NewbieDoc Docbook Guide

Jesse Goerz

jwgoerz@users.sourceforge.net

This document is intended to help new doc writers learn the basics of writing SGML documents. Once you have become familiar with this tutorial please read the document standards guide.

1. Introduction

1.1. Why SGML?

We chose to use SGML because of its inherent benefits. With SGML we can write one source document and render it to many formats.

One of the pitfalls many people fall into when writing SGML is that they try and concern themselves with formatting. Where with HTML you are often forced to manipulate tables or other elements to make the page look just the way you want, SGML is content based and cannot be manipulated this way. When writing SGML documents, formatting should be the last thing on your mind. Concentrate on writing a clear and concise document, because you can never predict the final output. The best you can do, and what is expected of all NewbieDoc authors, is to pay close attention to the formatting you are using, making sure that you always use the appropriate elements. The SGML parser will then take care of all the formatting.

If you can write HTML documents, then you can write SGML documents. In fact, HTML was designed as a small, web browser-focused version of SGML. If you haven't ever written an HTML document, you can still write SGML. For some people, it's actually harder to transfer writing skills developed for HTML than it is to learn SGML from scratch. Writing SGML requires a different philosophy than writing HTML, as explained above. Quit worrying about formatting, that will come later!

1.2. What you need to install

1.2.1. SGMLtools-2

The package you need to install is SGMLtools-2. This will give you a complete, working SGML parser. At the command line do (as root):

```
bash# apt-get install sgmltools-2
```

To use SGMLtools-2 simply do this at the command line (as a regular user):

```
bash$ sgmltools -b html name-of-sgml-file.sgml
```

You should execute the command exactly as you see above, substituting the name of your SGML file for `name-of-sgml-file.sgml`. This will create a subdirectory called `name-of-sgml-file` with your SGML file rendered as an html document inside it. (For the rest of the document I will use the term "build", "parse", or "render" to refer to running your file through the SGMLtools) As you are going through this tutorial feel free to cut and paste the examples into your favorite text editor. Then, simply save the file with any name you choose with an extension of .sgml. Then practice running your SGML parser on the file and then look at the html file you created with your favorite browser. If you need more help on the options for SGMLtools simply type:

```
bash$ sgmltools --help
```

1.2.2. SGMLtools-lite

Sgmltools-lite is the replacement to SGMLtools-2 but is only apt-gettable in unstable. Luckily, I came upon a version for potato. It can be downloaded from here (http://people.debian.org/~sano/sgmltools-lite/). I recommend you install SGMLtools-2 first using the instructions in sgmltools-2, and then use dpkg to upgrade the package. Since it's not apt-gettable you'll have to install/upgrade it like this (as root):

```
bash# dpkg -i sgmltools-lite_3.0.2.3.cvs0-0potato2_all.deb
```

As with all things Debian, upgrades rarely change the way you do things. Sgmltools-lite uses the same command and options (almost) as SGMLtools-2. Here's how you would use SGMLtools-lite to render your SGML file as html:

```
bash$ sgmltools -b html -s howtoh name-of-sgml-file.sgml
```

And for print based documents do this:

```
bash$ sgmltools -b html -s howtop name-of-sgml-file.sgml
```

1.2.3. Docbook: The definitive guide

Once you have mastered this document you will need this reference (it will probably help even before that!). You can browse this book online at http://www.docbook.org/tdg/html/docbook.html (http://www.docbook.org/tdg/html/docbook.html) or you can download it at http://www.docbook.org/tdg/index.html (http://www.docbook.org/tdg/index.html) If you do download it get the ZIP archive that is the same as the online version. This file will be called html.zip and when you unzip it it will put everything in a directory called html. Also, pay attention to their note that the root page is docbook.html, not index.html. You will need the program unzip to unzip it. This program is non-free so make sure you have the appropriate lines in your `/etc/apt/sources.list` file. At the command line do (the first command as root, the second as any user):

```
bash# apt-get install unzip

Then do:

bash$ unzip html.zip
```

There are currently three versions of the DocBook guide available on the DocBook site. Version 2.0.4 is the most current edition, as well as the most well-written. However, that book describes DocBook V4.2, which is not exactly the same as the version we use, LinuxDoc (DocBook V3.1). The 2.0.3 version of the book covers DocBook V4.1.2, which is almost exactly the same as V4.2, though it's not as detailed. Finally, Version 1.0.3 covers DocBook V3.1/LinuxDoc, but, again, is less detailed than the others.

