Authoring RFPK Software Documents

Docbook XML is the preferred format for RFPK Software documents, for reasons of efficiency, standardization, maintainability and ease of publishing. This tutorial explains how to get set up to use Docbook, how to to create new documents, how to edit documents with Emacs, how to make diagrams with Dia, how to check spelling, how to publish completed documents to the Software Team web page, how to archive document versions in CVS, and how to interface with Microsoft Word.

Table of Contents

Introduction

RFPK software documents fall into two categories:

- *Documents generated by development tools.* Prime examples are the documents created by Doxygen and JavaDoc.
- *Documents authored by Software Team members.* Such documents may include material prepared with the help of tools, such as Dia, the diagramming tool, but they are not generated automatically.

This *howto* is concerned exclusively with the latter case.

As a rule, all authored software documents should be written in one or another of the following XML markup languages:

- Docbook
- Xhtml

Docbook is designed for technical articles and books. The language contains a large collection of elements useful in technical documents. Elements are defined in a Document Type Definition (DTD), which itself is an XML document. The DTD makes it possible to validate a document, insuring that only valid elements are used and that the correct hierarchy of elements is observed. From a valid Docbook document, output in a number of formats can be automatically generated, include *html*, *pdf* and *rtf*.

Xhtml is used to create Web documents that are neither books nor articles. Xhtml is very similar to Html, for which it is a replacement. Over the years, the definition of Html has become murky, as various browser vendors have added their own proprietary features. Xhtml, by contrast, is defined by a DTD. Unlike a Docbook document, an Xhtml document does not have to be processed before it is placed on the web.

The existence of DTDs for both Docbook and Xhtml make it possible to configure Emacs as a powerful language-specific XML editor. A Docbook or Xhtml document contains in its preamble a reference to its DTD file. When Emacs opens a document, it first reads and parses the DTD. From that point onward, Emacs knows which elements are valid at each point in the document. It can provide you with a list of valid elements, automatically insert markup tags, nicely indent elements and content, and perform many other XML-specific and DTD-specific services.

Authored RFPK software documents fall into several categories:

- *Howto.* The current document is a typical example. A howto is a tutorial which explains some practical method or process.
- *White Paper.* A white paper is typically more general and high-level than a howto. A white paper can explain the theory behind a process or group of processes. It can present the rationale for an important decision. It can suggest or define a course of action.
- *Decision Research.* Developing software involves making many decisions where one alternative is chosen from among several possibilities. A decision research document summarizes supporting evidence for a development decision, including prototyping, benchmarking, and summarizing best practices or opinions of respected experts.

To improve the efficiency of creating and publishing authored software documents and to foster standardization, a set of processes have been developed for each of the document types listed above. Specific details of these processes will be described in this howto.

RPFK software documents are primarily published to the Software Team web page. The page of links to these documents is written in Xhtml. Each time a document is published for the first time, it is necessary to edit the Xhtml file. The process for doing this is also described, below.

Sometimes, the whole or parts of an RFPK software document must be included in a Microsoft Office document. Grant proposals are a good example. This howto also describes the process for making a software document accessible to Microsoft Office.

All RFPK software documents are under CVS version control. The process for maintaining this control is also described in this document.

In the sections that follow, we will demonstrate the process of authoring a document, from initial creation, to publishing on the Software Team web site. A hypothetical example is used, which is a howto about the way the Software Team uses the Perl language for processing XML. Our example could have been a white paper or a decision research paper just as well. The process would be the same in each case.

Configuring Your Workstation

Your workstation will need a bit of configuration in order to use the tools and framework described in this paper.

Prerequisites

It is assumed that the operating system for your workstation is Redhat Linux, version 8.0. You must have the following packages installed to process Docbook texts:

```
docbook-style-dsssl
docbook-dtds
docbook-utils-pdf
```

docbook-style-xsl
docbook-utils

These packages should already be installed in your workstation. To determine whether or not they are, use the command **rpm -qa | grep -i docbook**, in order to get a list of all installed packages that have docbook in their names. If any are missing, you should download them from the RedHat Network [1] web site.

Emacs Configuration

When Emacs starts up, it reads configuration information from a file, named `.emacs` or `_emacs`, that it finds in the user's home directory. If you have not already done so, you can download the configuration information that you need from whitechuck, and concatenate it to your current `.emacs` file.

As an ordinary user:

```
cd
scp whitechuck:/opt/download/dot.emacs .
cat dot.emacs >> .emacs
rm dot.emacs
```

XHMTL Configuration

In order to use Emacs to edit Xhtml files, you will need to install a package and one additional file.

