single point of failure. The Internet has seen an explosion in commercial use since the development of easy-to-use software for accessing the Internet.

ISO

(International Organization for Standardization) The ISO is an industry-supported organization that establishes worldwide standards for everything from data interchange formats to film speed specifications.

markup

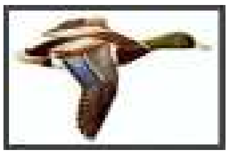
Markup is anything added to the content of the document that describes the text.

parser

A parser is a specialized software program that recognizes SGML or XML markup in a document. A parser that reads a DTD and checks and reports on markup errors is a validating parser. A parser can be built into an SGML or XML editor to prevent incorrect tagging and to check whether a document contains all the required elements.

tag

In the world of SGML and XML, a tag is a marker embedded in a document that indicates the purpose or function of the element. Each element has a beginning tag and an end tag.



World Wide Web

Often referred to as WWW or the Web, this usually refers to information available on the Internet that can be easily accessed with software usually called a “browser.” Organizations publish their information on the Web in a format known as HTML; this information is usually referred to as their “home page” or “web site”.

XSLT

(eXtended Style Sheet transformations) Style sheets language for XML documents that is itself a dialect of XML.