# Documenting your code with DocBook



## What documentation is for and why it doesn't get written

Traditionally speaking, users want documentation but coders don't want to write it. It is not as black and white as this of course. Some coders write documentation as well as software, and there are people who use software but do not want to bother reading documentation. The term “documentation” itself means different things to different people.

There are people who, when they get their latest electronic toy from the shop home, will immediately switch it on and start using it hoping that it will be easy enough to use that they won't have to patiently read the lengthy manual if they get stuck. Then there are others who will read the instruction booklet cover to cover before plugging it in.

Documentation is not limited to paper manuals. For software it can mean on-line help in the form of web pages and pop-up windows, or the manual pages available from the **man** command or the GNOME help tool.

When I use a well-designed program I do not expect to be looking for documentation immediately, because I hope that it will be intuitive and helpful on its own. A nice looking graphical interface with obvious layout, or (for tools where a graphical interface is overkill) command line switches similar to those used elsewhere, go a long way towards making me start using the program first before hunting for its documentation. All the same, I would be disappointed if there was none to be found when I eventually came to look up how to do something, or wanted to find out whether a particular behaviour is intentional or a mistake.

## Why DocBook is good/bad

DocBook is a format for writing technical documentation. It uses XML mark-up to create the document structure and to distinguish certain types of word or phrase. It looks a little like HTML, but uses a different set of tags.

There are tools available for converting DocBook documents into a variety of formats including HTML for web pages, PDF for printing, on-line manual pages, and plain text. Unfortunately the maturity of the tools for doing this varies.

There can be a moderate learning curve when starting to write things using DocBook. Depending on your background there might be an initial shock when switching over from a layout-based way of doing things (bold here, make this writing bigger) to semantic mark-up (emphasize this, make this a heading). This is easier if you can trust the tools to lay it all out correctly so the end result is nice to look at!

Then there is the vocabulary of DocBook XML tags to learn. DocBook uses a different set of tags to HTML, although if you are used to HTML or any

similar mark-up language, it will be easier than if you are not. Once you start writing you will pick them up quite quickly, and for the more obscure ones there is an on-line book which documents DocBook. Your editor might even prompt you with a list of available tags—see later.

Another hurdle is the separation between the DocBook format and the tools for editing and processing it. You may need to experiment a little to find out which tools work best in your environment. On the plus side there is usually a choice between different free and commercial tools, and one of them is bound to suit your needs.

It may be a bit of a climb but, as someone said to me, the view from the top of the hill is worth it. Once you get going with it you will have a single source of documentation for both print and on-line formats. It will be laid out on the page (or the browser) intelligently and consistently. Consistency is an even bigger plus when you start writing more extensive documentation. While it is good for a document to have a consistent way of expressing things such as sub-headings, file names, function names and so on, it is even better for a whole set of documents to use the same layout. The beauty of structured mark-up systems such as DocBook is that you can alter how a sub-heading should look and have it reflected across all of your documentation.

There is a whole area of documentation called “literate programming” which aims to intermingle software with instructions about how it should be used, for example with comments above function definitions. There is a large amount to be said about it, but I think it deserves a discussion of its own another time.

## Short history of DocBook

DocBook was designed in the early 1990s by HaL Computer Systems and O'Reilly & Associates (one of the first publishers to use SGML to produce their books). It was first formulated as an SGML document type definition, but is now expressed in XML. A document type definition, or DTD, is a specification of the set of tags allowed and of which tags are allowed where. It also specifies the attributes that each tag can have, or must have. It is now maintained by OASIS, the Organization for the Advancement of Structured Information Standards. New versions of the vocabulary are released from time to time, steered by the DocBook technical committee at OASIS. The committee has strict compatibility rules for changing the DocBook vocabulary of tags, and new releases of the official definition are numbered. The latest version at the time of writing is 4.2: “DocBook XML V4.2” or “DocBook SGML V4.2” depending on which mark-up type you are using.