Our suggestion is to use the latest version of the book, even if it applies to a newer version. Anything describing something that was a part of V3.1 will still work, but be aware that there are some new additions that may not work as expected.

1.3. Using the custom stylesheets

If you installed SGMLtools-2 you should already have a working system to render SGML documents into the format you choose. So this section is a little bit ahead of itself. If you wish, you can easily skip to the next section and continue on, coming back later when want your documents to be formatted in the same manner as this one.

In order to format your documents in the same manner as this one you should use the NewbieDoc custom stylesheets. The stylesheets are called `newbiedoc-html.dsl`, `newbiedoc-onehtml.dsl`, and `newbiedoc-pdf.dsl`. You can download them from newbiedoc cvs (http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/newbiedoc/newbiedoc/metatools/) or you can post to: `<newbiedoc-discuss@lists.sourceforge.net>`. Someone on the mailing list should be able to get you a copy. Once you get the custom stylesheets, save the stylesheets in the same directory as the SGML file you wish to parse. Here's the formal usage of SGMLtools:

sgmltools {-b {html | onehtml | pdf | ps}} [-s | *stylesheet*] {*SGML-file*}

If you don't understand the above syntax don't worry about it. I'll be giving specific examples of how to use the stylesheets in the following sections.

1.3.1. newbiedoc-html.dsl

Here's how you use the custom stylesheet for html using SGMLtools-2:

```
bash$ sgmltools -b html -s newbiedoc-html.dsl name-of-SGML-file.sgml
```

You should execute the command exactly as you see above, substituting the name of your SGML file for `name-of-SGML-file.sgml`. This will create a subdirectory called `name-of-SGML-file` with your SGML file rendered as multiple html documents inside it. The SGML document is split into seperate html files at each <sect1> tag.

1.3.2. newbiedoc-onehtml.dsl

This stylesheet, as the name implies, creates one single html file as its output. Here's how you use the custom stylesheet for onehtml using SGMLtools-2:

```
bash$ sgmltools -b onehtml -s newbiedoc-onehtml.dsl name-of-SGML-file.sgml
```

You should execute the command exactly as you see above, substituting the name of your SGML file for `name-of-SGML-file.sgml`. This will create a single html file called `name-of-SGML-file.html`.

1.3.3. newbiedoc-pdf.dsl

Here's how to use the pdf stylesheet:

```
bash$ sgmltools -b pdf -s newbiedoc-pdf.dsl name-of-SGML-file.sgml
```

Again, you should execute the command exactly as you see above, substituting the name of your SGML file for `name-of-SGML-file.sgml`. This will create a file called `name-of-SGML-file.pdf`.

2. The basic SGML document

The basic SGML document consists of a DTD or Document Type Declaration, one of several top level elements (otherwise known as tags or markups), paragraphs and text. The top level element should be a <book>, <chapter>, <article>, or <sect1>, depending on the type of document you are writing. We will be using <article> for our documents. Here is an example of a simple SGML document:

```
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook V3.1//EN">

<article>
   <sect1 id="introduction"><title>Hello world introduction</title>

      <para>
```

```
      Hello world!
      </para>

   </sect1>
</article>
```

For the rest of the tutorial I will use element, tag and markup interchangeably; they are synonymous. The first line is the DTD or Document Type Declaration. Notice that the first and last tags are both <article> and </article> tags. All other markup will be "contained" by those two tags. Notice the <sect1> tag. It has an attribute called "id". Don't worry about attributes for now; just know that all <sectX> tags where X is a number between 1 and 5 must have an "id" attribute if you want automatic hyperlinks created for HTML documents when you run the SGML parser on the file. Also, every <sectX> tag requires at least the paragraph tags and and ending </sectX> tag.

