Simple Web Sites Using DocBook XML and CSS

David Lynch

Abstract

How to build simple content Web sites using DocBook XML and CSS.

The Web was originally intended to make content easily accessible. Today, Web developers focus on style and marketing, but the need to put together content-driven Web sites quickly and easily remains as valid as when Tim Berners-Lee first conceived of HTML. I have taken the approach of using primarily DocBook XML and CSS, as well as some other readily available Linux tools, that allows me to bring up simple content-focused Web sites—a poor man's content management system.

I am an embedded software developer. HTML, XML, CSS and the Web in general are peripheral to what I do. I am not as intimate with the details and idiosyncrasies of HTML as I am of processors, NICs and UARTs. Yet today, the Web is part of everything. Proof that an embedded processor is up and running under Linux often consists of being able to browse Web pages on it. I look for clients, and clients seek me out over the Web. Although expertise in JavaScript, cross-browser HTML, CSS, PHP, Ruby on Rails and so forth is not essential, a basic knowledge of HTML and the ability to use some tools to create simple but useful Web sites quickly and easily is increasingly a core skill to software development, as well as many other jobs. DocBook XML provides a means of creating documentation focused on content, with the ability to use it easily in many forms, including Web pages.

This approach has a number of elements, and they are not heavily interdependent. Even if you do not like my overall approach, you can take bits and pieces from it and incorporate them into your own approach. I am a software-tools kind of guy. There are probably numerous IDEs for Web development that will do everything for you once you know them, and there are likely a number of Eclipse plugins. Powerful, dedicated tools typically have a steep learning curve that pays off only if you do a lot of that type of work.

This article is not about DocBook XML. It is about how to build Web sites using CSS to render DocBook XML documents simply. I am not a Web developer, and I opt to learn tools that have broad uses. The tools I use for building Web content are vim for editing, m4 or Perl for macro processing and HTML tidy for verification—the same tools I use to develop software and write documentation. During the past few years, I have added basic XML, particularly DocBook XML, to my list of fundamentals.

I keep a simple DocBook XML article template readily available and pull it up in vim whenever I feel inspired to write something technical that is larger than an e-mail. By using a DocBook XML template, I can focus mostly on content and produce results that are clear and meaningful, with minimal emphasis on presentation.

More recently, I have discovered that with a little help from CSS, DocBook XML documents can be viewed directly on any Web site by CSS-capable browsers, without transforming to HTML, making it easy to add to my Web site. For more complex documents, OpenOffice.org supports DocBook XML as

an output format, and there are increasingly more tools to produce and manipulate DocBook XML. DocBook XML can be read directly by OpenOffice.org or transformed easily into all commonly used document formats, such as HTML, PDF, Word and so on. One objective of XML (one that would be difficult to identify in the competing XML word-processor formats) is divorcing content from presentation. This is a principle I heartily endorse.

I make a distinction between the parts of a Web site used for navigation and the content of the Web site. I deliberately choose to separate content physically from navigation. With rare exceptions, all content pages are devoid of navigation and function as standalone documents. Today, I do them in DocBook XML. Previously, I used HTML; however, I always tried to maintain a separation between content and navigation. My first step is to build an HTML presentation/navigation framework. I create the main HTML index page for the site, and I use HTML FRAMES to divide the display into three regions: a header, a menu and a body. FRAMES are somewhat frowned upon within Web development, as they can be used to capture other people's Web content and create the impression that it is your own. They also can impede navigation, and they may be less friendly to people with disabilities. However, I am not aware of another equally easy-to-use Web construct that can be made to separate content from navigation and presentation. There are other means to achieve similar effects, but all of those that I am aware of incorporate navigation and presentation elements into the content. My objective is to be able to develop the content of the Web site in DocBook XML, modified only to include a stylesheet and to isolate presentation and navigation elsewhere.

There is one other heretical side effect to this approach—nothing about it requires a Web server. You can build and test all of this in the browser of your choice without installing a Web server, and when finished, you can drop it all on a CD-ROM where it can be viewed on any system with a Web browser.

The core of my index page is:

```
<frameset class="frame" cols="140,*" bordercolor="#000000"
frameborder="0" framespacing="0">
 <frame class="frame" src="margin.html" name="Margin" scrolling="no"
marginwidth="0" marginheight="0"
 <frameset class="frame" rows="100,*" bordercolor="#000000"
frameborder="0" framespacing="0">
  <frame class="frame" src="header.html" name="Header" scrolling="no"
marginwidth="0" marginheight="0" />
  <frame class="frame" src="home/index.xml" name="Body" scrolling="auto"
marginwidth="0" marginheight="0" frameborder="0" />
 </frameset>
</frameset>
```

This divides the browser display into three regions. A menu area on the left, a header at the top and a body for content in most of the remainder. My header page tends to be fairly trivial, basically:

```
<body class="header" id="body-header">
 <div class="header">
  <h1 class="header">My Title</h1>
 </div>
</body>
```

The class and id tags allow the use of CSS to overload style later.

