Linuxdoc-SGML User's Guide

by Matt Welsh. Updated by Greg Hankins, greg.hankins@cc.gatech.edu v1.5, 8 March 1996

This document is a user's guide to the Linuxdoc-SGML formatting system, a SGML-based system which allows you to produce a variety of output formats. You can create plain text output (ASCII and ISO-8859-1), DVI, PostScript, HTML, GNU info, LyX, and RTF output from a single SGML source file. This guide documents Linuxdoc-SGML version 1.5.

Contents

1 Introduction **2**

1.1 About This Document . . . 2

1.2 Why SGML? . . . 2

1.3 How It Works . . . 3

2 Installation **4**

2.1 Installing The Software . . . 5

2.2 Formatting SGML Documents . . . 6

2.2.1 Creating Plain Text Output . . . 6

2.2.2 Creating LaTeX, DVI or PostScript Output . . . 6

2.2.3 Creating HTML Output . . . 6

2.2.4 Creating GNU Info Output . . . 7

2.2.5 Creating LyX Output . . . 7

2.2.6 Creating RTF Output . . . 7

2.2.7 Checking SGML Syntax . . . 7

2.3 ISO 8859-1 Character Set . . . 7

3 Writing Documents With Linuxdoc-SGML **8**

3.1 Basic Concepts . . . 8

3.2 Special Characters . . . 8

3.3 Verbatim and Code Environments . . . 9

3.4 Overall Document Structure . . . 10

3.4.1 The Preamble . . . 10

3.4.2 Sectioning And Paragraphs . . . 11

3.4.3 Ending The Document 12
3.5 Cross-References 12
3.6 Fonts 13
3.7 Lists 13
3.8 Further Information 14

1 Introduction

This is a user's guide to the Linuxdoc-SGML document processing system, for use with Linux documentation. Linuxdoc-SGML is a SGML DTD (Document Type Definition) and set of "replacement files" which convert the SGML to groff, LaTeX, HTML, GNU info, LyX, and RTF source.

Linuxdoc-SGML is based heavily on the QWERTZ DTD by Tom Gordon, `thomas.gordon@gmd.de`. I have only made revisions to his DTD and replacement files for use with Linux documentation.

Linuxdoc-SGML is not meant to be a general document-processing system. Although it can be used for documents of many types, I have tailored it for use by the Linux documenters in producing HOWTOs, FAQs, and (later) the Linux Documentation Project manuals. Therefore, I have tweaked features into and out of the system for this purpose. If you see a lack of generality in the system, that is the reason. However, there's nothing binding Linuxdoc-SGML to Linux documentation, but all documents produced by the system will look a certain way. If you want things to look differently I suggest that you use a more generalized system such as the plain QWERTZ DTD, in combination with the Linuxdoc-SGML tools.

One of the goals of this system is to make documents easy to produce in numerous formats. Until now, most Linux documentation has been produced in plain text through manual editing. A system like groff can take care of the plain text formatting, but that still doesn't give you HTML (for use on the World Wide Web), LaTeX (for nicely printed documents), or other various output formats. Therefore, if there are features missing from this system that you would like, *please let me know!* The idea is that we shouldn't have to use a lot of hackery to produce good-looking documentss in multiple formats. The author should have to do as little as possible.

1.1 About This Document

This document is written using the Linuxdoc-SGML DTD. It contains more or less everything you need to know to write SGML documents with this DTD. See `example.sgml` for an example of an SGML document that you can use as a model for your own documents.

1.2 Why SGML?

I chose SGML for this system because SGML is made specifically for translation to other formats. SGML, which stands for Standard Generalized Markup Language, allows you to specify the *structure* of a document—that is, what kinds of things make up the document. You specify the structure of a document with a DTD (Document Type Definition). Linuxdoc-SGML is one DTD that specifies the structure for Linux HOWTOs

and other documents. QWERTZ is another DTD; the SGML standard provides DTD's for books, articles, and other generic document types.