Also notice that the DTD declares the document as an article, which allows you to use the <article> element as the top-level tag in the first place. If, for example, it said 'book' instead of 'article', the SGML parser would fail to render the document.

2.1. General structure of SGML

A good doc writer always makes sure he puts a license in the SGML file. So here is what our new SGML file looks like:

```
<!--
Copyright (c)  2001  your name, NewbieDoc project;
http://sourceforge.net/projects/newbiedoc
Permission is granted to copy, distribute and/or modify this
document under the terms of the GNU Free Documentation License,
Version 1.1 or any later version published by the Free Software
Foundation; with no Invariant Sections, with no Front-Cover
Texts, and with no Back-Cover Texts. A copy of the license can
be found at http://www.fsf.org/copyleft/fdl.html.
-->

<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook V3.1//EN">

<article id="hello-world">
   <sect1 id="introduction"><title>Hello world introduction</title>

      <para>
      Hello world!
      </para>

   </sect1>
</article>
```

Notice how we commented out the license using the <!-- and the -->. This is important; if you forget this you will get all kinds of errors when you run the file through the SGML parser. This information will not be viewable once you build it. The reason it is not viewable is the parser thinks it's just a comment (and it is!) so it just drops it out of the final parsed document. This is nice because we can add comments in our files to remind ourselves to do things later on. This presents us with a problem. There is no viewable license in our final document. That is fine. We will get to that later.

Also, notice that I added an "id" attribute to the article tag. This id is what the HTML file will be named after the document is parsed. To really understand this, do this as an exercise. Either type, or cut and paste the above code into the text editor of your choice. Save the file as `my1st.sgml`. Go to the command line and change to the directory the `my1st.sgml` is in. Execute this command:

```
bash$ sgmltools -b html my1st.sgml
```

Now look in that directory. You should see a new directory called `my1st`. If you change into that directory and look at it's contents you will see a file called `hello-world.html`. When you built this document the parser used the article's "id" attribute to create the name of the output file. The actual name of your SGML file is used to create the subdirectory name. This isn't that important now because we only have one html file, however, once you start adding multiple sections, you will see many more files in that subdirectory. If you fail to include an id tag for <article>, the HTML file will be arbitrarily named as something like "t1.html", which is no help at all.

Congratulations! You just built your first SGML file into another file format. Now open up your favorite html browser and look at your finished work.

Throughout the remaining sections I will show you snippets of docbook SGML code which you can insert into this same document (`my1st.sgml`). You should do that and experiment until you get the hang of it. Try pasting several things into it and then build it with the parser. Refresh you html browser and notice the changes. Then add a few more, build, refresh your browser. Repeat it as many times as necessary till you are comfortable. Then start adding content and create your own documents!

3. Article Headers

Article headers provide information about the article. Below is some sample code for an article header and an explanation of the code.

```
<!-- **License cut out for clarity -->
<article>  <--This tag for illustration, cut & paste below this line

<artheader>
   <title>Article Template</title>
      <author>
      <firstname>John</firstname>
      <surname>Doe</surname>
         <affiliation>
            <address>
               <email>your-email-address@isp.com</email>
            </address>
         </affiliation>
      </author>

   <revhistory>
      <revision>
      <revnumber>v1</revnumber>
      <date>30 March 2001</date>
      <authorinitials>jd</authorinitials>
```

```
      <revremark>
      This is the initial release.
      </revremark>
      </revision>
   </revhistory>

   <abstract>
      <para>
      This document is intended to help newbies do ...
      Copyright &copy; 2001 <ulink url="http://sourceforge.net/projects/newbiedoc">
      NewbieDoc project</ulink>. Permission is granted to copy, distribute and/or
      modify this document under the terms of the GNU Free Documentation License,
      Version 1.1 or any later version published by the Free Software Foundation;
      with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover
      Texts. A copy of the license can be found at the
      <ulink url="http://www.fsf.org/copyleft/fdl.html">Free Software Foundation</ulink>.
      </para>
   </abstract>

</artheader>

<sect1 id="the1stsection">  <--This tag for illustration, cut & paste above this line
```

Notice that all the article header information fits between your first <article> tag and your first <sect1> tag. Most of the code is self-documenting. To get a look at what this looks like parsed just look at the first page of this article. Or simply cut and paste it into your `my1st.sgml` document (taking care to paste it between your <article> tag and your first <sect1> tag), then build it.

