
Slacker's Docbook

2.6

Table of Contents

Overview	2
System Requirements	2
Directory Structure	2
Configuration Files for building	3
Motivation	5
Xml2Docbook	7
User's Guide	11
Element Reference	11
Conditions, and how they are used	12
Hyperlinks to documentation	14
The Python Classes	15
History and Version Notes	19
Bibliography	19
Index	19

Slacker's Docbook is yet another Docbook preprocessor. It is a document format, plus scripts, stylesheets and libraries for converting documents into standard Docbook/XML. Bulleted lists, conditional processing, subdocument and sourcecode example inclusions, easy incorporation of images, and a host of other commonly faced problems are solved by Slacker's Docbook. In particular, it can be used to develop multiple versions (such as a textbook and overhead slides) of a document from a single source. This system was designed for writing computer science courseware (tutorials, assignments, exams), so it well suited for writing books that contain many sourcecode examples.

Overview

This is a self-documenting document about how to write self-documenting documentation. It was written in XML, but you are probably reading it in XHTML or PDF, after it has gone through a number of transformations, including Docbook XML as an intermediate file format.

This project is hosted on Slackerdoc.Tigris.org [<http://slackerdoc.tigris.org/>]. You can gain access to file releases, as well as the subversion repository from there. You are welcome to join the project if you wish to contribute.

System Requirements

To use Slacker's Docbook to build the documentation, you need to install yourself:

- jdk 1.4 [<http://java.sun.com/j2se/1.4.2/download.html>] or later
- A full Installation of Python 2.3 [<http://www.python.org/>] (including distutils)

You also need to install these libraries, but a version of each is included with Slacker's Docbook for your convenience.

- Reportlab pyRXP [<http://www.reportlab.org/pyrxp.html>], a fast XML parser for python.
- A recent copy of docbook-xsl stylesheets [http://sourceforge.net/project/showfiles.php?group_id=21935&package_id=16608].

Finally, you need an xslt processor. According to Docbook-Xsl definitive guide [<http://www.sagehill.net/docbookxsl/XSLprocessors.html#XSLTprocessors>], the best xslt processor to use is `xsltproc`. The build scripts are written to use that. It comes standard with most Linux distros, and can be installed on Win32 systems via Cygwin's package installer as well.

If you do not wish to use `xsltproc`, this link above lists some other open source xslt processors which are suited for docbook processing.

Directory Structure

Here is a tour of the directory structure:

- `doc` - contains the source XML documents which were used to generate what you

are reading now.

- `bin` - contains some executable shell scripts for `ant`, as well as the python scripts that comprise Slacker's Docbook, the XML docbook generator used for the documentation you are reading right now.
- `lib` - Libraries needed by the `ant` build script or any of the classes it invokes. Also contains tarballs of the docbook stylesheets and `pyRXP`.
- `uml` - some UML diagrams I drew to describe the process of getting from XML to HTML/PDF.
- `xsl` - a docbook customization layer. `xsl`, `css`, `dtd` files all go here. The custom headers/footers can be found in `htmlstyle.xsl`.
- `icons` - Callout graphics, navigation graphics, used by the stylesheets.
- `xsd` - Contains the sourcecode for the W3C xml schema describing Slacker's Docbook.
- `www` - The web root, also containing generated docbook and HTML files, as well as a copy of many other files in the source tree.

Note

the svn repo has files under revision control in this directory, but that's only because the svn repo dir is used for <http://slackerdoc.tigris.org>. You can wipe out this directory and all files should be re-created there on the next rebuild.

Configuration Files for building

There is a file called `build.properties` in this project.:

`build.properties` contains variables used by `ant` and as well as the python XML preprocessor:

Example 1. `../build.properties`

```
slacker.dir=/home/ezust/workspace/slackerdoc
version=2.6
docbook.doc.url=http://www.docbook.org/tdg/en/html
docbook.dir=/usr/share/xml/docbook/stylesheet/nwalsh
docs.dir=doc
css.name=slacker.css
build-book.dir=${basedir}/www/pdf
output.pdf=article.pdf
```

```
condition=remark, textbook, html, dev, fromfile
```

These variables will be substituted for their values inside the XML documentation by using @keyname@ syntax, and they are replaced with their values in Ant build.xml with \${keyname} syntax. The important one is docbook.dir, which must point to the root of the unzipped docbook-xsl tarball.