The margin is almost as simple:

```
<body class="margin" id="body-margin">
 <div class="menu-box">
  <div class="menu" id="home">
    <a href="home/index.xml" target="Body">Home</a>
  </div>
...
 </div>
</body>
```

Again, the class and id tags are for CSS style. The menu-box block element surrounds all the menu items. The menu block elements can be repeated as needed. CSS can be used to style the menu items to suit personal taste. Specifying a target for the links means that when a menu item is clicked on, it changes the document in the “Body” frame of the frameset.

I use the following CSS to create highlighted menu buttons:

```
div.menu-box {
 display: block;
 border-width: 2pt;
 border-color: color_bkgr !important;
 border-style: inset ;
}

div.menu {
 border-style: inset ;
 border-width: 5px ;
 background: color_menu_bkgr1 !important;
 border-color: color_menu_bkgr !important;
 color: color_bkgr !important ;
 font-weight: bold;
 font-size: 8pt;
 height: 14pt ;
 Width: 110pt;
 vertical-align: middle;
 x-margin: 5pt;
 x-padding: 5pt;
 text-align: center;
 padding-left: 5pt;
}

div.menu:hover {
 position: relative;
 top: 1px;
 left: 1px;
 border-color: color_menu_bkgr1;
 background-color: color_menu_bkgr;
}

a.menu { text-decoration: none }
```

Those are all the key elements of the non-content portion.

The menu system can be nested. Changing the target of a menu item to “Margin” can pull in a new side menu, and that can be repeated as often as you like. Internet Explorer's handling of CSS, particularly positioning, is broken, so there are subtle differences in the display between it and properly conforming browsers. Complicated cross-browser CSS positioning can be extremely difficult, and it is further complicated because Internet Explorer 7 is slated to fix many CSS issues in ways that break most of the

published work-arounds for earlier versions. Also, I would advise being careful about background colors. I spent a short life time failing to figure out how to eliminate a white streak between the menu area and the body that appeared only with Internet Explorer and only if I used a background color. This article is not about how to become proficient at fancy cross-browser Web development; the focus is on providing a simple approach to easily display content that looks pleasant, regardless of the browser. Getting pixel-for-pixel identical CSS cross-browser results for numerous browsers is a complex task.

Up to this point, I have ignored the HTML headers and issues, such as the fact that color_menu_bkgr is not a valid HTML/CSS color.

HTML pages, such as index,html, header.html and margin.html need valid HTML headers, and they need a link element referencing the CSS stylesheet, such as:

```
<link rel="stylesheet" type="text/css" href="/css/stylesheet.css"
title="default">
```

added to the header.

The CSS excerpt above is from stylesheet.css, which also can include any additional CSS you might want to add or overrides for the default DocBook CSS. A number of CSS stylesheets are available for DocBook XML—several are listed on the DocBook Wiki, and the particular stylesheet I use is badgers-in-foil (see the on-line Resources). The badgers-in-foil stylesheet has allowed me to render DocBook XML articles pleasingly in several different browsers.

All XML pages need two stylesheet links added to the XML header:

```
<?xml-stylesheet href="/css/docbook-css/driver.css" type="text/css"?>
<?xml-stylesheet href="/css/stylesheet.css" type="text/css"?>
```

The second link is not strictly necessary, but it can be used to override or add additional style information to the DocBook XML files, without changing the DocBook XML stylesheet.

I handle the generation of the framework, XML and HTML wrappers and many repeated elements using the macro processor m4. It could be done as easily with Perl or bash/sed. This allows me to define standard headers, colors and other useful string substitutions as m4 macros. color_bkgr is an m4 macro and will be replaced by m4 with the background color I have chosen for this site anywhere it occurs. I reuse the same framework whenever I need to create a new Web site. I can create a new site with different content, titles, colors and so on by changing a few macros. However, the complexity gradually has increased to the point where I am starting to think of moving from m4 to Perl for the preprocessing. I am using automated generation of XML and HTML, and therefore it is an excellent idea to use HTML tidy after processing to verify it.

First, install HTML tidy and m4. I primarily work with Debian and Debian derivatives, so installing tidy and m4 consists of:

```
apt-get install tidy
apt-get install m4
```

Most distributions should provide m4 and have tidy available through their package system. See Resources for the main pages for tidy and m4.