Starting as an ordinary user and then switching to the super user:

```
cd
scp 'whitechuck:/opt/download/xhtml1*.rpm' .
scp whitechuck:/opt/download/xhtml1.soc .
su
rpm -Uhv xhtml1*.rpm
rm xhtml1*.rpm
mv xhtml1.soc /usr/share/sgml/xhtml1/xhtml1-20020801/DTD
```

Starting a New Document

All work on RFPK software documents should be performed in the CVS workspace on your workstation. If you do not have a CVS workspace, consult the RFPK CVS Howto [2] for details.

The CVS Workspace

Within your CVS workspace, the directory hierarchy which we are interested in looks like this:

```
r2
    doc
   CVS
   decision
   research
   howto
```

```
stylesheet
whitepaper
web
```

Various directories are dedicated to the following:

doc/decision/research

Decision research papers.

doc/howto

Howto tutorials.

doc/stylesheet

The locally modifiable stylesheet. Changes to the `local.dsl` file will affect the appearance of all RFPK software documents.

Do not modify `local.dsl`

doc/whitepaper

White papers.

web

Contains files for the Software Team web page.

Creating an Empty Document

First you need to come up with a title for your new document, and then distill that into an identifier that expresses the subject of the paper. In our example, the title will be *Perl Processing of XML* and the subject will be *perl-xml*.

Go to the directory designated for the type of document that you are about to produce and execute the **newdoc.sh** shell script to create the directory and directory contents that you need:

```
cd howto
sh newdoc.sh perl-xml
cd perl-xml
```

You will find that the new directory contains two files:

```
doc
    howto
```

```
   perl-xml
       perl-xml.xml
       Makefile
```

The file `perl-xml.xml` is the skeleton of your new document. It looks like this:

```
<?xml version="1.0"?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.2//EN"
 "/usr/share/sgml/docbook/xml-dtd-4.2-1.0-14/docbookx.dtd" [
  <!ENTITY uw "University of Washington">
  <!ENTITY dept "Department of Bioengineering">
]>
<article><title>perl-xml</title>
 <articleinfo>
    <revhistory>
      <revision>
 <revnumber>1.0</revnumber>
 <date>July 4, 1776</date>
 <authorinitials>???</authorinitials>
 <revremark>Initial version.</revremark>
      </revision>
    </revhistory>
  <abstract>
      <para>
      </para>
  </abstract>
 </articleinfo>
  <sect1>
    <title></title>
    <para>
    </para>
  </sect1>
</article>
```

First Modifications

When you open this file with Emacs, a status message should appear in the mini-buffer at the bottom of the frame, telling you that Emacs is parsing the DTD. You should then see the following menus added to the menu bar at the top of the frame: **SGML**, **Modify**, **Move**, **Markup**, **View** and **DTD**.

The first thing you will want to do is edit a few lines which have been given default values in the starting template.

Replace

```
<article><title>perl-xml</title>
```

with the full title

```
<article><title>Perl Processing of XML</title>
```

Modify

```
<date>July 4, 1776</date>
```

to show your expected date of publication.

Modify

```
<authorinitials>???</authorinitials>
```

to show your own initials.

Provide the section-one title

```
<title></title>
```

with a non-empty title.

Viewing Your First Steps

Now you should verify that everything is configured properly, and that your initial changes were successful.

1. Process the file

   `Tools => Compile`

   You will see in the mini-buffer the message

   `Compile command: make -k`

   Press the **Enter** key and the Docbook infrastructure will produce three output files: `perl-xml.html`, `perl-xml.pdf`, `perl-xml.rtf`.

2. Open the newly created `perl-xml.html` file with your browser. (You will need to enter the absolute path name in the URL window at the top of your browser window.)

   While writing in Docbook, it is very convenient to leave the browser pointed to your document. You should reprocess your file often, to make sure that you have not made changes that violate the Docbook structure. If you alter the command in the minibuffer to be **make html**, you will only produce a .html file, which will save time. Then you can click the **Reload** button in the browser to view your modified output.

3. You can also view your .pdf output with Acroread and your .rtf output with Open Office. It will slow down your document development if you check all formats every time you make a few additions. Checking .html with a browser is the most efficient way to work.

Building Your Document

Now that you have the skeleton of a document, the real fun begins. The best way to proceed is iteratively. Add a few paragraphs or other elements to your paper, process it, and view it with the browser. Repeat, until done.