Motivation

If you're like me, you learned HTML a long time ago, and were quite frustrated by its limitations as a documentation language.

Perhaps you also find yourself typing HTML tags all the time and linking to external webpages as well as internal document locations when you write text notes to yourself.

Slacker's Docbook consists of a preprocessor, some scripts, and some style sheets which extend Docbook XSL [<http://www.sagehill.net/docbookxsl/index.html>]. I developed Slacker v1 in Perl as I was learning Docbook myself. It is designed for coders like us who hate to do repetitive tasks and want to achieve the maximum possible re-use of every piece of text or sourcecode. Slacker v2 is a re-work in Python which is not backward compatible with Slacker v1.

By writing the source documents in Slacker, you can generate multiple renditions of it, in HTML, PDF, or other formats. And while it is not necessary to learn most of the Docbook language, you can incorporate Docbook tags at any time during your development (such as indexterm, glossaries, cross-references, bibliographies, etc).

Slacker's Docbook, therefore, is XML, and it is Docbook, but at the same time, *it looks a lot like XHTML!* If you already know a little HTML, you can start writing Slacker after learning the strict XML syntax rules and the meanings of a few basic Docbook tags [<http://www.docbook.org/tdg/en/html/part2.html>].

The build script in this package searches your “docs.dir” for all .xml documents and produces a docbook versions of each file. The subsequent transformations using the docbook stylesheets provide you a table of contents, list of code examples, and cross-reference capabilities.

You place no formatting or presentation info in your .xml file - only content and markup tags. The presentation styles are all handled by XSL and (if you are generating HTML) CSS stylesheets.

There are so many advantages to using Docbook/XML.

- You get to learn some free and very powerful XML tools.
- It enables you to maintain modular reusable documentation components, linking to and including other pieces of content, with even more power and flexibility than HTML.
- It is human-readable and machine-readable.
- Combined with a good source code management system such as CVS/SVN, it's incredibly easy to collaborate with others - you can work on different parts of the same document simultaneously.

- You can show others your work in progress without actually sending it to them (do your work in a `httpd`-accessible location).
- You can use any editor you want, and there are lots of free XML editors to choose from.
- Everything you write can be reused in other books or documents.
- The longer you use it, the more stuff you can reuse.
- Once your documentation is in this format, it can be converted easily to HTML, LaTeX, postscript, PDF (click here to see sample) [pdf/article.pdf], or plain text. Or any other format that comes up later.
- It generates the table of contents and index for you.
- It's customizability rivals LaTeX.
- There are lots of tools that can read Docbook as the input language.

Docbook is a little too cumbersome to compose documentation directly in - the tags are too verbose, and `xincludes` are difficult to get working. We need simpler HTML tags for the most commonly used elements, and convenient tags for the most common operations (inclusion, inserting images).

To get Docbook to handle conditional inclusion and included sourcecode formatting, the files need an extra preprocessing step anyway.

Slacker's Docbook produces standard Docbook/XML and permits the following:

- `bold` and `<i>italic</i>` tags just like XHTML
- `` `` listitem tags for `` unordered and `` ordered lists
-

```
You can use pre tags for
preformatted
output
```

- `<tt>` alternate font for computer output
- a convenient way of including XML documents
- a convenient way of including non-XML documents
- Comment-processing of included C++ and Java code, with `// start` and `//end`

comments.

- Comment-processing to translate multiline comments into callouts.
- Language-intelligent text formatting.
- Auto-Hyperlinking based on keywords, to HTML docs.
- Comment-processing for hash-comment languages like python, bash, and perl.

Xml2Docbook

`xml2docbook.py` is the script which generates standard Docbook from Slacker's Docbook. It is the first step in the build process, designated in `build.xml` as target name `docbookfiles`.