Then, I have a text file (pages.list) with a list of the base names for all pages, as well as their type: CSS, HTML and XML:

```
stylesheet,css
index,html
header,html
margin,html
home,xml
...
```

I use a short shell script to run m4 and HTML tidy on each page and place the results where they belong:

```
#!/bin/sh
# $Id:
# $URL:

#dest=../test
dest=..
lname=pages.list

dopage() {
 echo "$1"
 if [ "$2x" == "xmlx" ]; then
  if ! [ -d $dest/$1 ]; then
   mkdir $dest/$1
  fi
  m4 -D_xml $1.m4 | tidy -i -xml >$dest/$1/index.xml
 elif [ "$2x" == "htmlx" ]; then
  m4 $1.m4 | tidy -i  >$dest/$1.html
 elif [ "$2x" == "cssx" ]; then
  m4 -D_css $1.m4 >/var/www/share/css/$1.css
 else
  echo "Whoops $1 $2"
 fi
}

if [ -f $lname ]; then
 list=`cat $lname | grep -v '#' | awk '{print $1}' | tr '\n' ' '`
 for argv in $list ; do
  page=""; fmt=""
  page=`echo $argv | awk -F "," '{print $1}'`
  fmt=`echo $argv | awk -F "," '{print $2}'`
  dopage ${page} ${fmt}
 done
fi
```

Now, m4 can handle the generation of standard headers, links to stylesheets, macro substitutions, substitutions for color names and so forth. The menu items even can be generated automatically from macro items.

The header.m4 file to generate the header page becomes:

```
define(_page,header)dnl
include(defs.m4)dnl
include(hdr.m4)dnl
<div class="header">
 <h1 class="header">_title</h1>
</div>
```

```
include(ftr.m4)dnl
```

A Web server is not needed to view any of the framework and content we have created, but most Web pages are distributed by a Web server. No additional configuration should be needed for most Web servers; however, the following CSS config file added to /etc/apache2/conf.d creates an alias, allowing the CSS directory to be shared across multiple sites or to be referenced easily regardless of the relative path inside the Web site:

```
Alias /css /var/www/share/css/

<Location /css>
 Order allow,deny
 Allow from all
 Options Indexes FollowSymLinks MultiViews
</Location>
```

This is a software-tools approach. For a small number of Web sites with very little content, there is no benefit to adding the complexity of automating the generation of HTML or XML headers and footers. Where there is a significant amount of content, frequent modification or numerous unique sites, there can be a substantial benefit.

I have barely touched on DocBook XML. I started “word processing” in college using text formatters like runoff, nroff and text on my H8. The concept of separating content from appearance is a natural return to my non-WYSIWYG word-processing roots.

There are tools available to do WYSIWYG processing of XML documents. The easiest approach, if you are more comfortable with a WYSIWYG word processor, is to use OpenOffice.org, which can save documents as DocBook XML. OpenOffice.org's DocBook XML capabilities are limited, however. It is not typically possible to go from a well-formatted OpenOffice.org format or Word format file to a DocBook XML document without losing some facets of the presentation. Plain DocBook XML is more focused on content and structure than presentation details. OpenOffice.org does not associate a stylesheet with the saved DocBook XML document, so style items, such as typefaces, type size, indents and so on, will be supplied by the DocBook XML CSS you use. If you are not completely happy, you either can modify the stylesheet or override it by “cascading” a new stylesheet, changing the elements you want to change.

As I mentioned previously, I am happy with the badgers-in-foil stylesheet. My CSS makes very few changes. I am more focused on creating readable documents easily and getting them to my Web site or transforming them into other file formats as needed. As I mentioned, I usually choose to start with a simple DocBook XML article template. I use vim to add my content to that template. The template uses a bare minimum of DocBook XML, and aside from some XML fundamentals, such as making certain that start and end tags remain matched, my paragraphs use little more than a few very obvious tags.

Proficient DocBook XML users can master a rich set of DocBook XML constructs, but ordinary users can easily produce increasingly sophisticated documents by slowly learning only a few tags. I find DocBook XML significantly easier to use than HTML. XML is rigid in tag matching and nesting rules, and there are less, if any, idiosyncrasies. Structure and organization—lists, tables, paragraphs, chapters, sections and so on—are all done in DocBook XML. Appearance and presentation decisions are made in

the stylesheets. Capable CSS developers could transform a basic DocBook XML article into something elegant. However, my objective is not elegant documents and Web sites, but in making content informative and readable in a variety of formats quickly and simply.

DocBook XML is an increasingly popular approach to constructing Web documents. Numerous open-source projects, as well as the Linux kernel, are relying more heavily on DocBook XML as a standard format for documentation. The Linux Documentation Project provides an author's guide with the sample article template I frequently use, as well as a large number of links to other DocBook XML resources. Eric Raymond's “DocBook Demystification HOWTO” provides an excellent explanation of why DocBook XML is important and why it is replacing most other formats for open-source documentation. Michael Smith's “Take My Advice: Don't Learn XML” is similar and explains why making worthwhile use of DocBook XML does not have to involve becoming an expert in XML or the plethora of associated XML technologies. *The Definitive Guide* by Norman Walsh and Leonard Muellner will provide you with much more than you likely will need to know, as well as critical answers if your use of DocBook XML starts to become more sophisticated. And finally, I hope this article makes clear that making effective use of DocBook XML can be simple and requires developing minimal new skills.

Resources for this article: http://www.linuxjournal.com/article/9263.