Sources of Information about Docbook

A Docbook article or book is constructed as a hierarchy of elements. The best source of information about these elements is the book Docbook the Definitive Guide [3], by Norman Walsh and Leonard Muellner.

Docbook contains a lot of elements. It is unlikely that you will need more than a few dozen of these. A good way to learn what you actually do need is to look at the XML source of existing RFPK software documents. You will find copies of all of them in your CVS workspace.

Using Emacs

You will find Emacs to be a very powerful tool for building XML documents. Use the **Markup** menu to insert elements. Emacs will only let you insert elements that are valid at the point that you are editing. That way your document will stay valid as it grows. Having Emacs generate tag pairs really speeds things up. If you already have a good idea of what element you want, you can use the (C-c C-e) keystroke sequence for even greater efficiency. Just type a few leading characters, hit **Enter**, and Emacs will automatically complete the tag name and insert the tag pair.

Checking Spelling, with Emacs and Ispell

Emacs is interfaced to the Unix Ispell utility to provide spell checking for your document. The following menu sequence will initiate spell checking of your entire document.

```
Tools => Spell Checking => Spell Check Buffer
```

Emacs will high-light the first word-like token in your text that Ispell is not sure of. A window presenting choices will open at the top of the screen, and the mini-buffer at the bottom of the screen will contain help information.

Often the high-lighted token is not mis-spelled. It is simply not in Ispell's dictionary. Here are some of your choices, when this occurs:

Digit

Enter the digit, digits, or other symbol which corresponds to one of the choices at the top of the frame. The corresponding word will replace the high-lighted word.

Space

Accept the word this time. Each time the word appears in the remainder of your paper, it will be high-lighted again.

a

Accept the word for this entire editing session.

A

Accept this word, and add it to a special dictionary concatenated to the end of the document. It will be considered correct for the life of the document.

i

Accept this word, and add it to your personal dictionary.

x

Exit from the spell checker. Return to the originally editing point in your document.

There are a lot of additional commands. Check them out in the **Spell Checking** menu.

XML General Entities

General entities are both a necessary and a useful part of XML. They can be thought of as a kind of simple replacement macro. You place the markup for the entity in your document and it is replaced with other text and markup.

Predefined Entities

The following five symbols have special meaning in XML, because they are part of the markup syntax:

```
& ' > < "
```

There are five predefined entities, corresponding to these symbols. When you place the markup for one of these symbols in your document, you avoid confusing the XML parser, and the symbol itself appears in your output.

&

Ampersand (&)

'

Single quotation mark; apostrophe (’)

>

Greater-than sign (>)

<

Less-than sign (&)

"

Double quotation mark (")

You must always use < and the &. The others are optional in most contexts. Just keep in mind, that if a parser complains about one of these symbols, you will have to replace it with its entity markup.

User Defined Entities

You can define your own entities. Consider the preamble to the template that you used to start your document:

```
<?xml version="1.0"?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.2//EN"
 "/usr/share/sgml/docbook/xml-dtd-4.2-1.0-14/docbookx.dtd" [
  <!ENTITY uw "University of Washington">
  <!ENTITY dept "Department of Bioengineering">
]>
```

The space between the two square brackets, [and], is reserved for user additions to the DTD. Inside of those brackets, two general entities are defined: *uw* and *dept*. Here is an example of markup using them,

```
RFPK, a project in the &dept; of the &uw;.
```

and here is the output generated:

```
RFPK, a project in the Department of Bioengineering of the University of Washington.
```

Using these two entities could save time and effort, if the two phrases appeared often within your document. User defined entities can be more than a convenience, however. Consider for example, the server named whitechuck. The name comes from that of a river in Western Washington. If you simply place that name in your documents and they are published to the Web, your site might draw traffic from people interested in hiking or white water kayaking. This was not your intention, and it is a waste of both resources and other other people's time. It would be better to *semantically mark* the name whitechuck to indicate that it refers to a computer rather than anything else.

The semantic markup to designate whitechuck as a computer system looks like this:

```
<systemitem class="systemname">whitechuck</systemitem>
```

Not only is this a lot to write, but you want to be sure that you refer to whitechuck the same way every time. Here is where a user defined entity can provide standardization in addition to convenience. The entity would look like this (shown in the context of the document type declaration):

```
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.2//EN"
 "/usr/share/sgml/docbook/xml-dtd-4.2-1.0-14/docbookx.dtd"
[
    <!ENTITY whitechuck '<systemitem class="systemname">whitechuck</systemitem>'>
]>
```

Note the single quotes around the replacement text, which allow double quotes to be used for the class attribute value.