`xml2docbook.py` has the following usage:

- `[-p srcdir]` - the prefix to chop off when calculating relative pathnames.
- `[-d destdir]` - place generated docbook/dtd files (default is the current directory).
- `[-c condition(,condition)*]` - include conditional elements in the output.
- `file1 [file2, file3, ...]` - filenames to process

Typically, this step is run from inside an ant build script.

```
<target name="docbookfiles"
  description="generates Standard Docbook/xml from Slacker
  depends="prepare">
  <apply failonerror="true" executable="python"
    dir="${docs.dir}" dest="${build.dir}"
    relative="false">
    <arg value="${slacker.bin}/xml2docbook2.py" />
    <arg value="-prop" />
    <arg value="${propertyfile.local}" />

    <arg value="-c" />
    <arg value="${condition}" />
    <arg value="-d" />
    <arg value="${build.dir}" />
    <srcfile/>
    <fileset id="xmlinputs" dir="${docs.dir}" casesensitive=
```

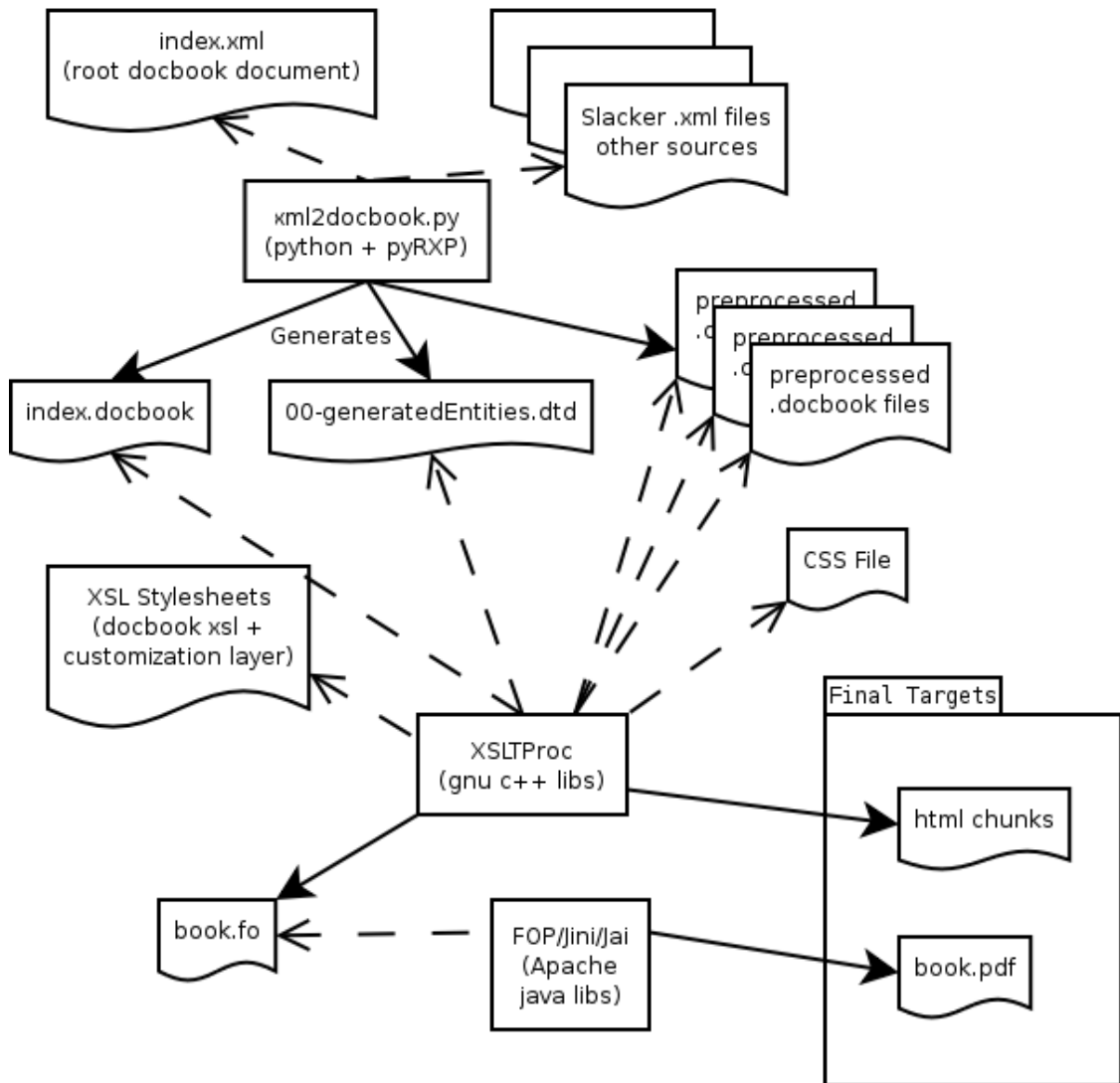
```

        <include name="**/*.*${input.extension}"/>
        <exclude name="build.xml"/>
    </fileset>
    <mapper type="glob" from="**.*${input.extension}" to="*.docbook"/>
</apply>
</target>