## SGML vs XML in brief

Put simply, XML is just a trendier version of SGML. There are several differences between SGML and XML but for the purposes of authoring DocBook documents there is not a lot to choose between them. As structured document formats, each has their advantages. The reason you will choose one above the other is likely to be the tools available. The SGML tools have been around longer than XML, although there are fewer

of them. Free software tools for XML on the other hand have taken off quite rapidly. Current efforts are focused on the XML tools, and that is where the future of DocBook lies. There is even talk of recreating DocBook from scratch entirely with XML, RELAX-NG, XML name spaces, and other technologies that were not around at the time of DocBook's birth.

## Style sheets and catalogues

There are two distinct tool chains, one for DocBook SGML and the other for DocBook XML. In general, there is a stage in which a style sheet is applied to the DocBook document in order to produce the desired output type. For print format output types there is usually another step needed in order to perform typesetting.

There is a style sheet for each type of output format and because DocBook SGML uses a different style sheet language to its XML variant, there are a separate set of style sheets for each. The architect behind the freely available DocBook style sheets, as with so many DocBook things, is Norman Walsh. Most current work is going into the style sheets for the XML version, and they are hosted at SourceForge so that other people can contribute as well.

To process DocBook using a style sheet, you obviously need to specify which style sheet to use. However since the choice is not limited to the ones provided by Norman Walsh, and you could create your own if you wished, there needs to be a way of naming the one you want to use. This is not done by just using the file name of the style sheet; provision is made for using style sheets available on the Internet but not installed locally. SGML and XML have different solutions to this naming problem but both of them involve catalogues, simple centralized databases for converting a style sheet name into the the location of the actual style sheet itself.

It is worth noting at this point that it is not just style sheets that catalogues can track, but also SGML/XML files, character definitions, document type definitions, and so on. They are different solutions to the larger problem of how to reference external “things” from SGML or XML files in such a way that the files referencing them can be moved between different systems and still have the references match up.

SGML Open Catalogs have been around in their present form since 1997. The OASIS technical resolution (TR 9401) specifies what they look like and how they work. The naming convention used is that of Formal Public Identifiers, which look like this:

```
-//Norman Walsh//DOCUMENT DocBook Print Stylesheet//EN
```

It looks confusing, but it breaks down like this:

- `-`

  A dash here, instead of a plus sign, just means that the organization owning this public identifier has not been registered (with ISO I think). Very often the organization is not registered.

- `Norman Walsh`

  This is the ownership identifier, or the organization responsible for the entity being described (in this case the style sheet).

- `DOCUMENT`

  This describes the type of object. This can be “DTD” for a document type definition, for instance.

- `DocBook Print Stylesheet`

  This part is a description of the object.

- `EN`

  This part denotes the language that the object is written in. “EN” is for English.

The SGML Open Catalog system is a simple text-based way of mapping Formal Public Identifiers to the locations of the objects they describe, and the program for processing DSSSL style sheets knows how to use it. Normally it will start reading the catalogue from the file `/etc/sgml/catalog`; this Open Catalog file can contain catalogue entries but usually contains a list of sub-catalogues that should be inspected. The DocBook SGML distribution contains an Open Catalog listing the objects that it provides, such as the document type definition itself.

XML now has a similar mechanism for referencing document objects: the XML Catalogs specification was formulated by OASIS in 2001. It is largely similar to SGML Open Catalogs, but differs in several ways. The most important difference is the naming convention. Formal Public Identifiers are not used; instead the more web-friendly Universal Resource Indicator (URI) naming scheme is the chosen format. To all intents and purposes, a URI is the same as a URL (Universal Resource Locator), but the URIs used in XML Catalog entity resolution must be persistent. There is no point in giving a name to something for the purpose of making it universally apparent which object is meant, only to change the name at a later date.

The URI equivalent of the Formal Public Identifier shown above, for the DocBook XML version of the style sheet, is:

`http://docbook.sourceforge.net/release/xsl/current/fo/docbook.xsl`

The different parts of this are interpreted in the same way as for any URL. If you copy it into the location bar of your favourite web browser you will find yourself looking at the style sheet. However, even if it were not a working URL it would still be a perfectly good object name—it is just a unique identifier after all.