Once the whitechuck entity has been defined, it is easy to include it in the document. For example, you might write the following:

```
Copy the file to &whitechuck;.  Every time a Web user references
the home page on &whitechuck;, the file will be executed. If
&whitechuck; is down, blah, blah, blah.
```

Adding Diagrams to Your Document

In principal, any sort of graphics image can be added to a Docbook document. This includes diagrams, photographs, and any sort of bitmap. It is easy for the author to do this. The hard work falls upon the infrastructure to scale the image appropriately for both the Web formats (html) and printed page formats (pdf, rtf). At present this does not always work.

With the infrastructure currently used by the Software Team, one reliable way to include diagrams in documents is to create them with Dia, an Open Source diagramming tool that bears some resemblance to Visio. Dia is available on your desktop through the menu sequence **Main** => **Office** => **Diagrams**. You can also launch it from a shell window in your document directory with the command **dia &**.

Dia is a mostly self-explanatory Gui application. A description of the interface is beyond the scope of this document. We do need to discuss the saving of files, however.

- *Saving the diagram for addition to the CVS repository.*

    1. Locate the mouse pointer anywhere within your diagram, then use the following mouse and menu sequence to save the diagram: **right-click** => **File Save As**
    2. In the window that pops up, un-check the *Compress diagram files* check box.
    3. In the *Selection* box, type in a name for this figure, and then click **OK**.

  The file will be saved as XML. Because XML is a text format, it can be readily maintained in CVS.

- *Saving output for inclusion in your document.*

    1. Locate the mouse pointer anywhere within your diagram, then use the following mouse and menu sequence to produce output for inclusion in your document: **right-click** => **File Export**
    2. In the *Determine File Type* selection list of the window that pops up, choose **Portable Network Graphics (*.png)**.
    3. In the *Selection* box, the file name should be the same one under which you saved the diagram. If not, change it so that the two names match, then click **OK**.

A good way to add a diagram to your document is to use the *figure* Docbook element. Here is an example:

```
<figure>
  <title>Three Tiered Architecture</title>
  <graphic scale="60" fileref="arch.png"/>
</figure>
```

The "scale" attribute requests the formatter to include the diagram at 60% of its original size. With your diagrams, you will need to experiment with this number, until things look right.

Publishing Your Document

To publish your document to the Software Team web site, you need to do two things:

1. Install the output files on the web server.
2. Add links to the web page.

Installing to the Web Server

Before installing to the Web Server, you should be sure that your document is complete. When you are satisfied with it, generate the output one last time, then install. In the document directory:

```
make clean
make
make install
```

You can invoke the **make** commands listed above either from the command line from inside Emacs, via the **Tools** => **Compile** menu selection.

The .html file, the .pdf file and all .png files will be copied to the appropriate directory on whitechuck by **make install**.

Adding Links to the Web Page

In the directory `r2/web` of your CVS workspace, edit the file `index.html` with emacs. Find the links to other documents of the same type as your document, clone one of them, then change the references to designate your document name.

Adding Your Document to the CVS Repository

The final step in creating a new document should be the placing of it in the CVS repository. In this example, once again, we assume that your document name is `perl-xml`.

Starting in the your document directory (`perl-xml`):

```
cd ..
```

```
cvs add perl-xml
cd perl-xml
cvs add perl-xml.xml
cvs add Makefile
cvs add *.png         # if you have an .png image files
```

At this point, everything has been added to your local workspace, but the changes have not yet been made to the repository on whitechuck. To accomplish that:

```
cd ..
cvs commit
```

Compatibility with Microsoft Office

In the directory for your document, **make install** copies the .html and .pdf output files to the Software Team web page. It does not, however, do anything with the .rtf output file. The purpose of that file is to facilitate the inclusion of your document, either whole or in part, in a Microsoft Office document.

You do not need to do anything with the .rtf file until it is needed for a Microsoft Office document. At that time you will need to transfer the .rtf output, along with any associated image files, to a workstation running Microsoft Office. You can do this in a number of ways: attaching the files to email, transferring them on a floppy disk, sending them via the **ssh**, etc. In Microsoft Office, just open the .rtf file. Office should be able to integrate your image files. You may, however, need to use Office to manipulate the scaling of the images. Finally, save the file as a Microsoft-standard .doc file.

Notes

1. http://www.redhat.com/software/rhn/
2. http://whitechuck.rfpk.washington.edu/soft/howto/cvs/cvs.html
3. http://whitechuck.rfpk.washington.edu/soft/books/html/tdg/en/html/docbook.html