The DTD specifies the names of "elements" within the document. An element is just a bit of structure—like a section, a subsection, a paragraph, or even something smaller like *emphasized text*. Unlike LaTeX, however, these elements are not in any way intrinsic to SGML itself. The Linuxdoc-SGML DTD happens to define elements that look a lot like their LaTeX counterparts—you have sections, subsections, verbatim "environments", and so forth. However, using SGML you can define any kind of structure for the document that you like. In a way, SGML is like low-level TeX, while the Linuxdoc-SGML DTD is like LaTeX.

Don't be confused by this analogy. SGML is *not* a text-formatting system. There is no "SGML formatter" per se. SGML source is *only* converted to other formats for processing. Furthermore, SGML itself is used only to specify the document structure. There are no text-formatting facilities or "macros" intrinsic to SGML itself. All of those things are defined within the DTD. You can't use SGML without a DTD—a DTD defines what SGML does.

1.3 How It Works

Here's how processing a document with SGML and the Linuxdoc-SGML DTD works. First, you need a DTD. I'm using the QWERTZ DTD which was produced, originally, by a group of people who needed a LaTeX-like DTD. I've modified the QWERTZ DTD to produce the Linuxdoc-SGML DTD for our purposes. The DTD simply sets up the structure of the document. A small portion of it looks like this:

```
<!element article - -
        (titlepag, header?,
         toc?, lof?, lot?, p*, sect*,
         (appendix, sect+)?, biblio?) +(footnote)>
```

This part sets up the overall structure for an "article", which is like a "documentstyle" within LaTeX. The article consists of a titlepage (`titlepag`), an optional header (`header`), an optional table of contents (`toc`), optional lists of figures (`lof`) and tables (`lot`), any number of paragraphs (`p`), any number of top-level sections (`sect`), optional appendices (`appendix`), an optional bibliography (`biblio`) and footnotes (`footnote`).

As you can see, the DTD doesn't say anything about how the document should be formatted or what it should look like. It just defines what parts make up the document. Elsewhere in the DTD the structure of the `titlepag`, `header`, `sect`, and other elements are defined.

You don't need to know anything about the syntax of the DTD in order to write documents. I'm just presenting it so you know what it looks like and what it does. You *do* need to be familiar with the document *structure* that the DTD defines. If not, you might violate the structure when attempting to write a document, and be very confused about the resulting error messages. We'll describe the structure of Linuxdoc-SGML documents in detail later.

The next step is to write a document using the structure defined by the DTD. Again, the Linuxdoc-SGML DTD makes documents look a lot like LaTeX—it's very easy to follow. In SGML jargon a single document written using a particular DTD is known as an "instance" of that DTD.

In order to translate the SGML source into another format (such as LaTeX or groff) for processing, the SGML source (the document that you wrote) is *parsed* along with the DTD by (you guessed it) the SGML *parser*. I'm using the `sgmls` parser by James Clark, `jjc@jclark.com`, who also happens to be the author of `groff`. We're in good hands. The parser (`sgmls`) simply picks through your document and verifies that it follows the structure set forth by the DTD. It also spits out a more explicit form of your document, with all "macros" and elements expanded, which is understood by `sgmlsasp`, the next part of the process.

`sgmlsasp` is responsible for converting the output of `sgmls` to another format (such as LaTeX). It does this using *replacement files*, which describe how to convert elements in the original SGML document into corresponding source in the "target" format (such as LaTeX or groff).

For example, part of the replacement file for LaTeX looks like:

```
<itemize>       +       "\\begin\{itemize\}      +
</itemize>      +       "\\end\{itemize\}        +
```

Which says that whenever you begin an `itemize` element in the SGML source, it should be replaced with

```
\begin\{itemize\}
```

in the LaTeX source. (As I said, elements in the Linuxdoc-SGML DTD are very similar to their LaTeX counterparts).

So, to convert the SGML to another format, all you have to do is write a new replacement file for that format that gives the appropriate analogies to the SGML elements in that new format. In practice, it's not that simple—for example, if you're trying to convert to a format that isn't structured at all like your DTD, you're going to have trouble. In any case, it's much easier to do than writing individual parsers and translators for many kinds of output formats; SGML provides a generalized system for converting one source to many formats.