```

To create an html manual from the result, you need to run the generated root document through an xslt processor such as xsltproc, given the docbook stylesheets.

Figure 1. From XML to HTML or PDF



One document must be designated a “root” document for XSLTproc, its file must contain a **root=“true”** attribute. The other files are included directory or indirectly from the root document.

Example 2. index.xml

```
<article id="root-article" root="true"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation =
'http://slackerdoc.tigris.org/xsd/slackerdoc.xsd'
  >
<title>Slacker's Docbook </title>

<articleinfo>
  <date>@date@</date>
  <releaseinfo>@version@ </releaseinfo>
</articleinfo>

<abstract> <p> Slacker's Docbook is yet another Docbook preprocessor.
It is a document format, plus scripts, stylesheets and libraries for
converting documents into standard Docbook/XML. Bulleted lists,
conditional processing, subdocument and sourcecode example inclusions,
easy incorporation of images, and a host of other commonly faced
problems are solved by Slacker's Docbook. In particular, it can be
used to develop multiple versions (such as a textbook and overhead
slides) of a document from a single source. This system was designed
for writing computer science courseware (tutorials, assignments,
exams), so it well suited for writing books that contain many
sourcecode examples. </p>
</abstract>

<include src="intro.xml" mode="xml" />
<include src="motivation.xml" mode="xml" />
<include src="styletest.xml" mode="xml" />
<include src="userguide.xml" mode="xml" />
<include src="pythonclasses.xml" mode="xml" />
<include src="topics/specialtopics.xml" mode="xml" c="junk" />
<include src="history.xml" mode="xml" />
<include src="bibliography.xml" mode="xml" />

<index id="keywordindex" />

</article>
```

The external files encountered by `xml2docbook` get entity definitions in a generated file called `00-generatedEntities.dtd`. Any root document that processed by `xml2docbook.py` will refer to this DTD at the top:

Example 3. ../www/00-generatedEntities.dtd

```
<!ENTITY uml.xml2html.png "uml/xml2html.png" >
<!ENTITY www.history.docbook SYSTEM
"/home/ezust/workspace/slackerdoc/www/history.docbook" >
<!ENTITY www.styletest.docbook SYSTEM
"/home/ezust/workspace/slackerdoc/www/styletest.docbook" >
<!ENTITY www.intro.docbook SYSTEM
"/home/ezust/workspace/slackerdoc/www/intro.docbook" >
<!ENTITY www.bibliography.docbook SYSTEM
"/home/ezust/workspace/slackerdoc/www/bibliography.docbook" >
<!ENTITY www.userguide.docbook SYSTEM
"/home/ezust/workspace/slackerdoc/www/userguide.docbook" >
<!ENTITY www.motivation.docbook SYSTEM
"/home/ezust/workspace/slackerdoc/www/motivation.docbook" >
<!ENTITY www.pythonclasses.docbook SYSTEM
"/home/ezust/workspace/slackerdoc/www/pythonclasses.docbook" >
<!ENTITY www.xml2docbook.docbook SYSTEM
"/home/ezust/workspace/slackerdoc/www/xml2docbook.docbook" >
<!ENTITY uml.textproc.png "uml/textproc.png" >
<!ENTITY uml.entitymgr.png "uml/entitymgr.png" >
<!ENTITY uml.nodemapper.png "uml/nodemapper.png" >
```

User's Guide

The formal grammar of Slacker's Docbook is written in W3C XML Schema Description language. It is based on the unofficial Docbook 4.4 xsd.