The XML Catalog is written in XML rather than the simple line-based format of an SGML Open Catalog. Its purpose is to convert one URI (the name of the object) into another (its real location). Although the `docbook.sourceforge.net` URI given above already works and gives the right result, in general it is a bad idea to fetch objects from remote locations as a matter of course. For one thing it can be very slow, as during the application of a style sheet there may be a huge number of XML Catalog look-ups. If each of these relied on Internet access, the speed at which you could process DocBook documents would be dictated by your Internet bandwidth. It would also prevent you from being able to process DocBook on a disconnected machine altogether.

Security is another reason for XML Catalogs to translate remote URIs into

local file references. Style sheet languages are quite powerful and the Internet is not altogether trustworthy, so fetching and applying style sheets unchecked from a remote machine is almost as bad as fetching executable code and hoping it does not contain a virus.

The program for applying a DocBook XML style sheet, `xsltproc` (see later), understands how to examine the XML Catalog to find objects that it needs, and can be told not to fetch remote URIs. There is also a command called `xmlcatalog` which can be used to interactively examine the XML Catalog.

Unfortunately XML Catalogs are not available on every system. Not even every version of Linux provides XML Catalog entries for the XML objects on the local system, although of course Red Hat Linux has provided them for several releases now.

## DocBook SGML tools

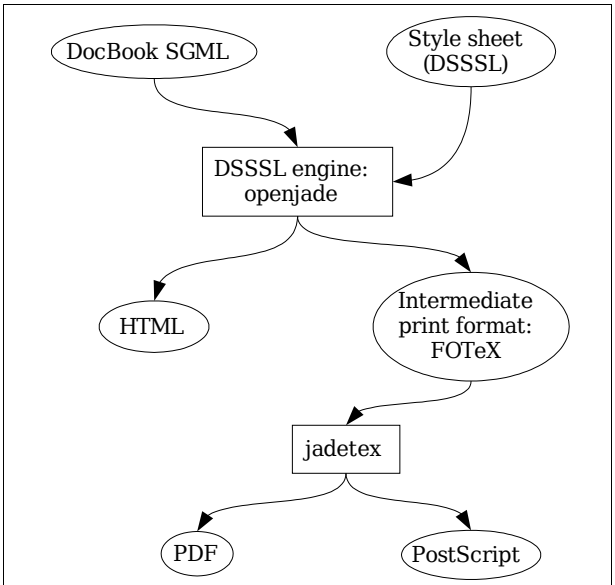

Illustration 1DocBook SGML tool chain

For SGML the style sheet language is DSSSL (deep breath: Document Style Semantics and Specification Language). DSSSL has two main purposes—transformation and layout. By transformation I mean re-arranging the document's structure. For example it can generate tables of index or contents. As for laying out the document's contents on the page, it specifies the fonts to use, how emphasized words appear, margins, spacing, justification and so on. The style sheet language itself looks a little like LISP, with lots of parentheses and indentations.

The style sheet is applied to the DocBook SGML using a DSSSL engine such as James Clark's `jade`, now named `openjade`. For on-line formats like HTML and manual pages, this single step is all that is needed, but for print formats there is more to do. The `openjade` DSSSL engine creates an intermediate output format called FOTeX which is then processed by `jadetex` to get the final PDF or PostScript. It may need to be run several times in a row in order to link up all the cross-references for tables of index and contents. FOTeX, like LaTeX, is a set of TeX macros for typesetting structured documents.

On Red Hat Linux the `docbook-utils` package contains some conversion utilities which know about the steps needed to convert from DocBook SGML to other formats, and which will re-run `jadetex` if necessary. Among the utilities included are `docbook2html`, `docbook2pdf`, `docbook2ps`, and `docbook2man`. In fact this last tool for creating on-line manual pages from DocBook SGML is a bit of a cheat: it uses a Perl script to do the

conversion, and avoids DSSSL.