Once `sgmlsasp` has completed its work, you have LaTeX source which corresponds to your original SGML document, which you can format using LaTeX as you normally would. Later in this document I'll give examples and show the commands used to do the translation and formatting. You can do this all on one command line.

But first, I should describe how to install and configure the software.

2 Installation

Get `linuxdoc-sgml-1.5.tar.gz` from one of the following ftp sites:

- `ftp://sunsite.unc.edu/pub/Linux/utils/text/linuxdoc-sgml-1.5.tar.gz`
- `ftp://tsx-11.mit.edu/pub/linux/docs/linuxdoc-sgml-1.5.tar.gz`
- `ftp://ftp.cc.gatech.edu/pub/people/gregh/linuxdoc-sgml/linuxdoc-sgml-1.5.tar.gz`

Uptodate patches to version 1.5 can be found at `ftp://ftp.cc.gatech.edu/pub/people/gregh/linuxdoc-sgml`.

You can also get uptodate information from the *Linuxdoc-SGML WWW Page* `<http://www.informatik.tu-muenchen.de/ schwarz/linuxdoc-sgml/>`.

The file `linuxdoc-sgml-1.5.tar.gz` contains everything that you need to write SGML documents and convert them to groff, LaTeX, HTML, GNU info, LyX, and RTF. In addition to this package, you will need the following tools - these are not required by the SGML system, but I suggest that you get them in order to format your documents and verify that they look all right before distributing them.

1. `groff`. You *need* version 1.08 or 1.09. Apparently some of the margin-handling in `groff` is in a state of flux from version to version; they both work, but you get slightly different results (particularly, with 1.09 the left margin isn't indented two characters as it is in 1.08. There is a way around it, but it looks terrible on 1.08). Versions previous to 1.08 *will not work*. You can get this from `ftp://prep.ai.mit.edu/pub/gnu`. There is a Linux binary version on `ftp://sunsite.unc.edu/pub/Linux/utils/text` as well. You will need `groff` to produce plain text from your SGML documents. `nroff` will *not* work!

2. TeX and LaTeX. This is available more or less everywhere; you should have no problem getting it and installing it (there is a Linux binary distribution on `sunsite.unc.edu`). Of course, you only need TeX/LaTeX if you want to format your SGML documents with LaTeX. So, installing TeX/LaTeX is optional.

3. If you want to view the generated HTML, I suggest getting NCSA Mosaic 2.6 or later, available on `ftp://sunsite.unc.edu/pub/Linux/system/Network/info-systems`.

4. `gawk` and the GNU info tools, for formatting and viewing info files. These are also available on `ftp://prep.ai.mit.edu/pub/gnu`, or on `ftp://sunsite.unc.edu/pub/Linux/utils/text` (for `gawk`) and `ftp://sunsite.unc.edu/pub/Linux/system/Manual-pagers` (for GNU info tools). `awk` will not work without hacking.

5. LyX (a quasi-WYSIWYG interface to LaTeX, with SGML layouts), is available on `ftp://ftp.via.ecp.fr`.

2.1 Installing The Software

The steps needed to install and configure the `linuxdoc-sgml` stuff are as follows:

1. First, unpack the tar file `linuxdoc-sgml-1.5.tar.gz` somewhere. This will create the directory `linuxdoc-sgml-1.5`. It doesn't matter where you unpack this file; just don't move things around within the `linuxdoc-sgml-1.5` directory.

2. Read the `INSTALL` file - it has detailed installation instructions.

   If all went well, you should be ready to use the system.

2.2 Formatting SGML Documents

Let's say you have the SGML document `foo.sgml`, which you want to format. Here is a general overview of formatting the document for different output. For a complete list of options, consult the man pages.

2.2.1 Creating Plain Text Output

If you want to produce plain text, use the command:

```
% sgml2txt foo.sgml
```

Note that I have tailored the groff conversion for plain text output. That is, I've removed page headers, page numbers, changed the margins, and so on. With some hacking you can produce PostScript and DVI from the groff output, but I suggest that you use LaTeX for that instead.