4. Sections

The top level tag in our slowly growing document is <article>. Because this is the top level tag it will "contain" all other tags. In other words, all the other markup and information that we want to show up in the document will be between the starting <article> tag and the ending </article> tag. The real structure of your document is created with the <sectX> tags. Let's add some more section tags and also do some nesting of <sectX> tags. (where X is a number between 1 and 5.) For example, here we have 3 sections:

```
<!-- license cut out for clarity -->
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
<article>
   <sect1 id="introduction"><title>Hello world introduction</title>
      <para>
      Hello world!
      </para>
   </sect1>

   <sect1 id="main-body"><title>Main Body</title>
      <para>
      This is the main body
      </para>
   </sect1>
```

```
   <sect1 id="conclusion"><title>Conclusion</title>
      <para>
      In conclusion...
      </para>
   </sect1>
</article>
```

Now we will add a few subsections by nesting some <sectX> tags. Below is an example and explanation of the code:

```
<!-- license cut out for clarity -->
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
<article>
   <sect1 id="introduction"><title>Hello world introduction</title>
      <para>
      Hello world!
      </para>

      <sect2 id="credits"><title>Credits</title>
         <para>
         I would like to take this time to thank the creators of Linux!
         </para>
      </sect2>
   </sect1>

   <sect1 id="main-body"><title>Main Body</title>
      <para>
      This introduces the main body
      </para>

      <sect2 id="point1"><title>Point1</title>
         <para>
         My first point is ...
         </para>
      </sect2>

      <sect2 id="point2"><title>Point2</title>
         <para>
         My second point is ...
         </para>
      </sect2>
   </sect1>

   <sect1 id="conclusion"><title>Conclusion</title>
      <para>
      In conclusion...
      </para>
   </sect1>
</article>
```

This would create a subsection under the Introduction called Credits, Point1 under the Main Body, and Point2 under the Main Body. You can nest these sections any way you see fit. The only limitation is you can not nest any deeper then <sect5>. You must also remember that you can only nest <sectX> tags with a higher number. In other words, you can *not* nest a <sect1> tag inside a <sect2> tag. You could nest a <sect3>, <sect4>, or <sect5> tag inside a <sect2> tag but *not* a <sect1>.

If you render this document, you will see that the HTML files created match the <sect1> elements!

5. Paragraphs

Paragraph tags are simple. You have already seen them in the previous examples. Just make sure you close them with their end tags. Here is a simple example:

```
<para>
I am a paragraph!
</para>

<para>
I am the next paragraph!!
</para>
```

and here is what it looks like formatted:

I am a paragraph!

I am the next paragraph!!

6. URL's or links

There are three types of links we will cover here. They are <ulink>, <link>, and <xref>. Here is an example of all three:

```
<para>
In this sentence <link linkend="admon-docbook-guide">this</link> word is
hot and points to the following section.
</para>

<para>
There is also a link to the
<ulink url="http://www.debian.org">Debian home page</ulink>
in this sentence.
</para>

<para>
Of course, we must cross-reference the <xref linkend="admon-docbook-guide"> section.
</para>
```

Below is how they look formatted and an explanation of the code.

In this sentence this word is hot and points to the following section.

There is also a link to the Debian home page (http://www.debian.org) in this sentence.

Of course, we must cross-reference the Admonitions section.