## DocBook XML tools

For XML the style sheet language is XSL (eXtensible Stylesheet Language). In the spirit of XML, XSL itself is written in XML. (I would mention at this point that DocBook SGML documents can be processed using XSL style sheets, and DocBook XML documents can be processed using DSSSL style sheets; however, there is seldom a reason to do this.)

There are, again, two parts to the language: transformation of the document structure, known as XSL-T (XSL transformations); and page layout, known as XSL-FO (XSL formatting objects).

This is not the whole story with XSL. Another important part of the language is XPath. Using this powerful expression language you can specify parts of the document you want to manipulate with XSL-T.

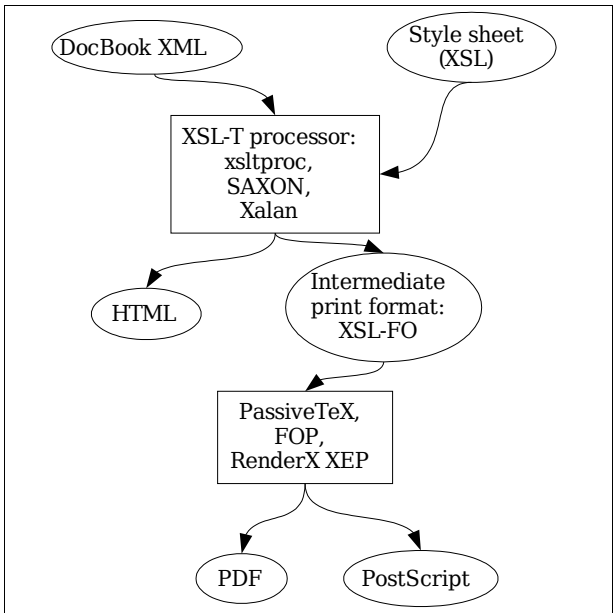

Illustration 2DocBook XML tool chain

For on-line formats a formatting language such as XSL-FO is not really needed. HTML already has structural mark-up and the web browser performs the job of laying out the content for the reader. For print formats such as PDF, XSL-FO needs to be converted. You can think of XSL-FO as a sort of presentation-oriented version of DocBook. The logical structure of a DocBook document is changed into a representation of what it should look like to the reader. A further processing step is needed to convert this representation into something like PDF.

The tools for the XSL-T stage of processing include Daniel Veillard's `xsltproc` (from `libxslt`), Michael Kay's SAXON, and the Apache Group's Xalan project. The one packaged with Red Hat Linux is `xsltproc`, and it is a fairly complete and efficient implementation.

For the XSL-FO-into-print step, there are again several different choices. FOP (Formatting Objects Processor) is the Java offering from the Apache Group—Xalan also has a Java version. The processor packaged with Red Hat Linux is Sebastian Rahtz's PassiveTeX. This program, built on David Carlisle's work on an XML parser written in TeX macros, can convert directly from XSL-FO into PDF.

There are commercial offerings in the area of XSL-FO processing too of course. There is still progress to be made with free software in this area, and there are a lot of approaches being tried. One possible future contender is the `xmlroff` project from Sun Microsystems which, unlike the others, is written in C rather than Java or TeX.

To make the DocBook XML tool chain easier there is a package called `xmlto`, and this is the equivalent to `docbook-utils` for DocBook SGML. This program is not limited to DocBook in theory, although in practice it only currently supports DocBook XML and XSL-FO as input formats. The idea is that you tell it what your desired destination format is, and it figures out which style sheet should be applied and what extra processing steps are needed.

Although it has always been a design goal of `xmlto` to use whichever tools are available to do the job, it only knows about `xsltproc` and PassiveTeX at the moment. Given that, it is extensible, and can be taught about other output formats and (XML-based) input formats simply by putting script files in the right places. When it is executed, `xmlto` first decides what type of XML file it is dealing with (DocBook, XSL-FO, or something else), and then looks for a style sheet for converting to the output format it was told to create. Once the style sheet has been applied, the post-processing script for that input-to-output conversion is run: this is just a shell script.