You can also create groff source for man pages, which can be formatted with `groff -man`. To do this, do the following:

```
% sgml2txt -man foo.sgml
```

2.2.2 Creating LaTeX, DVI or PostScript Output

To create a LaTeX documents from the SGML source file, simply run:

```
% sgml2latex foo.sgml
```

If you want to produce PostScript output (via `dvips`), use the `-p` option:

```
% sgml2latex -p foo.sgml
```

Or, you can produce a DVI file using the `-d` switch, as so:

```
% sgml2latex -d foo.sgml
```

2.2.3 Creating HTML Output

If you want to produce HTML output, do this:

```
% sgml2html -img foo.sgml
```

This will produce `foo.html`, as well as `foo-1.html`, `foo-2.html`, and so on—one file for each section of the document. Run your WWW browser on `foo.html`, which is the top level file. Also make sure that all of the HTML files corresponding to your document are in one directory, as they reference each other with local URLs. The icons referenced in the HTML output are located in `$LINUXDOCLIB/icons`. These will also need to be copied to the final location of the HTML documents. `$LINUXDOCLIB` is defined at the beginning of the SGML conversion scripts.

If you use `sgml2html` without the `-img` flag, HTML documents will have the labels "Previous", "Next", and "Table of Contents" for navigation. You can override these defaults by creating a file in `$LINUXDOCLIB/rep/html/<filename>`, and substituting your own words for different languages. The file has the following format:

```
PrevPage:    newvalue
NextPage:    newvalue
TOC:         newvalue
```

See `deutsch` for an example.

2.2.4 Creating GNU Info Output

If you want to format your file for the GNU info browser, just run the following command:

```
% sgml2info foo.sgml
```

2.2.5 Creating LyX Output

For LyX output, use the the command:

```
% sgml2lyx foo.sgml
```

2.2.6 Creating RTF Output

If you want to produce RTF output, run the command:

```
% sgml2rtf foo.sgml
```

This will produce `foo.rtf`, as well as `foo-1.rtf`, `foo-2.rtf`, and so on—one file for each section of the document.

2.2.7 Checking SGML Syntax

If you just want to capture your errors from the SGML conversion, use the `sgmlcheck` script. For example.

```
% sgmlcheck foo.sgml
```

2.3 ISO 8859-1 Character Set

The ISO 8859-1 (latin1) character set may be used for international characters in plain text, LaTeX, HTML, LyX, and RTF output (GNU info support for ISO 8859-1 may be possible in the future). To use this feature, give the formatting scripts the `-l` flag, for example:

```
% sgml2txt -l foo.sgml
```

You also can use ISO 8859-1 characters in the SGML source, they will automatically be translated to the proper escape codes for the corresponding output format.

3 Writing Documents With Linuxdoc-SGML

For the most part, writing documents using the Linuxdoc-SGML DTD is very simple, and somewhat like LaTeX. However, there are some caveats to watch out for. In this section I'll give an introduction on writing SGML documents. See the file `example.sgml` for a SGML example document (and tutorial) which you can use as a model when writing your own documents. Here I'm just going to discuss the various features of SGML, but the source is not very readable as an example. Instead, print out the source (as well as the formatted output) for `example.sgml` so you have a real live case to refer to.

3.1 Basic Concepts

Looking at the source of the example document, you'll notice right off that there are a number of "tags" marked within angle brackets (< and >). A tag simply specifies the beginning or end of an element, where an element is something like a section, a paragraph, a phrase of italicized text, an item in a list, and so on. Using a tag is like using a LaTeX command such as `\item` or `\section{...}`.

As a simple example, to produce **this boldfaced text**, I typed

```
As a simple example, to produce <bf>this boldfaced text</bf>, ...
```

in the source. `<bf>` begins the region of bold text, and `</bf>` ends it. Alternately, you can use the abbreviated form

```
As a simple example, to produce <bf/this boldfaced text/, ...
```

which encloses the bold text within slashes. (Of course, you'll need to use the long form if the enclosed text contains slashes, such as the case with Unix filenames).