To specify that a document uses this grammar, place the following attributes in the root element. For example, here is the root element of this section:

```
<section id="userguide"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation='http://slackerdoc.tigris.org/xsd/s
>
<title> User's Guide </title>
```

Element Reference

Table 1. Slacker Docbook Elements

Element	Meaning
<code></code>	includes an image, inserting a <code><i ma ge da ta <mediaobject><imageobject>></code> in its place, referring to an entity reference that is auto-generated. You can supply a relative <i>pathname</i> . In such a case, the actual image will be copied into the destination directory, to ensure it is addressible. Other attributes, such as width, height, scale, and scalefit are passed to the <code><imagedata></code> .
<code><include src="pathname" mode="modestr"></code>	<code>src</code> is assumed to be a relative path from this document. <i>modestr</i> can be: <ul style="list-style-type: none"> • <code>cpp</code> - included as code example with C++/Java style comments, with some

Element	Meaning
	<p>callout processing. (no linewrap)</p> <ul style="list-style-type: none"> • <code>code</code> - included as example python/perl/shellscript comments (no linewrap) • <code>txt</code> or <code>text</code> - included as an example text file with linewrap. • <code>tex</code> - included as a LaTeX file (? experimental) • <code>xml</code> - Replaced with an entity reference to XML file - does not process included document. <p>If mode is not “xml”, this tag includes a source file at preprocess time, and inserts its contents inside an <code><example><programlisting></code>. Since each <code><example></code> should have an <code>id</code> attribute, it is recommended you also supply an <code>id=“idAttr”</code> to the <code><include></code>, which will get passed onto the generated <code><example></code>.</p>
<code></code>	Equivalent to <code><listitem><para></code>
<code></code>	<code><orderedlist></code>
<code><p></code>	<code><para></code>
<code><pb></code>	“parabullet” - rendered as an <code><orderedlist></code> when “slides” condition is true. Place <code></code> around each major idea or sentence. When slides is not true, a <code><pb></code> is rendered as a single <code><para></code> , and all the <code></code> tags inside the <code><pb></code> are removed.
<code></code>	<code><itemizedlist></code>

Conditions, and how they are used

Any element can have a `condition` (sometimes shortened to just a `c`) attribute. For example:

```
<pb>
<li> This shows up in in all versions. </li>
<li condition="slides"> This is in the slides </li>
<li c="textbook"> This is a much longer sentence for the textbook. </li>
</pb>
```

To process a document, you can select multiple conditional values as separate `-c` options, or as strings separated by commas (no spaces) after a single `-c`.

```
python xml2docbook.py -c remark -c slides,html
python xml2docbook.py -c all
```

The first executes the script with conditions `remark`, `slides`, and `html` switched on, and all other conditions switched off. The second switches “all” conditions on ¹

Condition attributes are used in a variety of ways. You can add new conditions to suit your particular document. Only some of them have special meaning inside the python scripts, and those meanings are listed here.

Currently used Conditions

<code>html</code>	For HTML output
<code>pdf</code>	pdf output
<code>ps</code>	ps output - converts as needed, all png images to eps and substitutes the extensions of all <code> src</code> attributes.
<code>fromfile</code>	displays the red “fromfile” information at the top of each page.
<code>slides</code>	renders all <code><pb></code> tags as <code><itemizedlist></code>
<code>textbook</code>	renders all <code><pb></code> tags as <code><para></code> , and ignores the enclosed <code></code> tags.
<code>remark</code>	output in red/italic
<code>todo</code>	like <code>remark</code>
<code>solution</code>	solutions guide

¹To see exactly which conditions are switched on, look at `slackerxp.ConditionHandler`.

