Using the OSDoc Documentation System[†]

Version 0.1

Jonathan Shapiro, Ph.D.
Systems Research Laboratory
Dept. of Computer Science
Johns Hopkins University

Abstract

OSDoc is an XML-based documentation system that can be used to generate documents in both online and paper form from the same document source. It includes a DTD for the documents, and XSLT transformers that can be used to generate HTML and PDF output. It is designed to support traditional (linear) documents that must also be made available in online form.

In contrast to DocBook, OSDoc is designed to be familiar to existing authors of HTML, and to users who work in text-based editors rather than XML authoring environments (which have been very slow to appear). In addition, OSDoc also provides some degree of direct control over output formatting. We agree with the SGML/XML community that presentation-based formatting is a thing to be done lightly, but our view is that presentation in documents is a vitally important element of document quality. The firm separation of content and presentation in the HTML world has in practice relegated presentation to a poorly supported second cousin.

OSDoc is a work in progress, and has some known shortcomings. For the most part these are relatively minor and the ones we know about are identified below.

1 Introduction

When we started building the documentation for the *EROS* system in 1991, we attempted to use HTML exclusively. Our hope was that by using a standardized document format we would arrive at something relatively maintainable. We anticipated that decent printing and formatting support would eventually emerge (we are still waiting), and of course, we wanted to be part of the early web. HTML has one fundamental advantage: developers already know it.

As a documentation method for large systems, HTML has several shortcomings:

- Semantically, it isn't rich enough to let the resulting documents be manipulated. For example, there is no way to say "this is a document title" as distinct from "this is emphasized text that should be wrapped by a link."

  The simplicity of HTML was critical to its success, but the inability to refi ne and customize it for particular applications is a signifi cant limitation for large scale use that is only partially addressed by the `span` and `div` tags.

- Generating documents automatically is challenging. For example, there is no easy way to build a unifi ed glossary database that can be referenced by other documents. This is something we haven't done in OSDoc yet either, but it is coming.

- There still isn't good support for printing.

- It is diffi cult to aggregate HTML documents in the context of a larger website. Often, successful organization requires some degree of processing and content extraction that HTML wasn't designed to support.

Why OSDoc?

Q: *Why not just use the docbook DTD and the associated processing packages?*

Initially we *did* intend to use DocBook. As chance would have it, the origins of the DocBook DTD trace back to the Davenport Group, a working group formed by Marcia Alan at HaL Computer Systems. Shapiro was one of the HaL founders, and we'ld certainly like to use the work that Marcia's group started and Norm Walsh has steadfastly continued.

Unfortunately, the DocBook DTDs have several shortcomings:

- A religious commitment to avoiding presentation (shared, we might add, but the entire XML-based documentation community). Some things just cannot be handled this way — notably tables.
- A large number of hard to type tags that (a) deviate unnecessarily from HTML, and (b) provide insuffi cient usage guidance to use them well.

DocBook inherited from the SGML world the assumption that most XML/SGML documents would be written using an XML-aware WYSIWYG document system. In consequence, DocBook uses very long tag names that no reasonable human being would choose to type. In our experience, nearly all XML/SGML input is actually authored using plain text editors. We wanted something closer to HTML.

The main issue is that DocBook was specialized to a set of problems that didn't really match our needs. We needed different inline tags for different kinds of annotations, and given the tag verbosity problem it was simply easier to roll our own XSLT transformations than to customize DocBook.

Finally, it became known to us that the XSL-FO standard screwed up lists so completely that no conforming implementation is useful for non-trivial documents. At that point we decided to go through LaTeX, and we were committed to writing our own transforms anyway.

Why LaTeX? Why not XSL-FO?

There are two reasons that we have chosen *not* to use XSL-FO:

1. We are committed to open-source solutions. While FOP is steadily improving, it has some very strange bugs and it is *very* incomplete.
2. The XSL-FO standard is fatally flawed. The specifi cation for list structures does not support adequate control over vertical alignment, and the specifi cation for indent inheritance within lists is completely broken. This means that a conforming implementation of the XSL-FO standard is pretty much useless for any sort of interesting document formatting.

Rather than fi ght city hall, we decided to go down the LaTeX path. Venerable, but it still does a better job than anything else we know about.

Eventually it became apparent that HTML wasn't the right tool for the job. Unfortunately, there wasn't much in the way of alternatives at the time. Today, XML exists to do the job.

1.1 What's In OSDoc?

The OSDoc package consists of several parts:

- The OSDoc DTD, which defines the structure of several OSDoc documents.
- An XSLT script that transforms OSDoc input into HTML. This script can optionally incorporate header and menu regions for use in a web site, as you can see at the *Coyotos* web site.
- A second XSLT script that transforms OSDoc input into LaTeX input. This can be run through `latex` and `ps2pdf` to provide PDF output.

  It is a nuisance to have to go through yet another tool to produce PDF. See the sidebar for further discussion of this.
- Makefile fragments for taking `xfig` drawings and converting them to encapsulated postscript and PNG files.
- An ancillary XSLT script for processing collections of documents written using OSDoc to extract synopsis information and fabricate a web page that is similar in feel to a table of contents. If you are reading the HTML version of this document, you can see an example of such a web page here.

The HTML output works fairly hard to be flexibly formatted using cascading style sheets. The LaTeX output attempts to provide a reasonably faithful rendering on paper of the same document.

The HTML output can be generated in chunked or continuous form. The web has a firmly established bad habit of chunking things in tiny ways. This makes them difficult to browse and read! Remember that OSDoc is designed to manage documents that might reasonably appear on paper. A linearized presentation of units up to the chapter level is entirely appropriate. The chunking mechanism *does* break books into chapter-sized chunks.