The <link> is the easiest and the most common. You can use it for hyperlinks within the document and "off-site", however, <ulink> is the proper tag for "off-site" links. The attribute "linkend" points to the "id" attribute of another element. In this case, the linkend points to a later section, Admonitions. If you were to look at the admonitions section it's code would look like this:

```
<sect1 id="admon-docbook-guide" xreflabel="Admonitions">
```

As you can see, the "linkend" points directly at the Admonition <sect1>'s "id" attribute.

The "url" attribute of <ulink> points to Debian's web site. It is proper to include the "http://" prior to the web site. Some browsers may not handle this well if you leave that out.

<xref>'s are a little more complicated. You'll notice that they have no ending tag. They get the text they are represented as from the targets "xreflabel" attribute if it has one. If it doesn't have one, the parser will guess at what it should be. It usually guesses correctly, however, it may not be totally correct. If you use the <xref> tag in your documents it's a good idea to use the xreflabel liberally on all elements within your document. If you look at the Admonitions SGML code you'll see it has an "xreflabel" attribute.

7. Lists

Lists are fairly simple as well. Let's add one to our growing document.

```
<itemizedlist>
   <listitem>
   <para>
   Item 1
   </para>
   </listitem>

   <listitem>
   <para>
   Item 2
   </para>
   </listitem>

   <listitem>
   <para>
   Item 3
   </para>
   </listitem>
</itemizedlist>
```

And here is what that list looks like after it's run through the SGML parser:

- Item 1
- Item 2
- Item 3

Here's how you create one which has numbers:

```
<!-- we could also change the numeration attribute to equal any of Arabic, Loweralpha,
```

```
     Lowerroman, Upperalpha, Upperroman
-->

<orderedlist numeration="Arabic">
   <listitem>
   <para>One</para>
   </listitem>

   <listitem>
   <para>Two</para>
   </listitem>

   <listitem>
   <para>Three</para>
   </listitem>

   <listitem>
   <para>Four</para>
   </listitem>
</orderedlist>
```

Notice I added a comment just above our list. We can change the numeration value to equal any of those values and get different types of numbered lists! And here is what that one looks like:

1. One
2. Two
3. Three
4. Four

8. Examples

Examples are not all that difficult. Here is one:

```
<example id="c-example"><title>A C example</title>
   <programlisting>
   #include &lt;stdio.h&gt;
   main() {
      printf("Hello world\n");
   }
   </programlisting>
</example>
```

Here is what it looks like parsed:

Example 1. A C example

```
#include <stdio.h>
main() {
   printf("Hello world\n");
}
```

Notice that in this case the lines on the example are numbered. This is because we used the <programlisting> tag instead of the <screen> tag. We'll cover this in more detail when we get to the Programlisting and Screens section. This is a feature we enabled using the `newbiedoc-html.dsl` custom stylesheet. If you didn't render your document using the custom stylesheet you won't get the line numbers. You don't need the line numbers but if it is something you want to help with your presentation. Check out the Using the custom stylesheets section and then the Programlisting and Screens section for how to use those tags properly.

9. Tables

Tables are a bit more complex and you should reference [DocBook: The Definitive Guide (http://www.docbook.org/tdg/html/docbook.html)] to get more information on them. Here is a simple example:

```
<table frame=all><title>Sample Table</title>
<tgroup cols=2 align=left colsep=1 rowsep=1>

<thead>
   <row>
   <entry>Examples</entry>
   <entry>What they mean</entry>
   </row>
</thead>

<tbody>
   <row>
      <entry>s1 == s2</entry>
      <entry>s1 matches s2</entry>
   </row>

   <row>
      <entry>s1 != s2</entry>
      <entry>s1 does not match s2</entry>
   </row>

   <row>
      <entry>s1 < s2</entry>
      <entry>s1 is less than s2</entry>
   </row>

   <row>
      <entry>s1 > s2</entry>
```

```
      <entry>s1 is greater than s2</entry>
   </row>

   <row>
      <entry>-n s1</entry>
      <entry>s1 is not null (contains one or more characters)</entry>
   </row>

   <row>
      <entry>-z s1</entry>
      <entry>s1 is null (Does NOT contain any characters)</entry>
   </row>
</tbody>
</tgroup>
</table>
```