There are other things to watch out with respect to special characters (that's why you'll notice all of these bizarre-looking ampersand expressions if you look at the source; I'll talk about those shortly).

In some cases, the end-tag for a particular element is optional. For example, to begin a section, you use the `<sect>` tag, however, the end-tag for the section (which could appear at the end of the section body itself, not just after the name of the section!) is optional and implied when you start another section of the same depth. In general you needn't worry about these details; just follow the model used in the tutorial (`example.sgml`).

3.2 Special Characters

Obviously, the angle brackets are themselves special characters in the SGML source. There are others to watch out for. For example, let's say that you wanted to type an expression with angle brackets around it,

as so: <foo>. In order to get the left angle bracket, you must use the < element, which is a "macro" that expands to the actual left-bracket character. Therefore, in the source, I typed

```
angle brackets around it, as so: <tt>&lt;foo></tt>.
```

Generally, something beginning with an ampersand is a special macro. For example, there's % to produce %, | to produce |, and so on. For all "special characters" there exist these ampersanded-entities to represent them.

Usually, you don't need to use the ampersand macro to get a special character, however, in some cases it is necessary. The most commonly used are:

- Use & for the ampersand (&),
- Use < for a left bracket (<),
- Use > for a right bracket (>),
- Use &etago; for a left bracket with a slash (</)
- Use $ for a dollar sign ($),
- Use # for a hash (#),
- Use % for a percent (%),
- Use ˜ for a tilde (~),
- Use `` and '' for quotes, or use &dquot for ".

For a complete list of special characters, look at one of the replacement files. Usually LaTeX complains the most about special characters, so paging through $LINUXDOCLIB/rep/latex/general would be a good place to start. $LINUXDOCLIB is defined at the beginning of the SGML conversion scripts.

3.3 Verbatim and Code Environments

While we're on the subject of special characters, I might as well mention the verbatim "environment" used for including literal text in the output (with spaces and indentation preserved, and so on). The verb element is used for this; it looks like the following:

```
<verb>
  Some literal text to include as example output.
</verb>
```

The verb environment doesn't allow you to use *everything* within it literally. Specifically, you must do the following within verb environments.

- Use &ero; to get an ampersand,

- Use `&etago;` to get `</`,
- Don't use `\end{verbatim}` within a `verb` environment, as this is what LaTeX uses to end the `verbatim` environment. (In the future, it should be possible to hide the underlying text formatter entirely, but the parser doesn't support this feature yet.)

The `code` environment is much just like the `verb` environment, except that horizontal rules are added to the surrounding text, as so:

```
Here is an example code environment.
```

You should use the `tscreen` environment around any `verb` environments, as so:

```
<tscreen><verb>
Here is some example text.
</verb></tscreen>
```

`tscreen` is an environment that simply indents the text and sets the sets the default font to `tt`. This makes examples look much nicer, both in the LaTeX and plain text versions. You can use `tscreen` without `verb`, however, if you use any special characters in your example you'll need to use both of them. `tscreen` does nothing to special characters. See `example.sgml` for examples.

The `quote` environment is like `tscreen`, except that it does not set the default font to `tt`. So, you can use `quote` for non-computer-interaction quotes, as in:

```
<quote>
Here is some text to be indented, as in a quote.
</quote>
```

which will generate:

> Here is some text to be indented, as in a quote.

3.4 Overall Document Structure

Before we get too in-depth with details, I'm going to describe the overall structure of a document as defined by the Linuxdoc-SGML DTD. Look at `example.sgml` for a good example of how a document is set up.