1.2 Relationship to HTML

OSDoc deliberately attempts to be similar to HTML tags wherever possible. HTML is widely used, and there is no compelling reason to invent new tag names for things like bullet lists, paragraphs, and so forth. OSDoc deviates from HTML when there is a significant difference between the HTML behavior and the OSDoc behavior or when HTML simply doesn't have a way to express what we wanted.

2 A Quick Introduction

OSDoc provides two main document types: books and articles. In this section, I'll try to do a lightning-quick pass over the major items in the OSDoc DTD.

Every document begins with either a **book** or an **article** tag. The first sub-element of every document is the **docinfo** section, which provides information about the document, including its title and subtitle, publication information, copyright information, and a brief synopsis.

In contrast to HTML, OSDoc sections are properly nested. A **sect2** section must reside within a **sect1** section. The DTD provides for section markers down to **sect4**. The **sect4** marker is typically rendered as a named paragraph.

Once you get into the section content, the structure of OSDoc documents can be broken into block elements and character elements. Block elements include paragraphs, subsections, figures, tables, lists, and listings. Character elements include things like emphasis and boldface tags or other attributes. Some block elements can contain other block elements hierarchically — subsections are an example of this. Similarly, character elements can often contain other character elements (e.g. *a **bold** word within emphasized text*). With the sole exception of footnote elements, character elements do not contain block elements.

The basic structure of a section is a **title**, some number of leading vertical elements (usually paragraphs), and then some number of nested subsections (if any).

2.1 Document Scaffolding

As with HTML documents, there is some amount of "scaffolding" for an OSDoc document. Depending on whether the document is a book or an article, it can have one of two structures. The overarching structure of a book is shown below. Items followed by a question mark are optional. Items followed by a parenthesis may be appear zero or more times. The marker "*...content...*" indicates that normal text content appears at that point. The structure of normal text content is discussed below. A book can have front matter, some number of chapters, some number of parts consisting of a part introduction and chapters, several appendices and a bibliography. The **toc** tag, if present, indicates that a table of contents should be inserted at that point.

```
book
  docinfo
    ...see below
  abstract?
    ...paragraphs...
  toc?
  (dedication|copyrightpg)*
  forward*
    title?
    ...content...
  preface*
    title?
    ...content...
  chapter*
    title
    subtitle?
    ...content...
  part*
    title
    subtitle?
    [partintro]
    chapter*
    appendix*
  appendix*
    title
    subtitle?
    ...content...
  bibliography?

dedication
  ...content...
copyrightpg
  ...content...
```

The outline of an article is much simpler:

```
article
  docinfo
    ...
  abstract
    ...paragraphs...
  toc?
  ...content...
  appendix*
    title
    subtitle?
    ...content...
  bibliography?
```

The article hierarchy is designed to fi t into the book hierarchy as a "drop-in" alternative to the chapter element. This allows collections of articles to be assembled into books easily.[1]

2.2 The docinfo Element

The meta-information for a document goes in the **docinfo** element. The idea is that this element should supply all of the necessary information to describe the document, including a short synopsis. There are tools that know how to scan the **docinfo** element in order to build tables of contents for online viewing.

Some of the information in the **docinfo** section is bibliographic. Eventually, we plan to extend the **docinfo** element to optionally contain a full copy of the bibliographic information for the current document. A radical idea: the ability to cite a document accurately

```
docinfo
  title
  subtitle?
  (editor|author|authorgroup)?
  edition?
  volumenum?
  pubdate?
  date?
  copyright*
    year+
    holder+
    copyterms?
  legalnotice?
    ...paragraphs...
  synopsis?
    ...paragraphs...

author
  (honorific|firstname|othername|surname|degree)+
  authorblurb?
    ...paragraphs...

authorgroup
  author+,
  affiliation?
    orgname?
    address?
    email?
```

For most users, the commonly used sub-elements of **docinfo** will be title, author, and perhaps copyright and authorgroup. The **authorgroup** tag provides a mechanism to gather several authors together and say that they have the same organizational affi liation. There is no way to specify an affi liation without forming an **authorgroup**, but it is perfectly okay to have an authorgroup with only one author.[2]

[1] One of the minor irritations with DocBook is that the article and book outlines *aren't* designed for this!
[2] This portion of the OSDoc DTD is currently in flux.

Within a copyright, wrap each copyright year in a **year** tag and each holder in a **holder** tag. The terms of redistribution, if any, should be wrapped by **copyterms**, and will appear as a separate paragraph following the copyright notice.

3 Block Elements

The most commonly used block elements are paragraphs (**p**), ordered lists (**ol**), unordered lists (**ul**), tables (**table**), fi gures (**figure**), defi nition lists (**deflist**). The occasional intrepid author may also venture into sidebars (**sidebar**). In addition to these, various admonition wrappers are available (**caution**, **warning**, **note**).

There are currently two forms of listings (**programlisting**, **grammarlisting**.[3] There is also an **example** wrapper for code examples. All of these listing-like forms honor white space and line breaks.

The **ol** and **ul** elements take an optional title. The outline of an ordered list is:

```
ol
  title?
  li+
    ...paragraphs...
```

Note that in contrast to HTML, the content within a list item *must* be paragraphs or similar block elements.

Defi nition lists have a richer structure:

```
deflist
  title?
  defli+
    label
      ...character elements...
    li
      ...paragraphs...
```

4 Character Elements

[3] I will probably collapse these into a single tag **listing** with a qualifying attribute. It's the kind of thing where people will come up with new kinds of listings that they will want to be able to say things with, and OSDoc shouldn't get in their way.