`instructor` `instructor's notes`

Hyperlinks to documentation

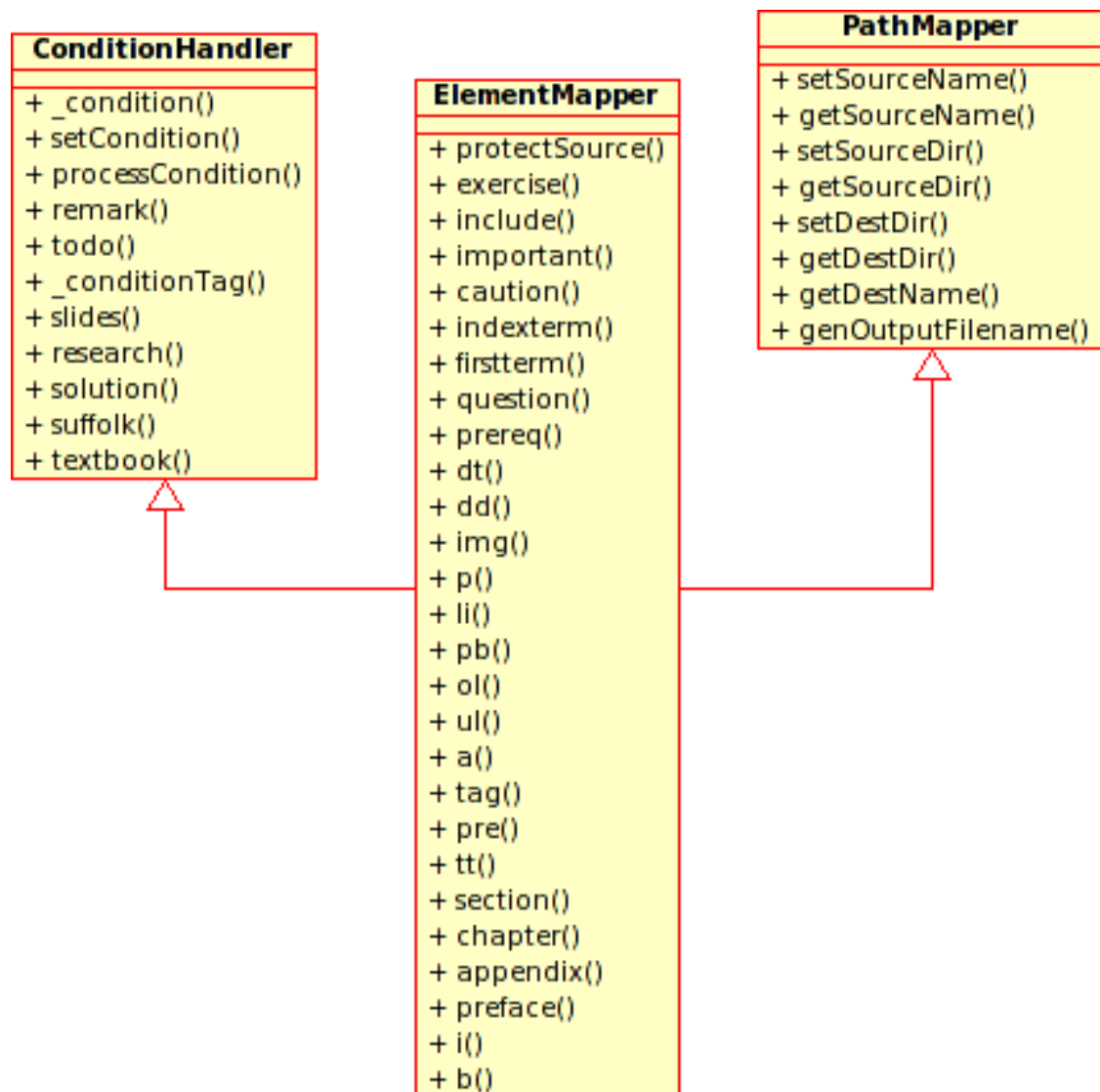
If you have a list of classnames, or element names, and can write a function that, given a classname, returns a URL, then Slacker's Docbook can add hyperlinks to API docs for each of those names.

The Python Classes

The preprocessor which generates docbook/xml is written in Python and uses the pyRXP XML parser, as well as a few recipes from the Python Cookbook. It is designed to be very modular and extensible, but makes heavy use of (multiple) inheritance. This section will demystify the way the classes fit together and help you more easily find the code of interest relating to the feature you may want to customize.