## Authoring

Missing from the tool chain diagram is any mention of authoring tools. The technology is still evolving in this area unfortunately, but at least there are some choices to consider.

I use the Emacs text editor, with a package called `psgml`. (If you are averse to Emacs altogether, skip the next few paragraphs!) Although it does not offer much help with setting up a DocBook file, once the basic bits are in place it will show you the tags that are available at the current cursor position (C-c C-t), can automatically close the current tag (C-c C-/), and will indent to the current nesting level when you press TAB.

It can take a while for `psgml` to parse the document type definition for DocBook (the specification of which tags are available and how they should be used), but it is possible to save a parsed document type in a format that is much quicker to load. All you need to do is create a file called `ECAT` in an `sgml` subdirectory of your home directory (i.e. `~/sgml/ECAT`). Also create a subdirectory called `~/sgml/cdtd`. Then in the `ECAT` file put the line:

```
public "-//OASIS//DTD DocBook XML V4.2//EN" cdtd/docbkx42.ced
```

The next time you edit a DocBook XML V4.2 document, the parsed type definition will be saved for future use.

Another useful feature of `psgml` is the ability to collapse sections of the document. It is often useful to be able to see the high-level structure of the document at a glance—the first few words of the each chapter, for instance—while you are still working on another chapter. This is exactly what `psgml` allows you to do with the M-x `sgml-fold-element` (C-c C-f C-e) and M-x `sgml-unfold-element` (C-c C-u C-e) commands.

Another useful Emacs package for DocBook-related authoring is Tony Graham's `xslide`, a development environment for XSL style sheets. This includes a handy do-nothing style sheet template, keyword highlighting and clever indentation. It also puts in the closing tag for any start tags you create. It won't make editing DocBook documents themselves easier, but if

you find yourself tweaking style sheets a lot it will certainly help.

Using the `xslide` package as a starting point Norman Walsh created a package called `docbookide`, the aim being to make it simpler to author DocBook documents as well as style sheets. It does this job very well, although it is now an abandoned project: Norman Walsh now uses nXML.

I have to admit that before I started writing this I had not heard of nXML, or indeed `docbookide`, despite the fact that I write things using DocBook quite frequently. I think I had assumed all of the other Emacs-using DocBookers used `psgml`. On learning that Norman Walsh used a package I hadn't heard of for writing documentation, I was eager to try it out to see what I was missing.

Although it does not have the element-collapsing capability of `psgml`, it does have a rather nice benefit: it will validate your document as you type, putting a polite warning in the status bar if there is a problem with it and underlining the problem area. If you move the mouse over an underlined piece of text, a tool-tip will show you what the validity problem is. Already it is a serious alternative to `psgml`, although it only became available a month ago. The `nXML` Emacs package, as with so many SGML-related bits of free software, is the work of James Clark.