And here's the table parsed:

Table 1. Sample Table

Examples	What they mean
s1 == s2	s1 matches s2
s1 != s2	s1 does not match s2
s1 < s2	s1 is less than s2
s1 > s2	s1 is greater than s2
-n s1	s1 is not null (contains one or more characters)
-z s1	s1 is null (Does NOT contain any characters)

10. Programlisting and Screens

Programlisting or screens are tags used to show something you would see on a computer screen (in the case of screen) and/or show some code. Much of this document was written using <screen> tags. Here's an example of a programlisting:

```
<programlisting>
#include &lt;stdio.h&gt;
main() {
printf("Hello world\n");
}
</programlisting>
```

One thing you should note about <programlisting> is that you can have markup included inside the tags and it will get rendered. Here, to get the desired effect of producing a "<" and ">" I need to use the character entities (in this case that is the < and the >) in the above code. Otherwise the parser will think that <stdio.h> is an SGML tag and try to render it. We'll cover character entities later but for now just realize that you have be careful. Another thing to notice is the formatted version below has line numbers. This is something which will only appear if you used the newbiedoc custom stylesheets. If you're using the custom stylesheets make sure you only use <programlisting> when you want to explain something using line numbers. Otherwise, use the <screen> tag. If you would like to use the custom stylesheets see the Using the custom stylesheets section. The final thing to notice is that the position of the text does matter. For instance, The above programlisting was indented one tab from the left edge of the screen in the SGML source file. Here's what it'll look like formatted:

```
#include <stdio.h>
main() {
printf("Hello world\n");
}
```

It's difficult to illustrate this so I will show an extreme example. Here's the same code indented 5 tabs from the left edge of the screen in the SGML source file:

```
                <programlisting>
                #include &lt;stdio.h&gt;
                main() {
                printf("Hello world\n");
                }
                </programlisting>
```

And here is what that looks like rendered:

```
                #include <stdio.h>
                main() {
                printf("Hello world\n");
                }
```

Do you see how that made a difference? The effect is the same with <screen> tags.

As far as <screen> tags go consider them to be the same as <programlisting>. The only difference being the line numbers if you're using the custom stylesheets as mentioned earlier.

11. Admonitions

Admonitions are used to draw attention to a specific subject. Here is a note:

```
<note><title>Please Note:</title>
   <para>
   Using a hammer to put together your computer is bad.
   </para>
</note>
```

Depending on what stylesheet you use to render your document you may need the admonition graphics for this to render properly. This is true of the default stylesheet if you followed my installation instructions at the beginning of this document or you are using the custom stylesheets. You can configure your parser to use another stylesheet like the ldp.dsl but I won't be covering that here...yet. Also, the <title> tags are optional for a note. Here's what it looks like.

Please Note:

Using a hammer to put together your computer is bad.

Here is an <important> and a <tip> admonition:

```
<important><title>Important!</title>
   <para>
   Watch where you're swinging that hammer!
   </para>
</important>

<tip><title>Tip</title>
   <para>
   Do not hit your thumb with the hammer, it hurts!
   </para>
</tip>
```

Once again, the <title> tags are optional. Here is what they look like parsed:

Important!

Watch where you're swinging that hammer!

Tip

Do not hit your thumb with the hammer, it hurts!

Here is a <caution> and a <warning> admonition:

```
<caution><title>Caution</title>
   <para>
   Hitting your thumb with a hammer may lead to an unwanted trip to the hospital!
```

```
      </para>
   </caution>

   <warning><title>Warning</title>
      <para>
      Do not, under any circumstances, admit that you hit your own thumb with a hammer.
      The ridicule you will face is astounding!
      </para>
   </warning>
```

Once again, the <title> tags are optional and are usually left out. I placed them here just so you know you can use them. Here is what they look like parsed:

Caution

Hitting your thumb with a hammer may lead to an unwanted trip to the hospital!

Warning

Do not, under any circumstances, admit that you hit your own thumb with a hammer. The ridicule you will face is astounding!