We have divided up the classes into two categories. First the main classes that do straight XML element-to-element mapping:

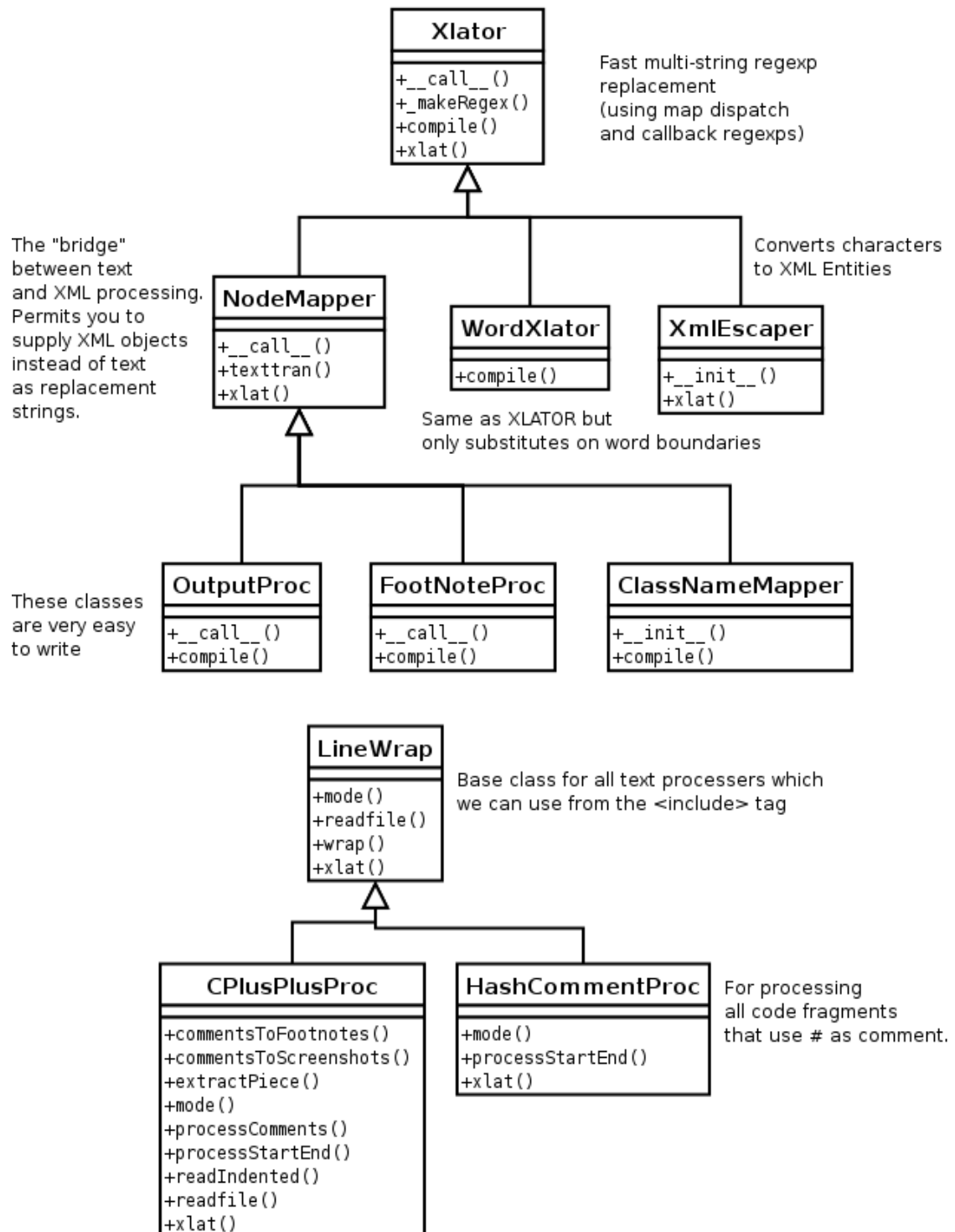
Figure 2. Classes for mapping XML elements and paths



As you can see, the ElementMapper class has a method for each element that we have added in Slacker's Docbook. The method returns an XML node which gets swapped into the tree.