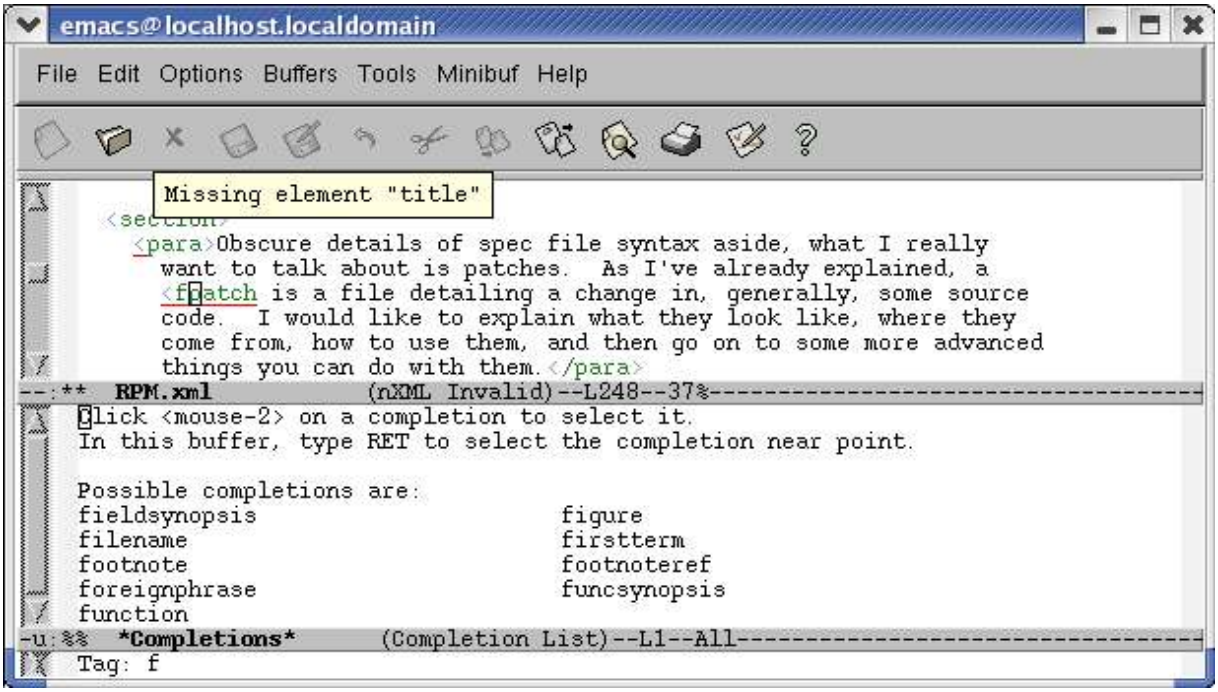

Illustration 3Emacs nxml-mode

It is possible to write DocBook using the Vim editor. Syntax highlighting is available, but there does not seem to be anything resembling the flexibility or power of `psgml` or nXML. If you cannot or will not use Emacs, you might be better off with a graphical tool such as LyX or Conglomerate.

LyX was originally written as a graphical front-end to the LaTeX typesetting system, wide-spread in academia. LaTeX is a form of structured mark-up as is DocBook, and recently LyX grew support for editing DocBook as well. It is not as complete or robust as the Emacs modes, but it seems to be a good start.