3.4.1 The Preamble

In the document "preamble" you set up things such as the title information and document style. For a Linux HOWTO document this should look like:

```
<!doctype linuxdoc system>

<article>
```

```
<title>Linux Foo HOWTO
<author>Norbert Ebersol, <tt/norb@baz.com/
<date>v1.0, 9 March 1994
<abstract>
This document describes how to use the <tt/foo/ tools to frobnicate
bar libraries, using the <tt/xyzzy/ relinker.
</abstract>

<toc>
```

The elements should go more or less in this order. The first line tells the SGML parser to use the Linuxdoc-SGML DTD. The <article> tag forces the document to use the "article" document style. (The original QWERTZ DTD defines "report" and "book" as well; I haven't tweaked these for use with Linuxdoc-SGML).

The `title`, `author`, and `date` tags should be obvious; in the `date` tag include the version number and last modification time of the document.

The `abstract` tag sets up the text to be printed at the top of the document, *before* the table of contents. If you're not going to include a table of contents (the `toc` tag), you probably don't need an `abstract`. I suggest that all Linux HOWTOs use this same format for the preamble, so that the title, abstract, and table of contents are all there and look the same.

3.4.2 Sectioning And Paragraphs

After the preamble, you're ready to dive into the document. The following sectioning commands are available:

- `sect`: For top-level sections (i.e. 1, 2, and so on.)
- `sect1`: For second-level subsections (i.e. 1.1, 1.2, and so on.)
- `sect2`: For third-level subsubsections.
- `sect3`: For fourth-level subsubsubsections.
- `sect4`: For fifth-level subsubsubsubsections.

These are roughly equivalent to their LaTeX counterparts `section`, `subsection`, and so on.

After the `sect` (or `sect1`, `sect2`, etc.) tag comes the name of the section. For example, at the top of this document, after the preamble, comes the tag:

```
<sect>Introduction
```

And at the beginning of this section (Sectioning and paragraphs), there is the tag:

```
<sect2>Sectioning And Paragraphs
```

After the section tag, you begin the body of the section. However, you must start the body with a <p> tag, as so:

```
<sect>Introduction
<p>
This is a user's guide to the Linuxdoc-SGML document processing...
```

This is to tell the parser that you're done with the section title and are ready to begin the body. Thereafter, new paragraphs are started with a blank line (just as you would do in TeX). For example,

```
Here is the end of the first paragraph.

And we start a new paragraph here.
```

There is no reason to use <p> tags at the beginning of every paragraph; only at the beginning of the first paragraph after a sectioning command.

3.4.3 Ending The Document

At the end of the document, you must use the tag:

```
</article>
```

to tell the parser that you're done with the `article` element (which embodies the entire document).

3.5 Cross-References

Now we're going to move onto other features of the system. Cross-references are easy. For example, if you want to make a cross-reference to a certain section, you need to label that section as so:

```
<sect1>Introduction<label id="sec-intro">
```

You can then refer to that section somewhere in the text using the expression:

```
See section <ref id="sec-intro" name="Introduction"> for an introduction.
```

This will replace the `ref` tag with the section number labeled as `sec-intro`. The `name` argument to `ref` is necessary for groff and HTML translations. The groff macro set used by Linuxdoc-SGML does not currently support cross-references, and it's often nice to refer to a section by name instead of number.

For example, this section is 3.5.

There is also a `url` element for Universal Resource Locators, or URLs, used on the World Wide Web. This element should be used to refer to other documents, files available for FTP, and so forth. For example,

```
You can get the Linux HOWTO documents from
<url url="http://sunsite.unc.edu/mdw/HOWTO/"
     name="The Linux HOWTO INDEX">.
```

The `url` argument specifies the actual URL itself. A link to the URL in question will be automatically added to the HTML document. The optional `name` argument specifies the text that should be anchored to the URL (for HTML conversion) or named as the description of the URL (for LaTeX and groff). If no `name` argument is given, the URL itself will be used.

For example, you can get the Linuxdoc-SGML package from
`ftp://sunsite.unc.edu/pub/Linux/utils/text/linuxdoc-sgml-1.5.tar.gz`.