Figure 3. Text Processing Classes

Xlator and related classes for text processing



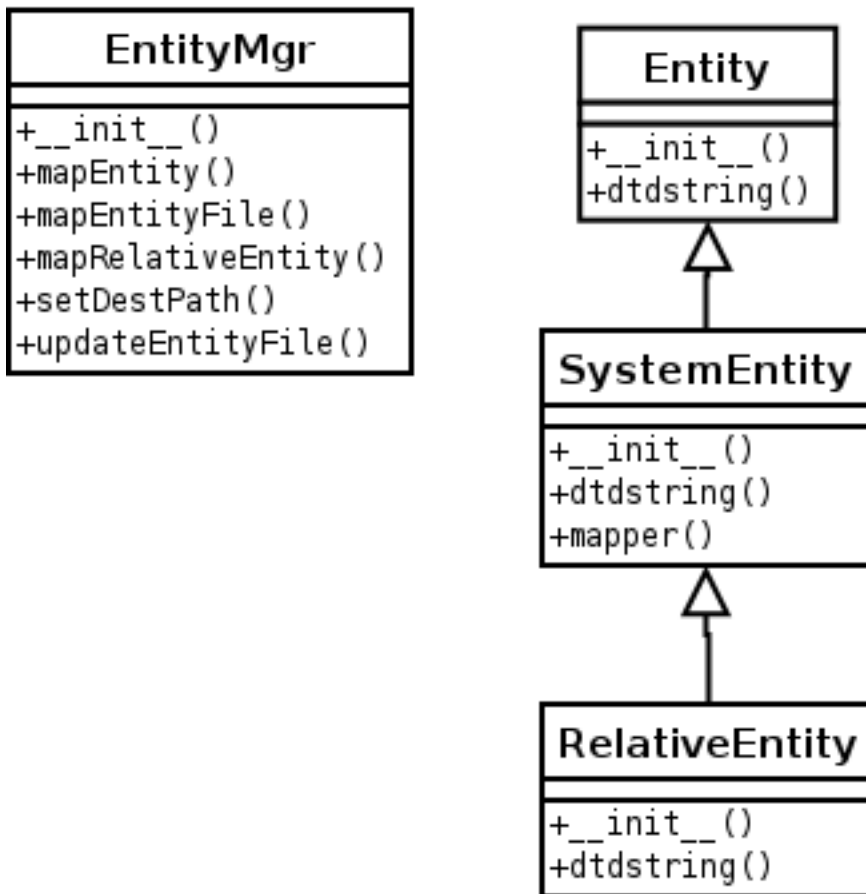
This diagram shows the text processing routines (which accept strings rather than xml nodes as the input).

Notice every class has an `xlat()` method. All `NodeMapper`-derived `xlat` methods return XML objects rather than simple strings. All `NodeMapper`-derived classes can also accept XML objects or simple strings as input to `xlat()`

Finally, we have a couple of classes for managing the OO-generatedEntities file:

Figure 4. Classes for managing DTD entities

Used by `xml2docbook` to load and store XML DTD Entities



History and Version Notes

Version 2.6 - June 27, 2006 . Formal W3C xsd grammar available.

Version 2.41 - June 10, 2006 . Separated from project “local” and added to tigris.org.

Version 2.0pre5 - July 8, 2004 . Many methods redesigned and debugged - especially in the docbook processing department. Descriptions of new features will be added to the documentation shortly.

Version 1.9 - June 12, 2004 . Replaced the perl script with a series of more general-purpose python classes, which are not 100% backward compatible - some changes were made to the grammar.

Version 1.02 - March 18, 2004 . Currently uses Ant to build everything.

Bibliography

Docbook References

[docbook] *Docbook: The Definitive Guide*. Norman Walsh. 2006. O'reilly Associates [http://www.docbook.org/tdg/en/html/docbook.html] .

[docbookxsl] *Docbook XSL: The Complete Guide*. Bob Stayton. 2005. SageHill Enterprises [http://www.sagehill.net/docbookxsl/] .

Books

[pycook] *Python Cookbook*. Alex Martelli, Anna Martelli Ravenscroft, and David Ascher. March 2005. O'Reilly. 0-596-0797-3.

[Gof95] *Design Patterns* . Elements of Reusable Object-Oriented Software . Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. Addison-Wesley. 0-201-63361-2.

[Fowler04] *UML Distilled*. Third Edition. Martin Fowler. 2004. Addison Wesley. 0-321-19368-7 .

Index

D

Docbook/XSL, 5

H

HTML, 5

X

XHTML, 5

XML, 5