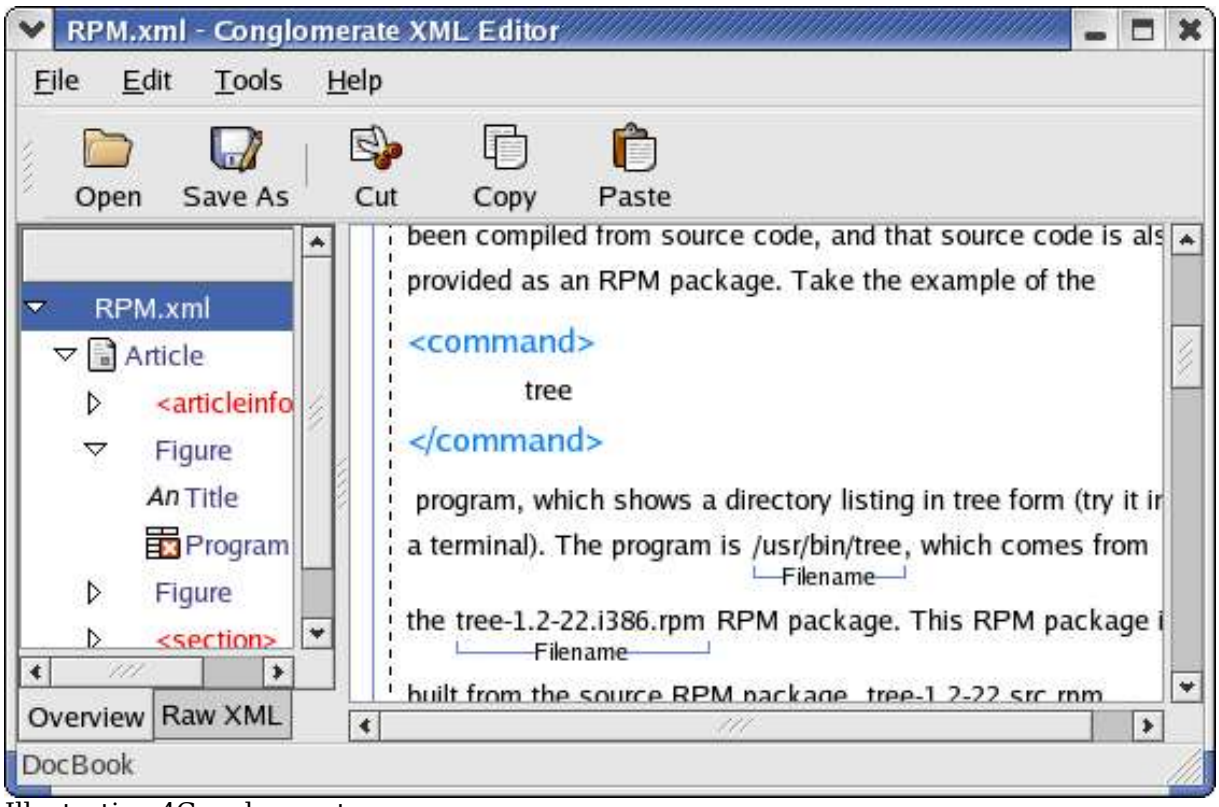

Illustration 4Conglomerate

Conglomerate aims to be a complete solution for working with documents, including editing them, storing them and publishing them. It is already quite advanced and has an extremely nice visual presentation of the structure of the document. This looks like a very promising tool indeed.

Structured mark-up ideas have secretly been filtering through into the word-processors we use all the time. Actually in some cases they have been there from the start. People use them in the way they expect them to be used, rather than taking advantage of the fact that they allow you to semantically mark up large parts of what you write. Paragraph styles can differentiate between headings, sub-headings, normal paragraphs, and so on, and character styles can be given names like “emphasis” and “function name”—and there is a move towards XML file formats in the word-processing world. OpenOffice.org 1.1 already has the beginnings of rudimentary support for DocBook by allowing XSLT document filters. I hope this will continue to become easier to use (the available filters are not quite ready yet).

## Project integration

I would suggest that most projects considering switching over to DocBook for their documentation, or starting to write their documentation from scratch, should first consider using DocBook XML. Already-written manual pages can be “lifted” into DocBook format use Eric Raymond's extremely useful `doclifter` tool.

The `xmlto` package provides some Makefile rules for processing DocBook XML in the `/usr/share/xmlto/xmlto.mak` file, which can be included by your project. If you are using `automake`, it is a good idea to add the names of your manual pages (like “`project.1`”) to the `EXTRA_DIST` variable, so that they will not need to be regenerated unless the XML has been

modified. This makes it possible for users to build the project on a system that has no DocBook processing tools installed.

## Pointers and links

- http://cyberelk.net/tim/docbook/selfdocbookx

  It is almost always easier to see a working example than to start from scratch by following someone's description of how it should work. For this reason I wrote a short article about DocBook written in DocBook itself—and the appendix includes the XML it came from. It is, to some extent, self-documenting.

- http://gphoto.sourceforge.net/

  The gPhoto program is one example of a project that uses DocBook XML as its primary format for documentation. It uses `xmlto` for generating manual pages.

- http://www.docbook.org/

  This is a good starting point for learning about DocBook and the surrounding tools.

- http://www.menteith.com/xslide/

  This is the home of the Emacs XSL editing package that Norman Walsh based his own DocBook-editing variant on.

- http://www.xmlhack.com/read.php?item=2061

  Here is an announcement of the nXML package, including some more information and screenshots.

- http://www.conglomerate.org/

  The Conglomerate structured document management system.