The trick to making your admonition graphics show up for html documents is to keep a directory called `images` "on the same level" as your html files which includes all the necessary graphics. For a little more information check out the Graphics section. If you'll be posting your documents to newbiedoc cvs (and we hope you do!) you shouldn't have to worry about the admonition graphics. There are already copies in cvs and on the website which will allow your admonition graphics to render in html properly.

12. Callouts

Callouts are used to draw attention to a specific area. They are a little complex so it's best if you experiment with them a little as you go. Here is an example <callout>:

```
<screen>
bash@host:~/cvs/newbiedoc$ ls -l
total 48
```

```
<co id="perm">drwxr-sr-x    2 jesse    jesse        4096 May  4 16:26 CVS<co id="cvs">
drwxr-sr-x    3 jesse    jesse        4096 Mar 29 03:29 dev
drwxr-sr-x    3 jesse    jesse        4096 Apr  8 19:31 general
drwxr-sr-x    3 jesse    jesse        4096 Apr  9 00:15 images
-rw-r--r--    1 jesse    jesse        4133 Apr 22 05:18 index.sgml
drwxr-sr-x    3 jesse    jesse        4096 Apr  2 02:25 metadoc
drwxr-sr-x    3 jesse    jesse        4096 May  4 19:33 metatools
drwxr-sr-x    3 jesse    jesse        4096 Apr  9 02:02 system
drwxr-sr-x    3 jesse    jesse        4096 Mar 29 01:24 text_editing
drwxr-sr-x    3 jesse    jesse        4096 May  4 00:17 tips
drwxr-sr-x    3 jesse    jesse        4096 Mar 29 01:24 utils
</screen>

<calloutlist>
   <callout arearefs="cvs">
      <para>
      This is the CVS directory.  CVS files are stored here.
      </para>
   </callout>

   <callout arearefs="perm">
      <para>
      These are the permissions for the CVS directory.
      </para>
   </callout>
</calloutlist>
```

Below is an explanation and how it looks formatted.

```
bash@host:~/cvs/newbiedoc$ ls -l
total 48
❶drwxr-sr-x    2 jesse    jesse        4096 May  4 16:26 CVS❷
drwxr-sr-x    3 jesse    jesse        4096 Mar 29 03:29 dev
drwxr-sr-x    3 jesse    jesse        4096 Apr  8 19:31 general
drwxr-sr-x    3 jesse    jesse        4096 Apr  9 00:15 images
-rw-r--r--    1 jesse    jesse        4133 Apr 22 05:18 index.sgml
drwxr-sr-x    3 jesse    jesse        4096 Apr  2 02:25 metadoc
drwxr-sr-x    3 jesse    jesse        4096 May  4 19:33 metatools
drwxr-sr-x    3 jesse    jesse        4096 Apr  9 02:02 system
drwxr-sr-x    3 jesse    jesse        4096 Mar 29 01:24 text_editing
drwxr-sr-x    3 jesse    jesse        4096 May  4 00:17 tips
drwxr-sr-x    3 jesse    jesse        4096 Mar 29 01:24 utils
```

❷ This is the CVS directory. CVS files are stored here.

❶ These are the permissions for the CVS directory.

First we'll start with the <callout> tags themselves. As you can see above they can pretty much be placed anywhere as long as they are contained by the <calloutlist> and </calloutlist> tags. They must have an arearefs="something" attribute. This is essentially a "pointer" which points to the <co> tag which tells the parser where to place the callout graphic. You'll notice that the graphics are in reverse order. This is because the parser parses the code from left to right and top to bottom, one line at a time. I did this by accident but it illustrates something to watch for. In html documents, the callouts are also hyperlinked. A maximum of 10 total callouts can be used. After that, plain text numbers will be used. To get a better understanding of callout graphics, experiment, and reference [DocBook: The Definitive Guide (http://www.docbook.org/tdg/html/docbook.html)]