A useful variant of this is `htmlurl`, which suppresses rendering of the URL part in every context except HTML. What this is useful for is things like a person's email addresses; you can write

```
<htmlurl url="mailto:esr@snark.thyrsus.com"
           name="esr@snark.thyrsus.com">
```

and get "esr@snark.thyrsus.com" in text output rather than the duplicative "esr@snark.thyrsus.com <mailto:esr@snark.thyrsus.com>" but still have a proper URL in HTML documents.

3.6 Fonts

Essentially, the same fonts supported by LaTeX are supported by Linuxdoc-SGML. Note, however, that the conversion to plain text (through `groff`) does away with the font information. So, you should use fonts as much as possible, for the benefit of the conversion to LaTeX. But don't depend on the fonts to get a point across in the plain text version.

In particular, the `tt` tag described above can be used to get constant-width "typewriter" font which should be used for all e-mail addresses, machine names, filenames, and so on. Example:

```
Here is some <tt>typewriter text</tt> to be included in the document.
```

Equivalently:

```
Here is some <tt/typewriter text/ to be included in the document.
```

Remember that you can only use this abbreviated form if the enclosed text doesn't contain slashes.

Other fonts can be achieved with `bf` for **boldface** and `em` for *italics*. Several other fonts are supported as well, but I don't suggest you use them, because we'll be converting these documents to other formats such as HTML which may not support them. Boldface, typewriter, and italics should be all that you need.

3.7 Lists

There are various kinds of supported lists. They are:

- `itemize` for bulleted lists such as this one.
- `enum` for numbered lists.
- `descrip` for "descriptive" lists.

Each item in an `itemize` or `enum` list must be marked with an `item` tag. Items in a `descrip` are marked with `tag`. For example,

```
<itemize>
<item>Here is an item.
<item>Here is a second item.
</itemize>
```

Looks like this:

- Here is an item.
- Here is a second item.

Or, for an `enum`,

```
<enum>
<item>Here is the first item.
<item>Here is the second item.
</enum>
```

You get the idea. Lists can be nested as well; see the example document for details.

A `descrip` list is slightly different, and slightly ugly, but you might want to use it for some situations:

```
<descrip>
<tag/Gnats./ Annoying little bugs that fly into your cooling fan.
<tag/Gnus./ Annoying little bugs that run on your CPU.
</descrip>
```

ends up looking like:

Gnats.

Annoying little bugs that fly into your cooling fan.

Gnus.

Annoying little bugs that run on your CPU.

3.8 Further Information

- The QWERTZ User's Guide is available from `ftp://ftp.cs.cornell.edu/pub/mdw/SGML`. QWERTZ (and hence, Linuxdoc-SGML) supports many features such as mathematical formulae, tables, figures, and so forth. I don't recommend using most of these features in the Linux HOWTOs because they won't render well in plain text. If you'd like to write general documentation in SGML, I suggest using the original QWERTZ DTD instead of the hacked-up Linuxdoc-SGML DTD, which I've modified for use particularly by the Linux HOWTOs and other such documentation.

- Tom Gordon's original QWERTZ tools can be found at `ftp://ftp.gmd.de/GMD/sgml`.
- More information on SGML can be found at the following WWW pages:
  1. *SGML and the Web* `<http://www.w3.org/hypertext/WWW/MarkUp/SGML/>`
  2. *SGML Web Page* `<http://www.sil.org/sgml/sgml.html>`
- James Clark's `sgmls` parser, and it's successor `nsgmls` and other tools can be found at `ftp://ftp.jclark.com` and at *James Clark's WWW Page* `<http://www.jclark.com>`.
- You can join the Linuxdoc-SGML mailing list by sending mail to `majordomo@via.ecp.fr` with `subscribe linuxdoc-sgml` in the message body. The list address is `linuxdoc-sgml@via.ecp.fr`.
- More information on `LyX` can be found at the *LyX WWW Page* `<http://wsiserv.informatik.uni-tuebingen.de/ ettrich/>`. `LyX` is a high-level word processor frontend to LaTeX. Quasi-WYSIWYG interface, many LaTeX styles and layouts automatically generated. Speeds up learning LaTeX and makes complicated layouts easy and intuitive.