13. CDATA

CDATA stands for character data. This technically is an entity which tells the parser that a stream of characters is to follow until it reaches its end tag. I have used this entity quite a bit in the writing of this document. The advantage of using this is that the parser ignores all tags within the CDATA "container". Here's what it looks like:

```
<![ CDATA [
This text will not <be> parsed no matter <what> I put in <here>.  Even if I put <illegal>
tags!
]]>
```

Usually, this is used inside a <programlisting> or <screen> tag because program code or user input on a screen many times contains characters which may confuse the parser. Here's what the above looks like formatted within a <screen>:

```
This text will not <be> parsed no matter <what> I put in <here>.  Even if I put <illegal>
tags!
```

And here's what it looks like when it is "inline":

This text will not <be> parsed no matter <what> I put in <here>. Even if I put <illegal> tags!

The key here is to note that the entity begins with "<![". Then the CDATA entity is given. Then the first "container" marker for the CDATA is "[". You then input the text you want treated as text only (i.e. don't parse it). You close the "container" with "]]>".

14. Character Entities

Character entities are sort of like variables. Because their are so many types of characters out there, the creators of SGML decided to create a huge list of entities. These are just variables which are called to represent some specific character. Here are a few that I've used in writing this document and a few that may be useful in your own documents. For a full list of possible character entities consult [DocBook: The Definitive Guide (http://www.docbook.org/tdg/html/docbook.html)].

```
&lt;  &gt;  &copy;  &auml;  &Auml;  &euml  &Euml;  &ouml; &Ouml;  &uuml;  &Uuml;
```

And here is what they look like:

< > © ä Ä ë Ë ö Ö ü Ü

15. Citations

Citations are fairly simple. Good doc writers use them liberally to avoid plagiarism and give credit where credit is due. Here's an example:

```
<citation><xref linkend="docbook"></citation>
or
<citation>
   <ulink url="http://www.docbook.org/tdg/html/docbook.html">
   DocBook: The Definitive Guide
   </ulink>
</citation>
```

And here's what it looks like formatted (Note: I left out the first one because I don't have a cross reference to it in this doc, I'm only showing the second one.)

[DocBook: The Definitive Guide (http://www.docbook.org/tdg/html/docbook.html)]

16. Graphics

Graphics are not all that difficult but you have to do a little advanced planning. Here's some example SGML for a graphic:

```
<mediaobject>
   <imageobject>
      <imagedata fileref="images/newbieDocLogotype.ps" format="ps">
   </imageobject>

   <imageobject>
      <imagedata fileref="images/newbieDocLogotype.eps" format="eps">
   </imageobject>

   <imageobject>
      <imagedata fileref="../images/newbieDocLogotype.gif" format="gif">
   </imageobject>

   <textobject>
      <phrase>Newbiedoc: Docs for & by Debian newbies.</phrase>
   </textobject>

   <caption>
      <para>
      Newbiedoc: Docs for & by Debian newbies.
      </para>
   </caption>
</mediaobject>
```

You'll notice that there are two separate file formats listed. Seems redundant, but really what it's doing is giving the parser a choice. When your parser is run for rtf or a "printed" file format it uses the eps graphic. If you're outputting to html format then the other format is used. The "fileref" attribute gives the relative path to where the graphic is stored. The important thing to remember here is that for the "printed" version this is where the graphic "lives" prior to you rendering the document (it needs this because the graphic will be *included* in the document). The "fileref" attribute for the html version simply provides a link to the graphic and this is where the graphic "lives" after the document is rendered. Unfortunately, the only graphic formats currently permitted are mostly non-free types, so .png is outta there for now. The current list of supported formats is: "BMP", "CGM-CHAR", "CGM-BINARY", "CGM-CLEAR", "DITROFF", "DVI", "EPS", "EQN", "FAX", "GIF", "GIF87A", "GIF89A", "JPG", "JPEG", "IGES", "PCX", "PIC", "PS", "SGML", "TBL", "TEX", "TIFF", "WMF", and "WPG". Here's what it looks like parsed:

Newbiedoc: Docs for & by Debian newbies.