

# Practical XML

November 12, 2006

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Installation</b>	<b>2</b>
<b>3</b>	<b>Libraries</b>	<b>2</b>
<b>4</b>	<b>Applications</b>	<b>2</b>
4.1	publishxml . . . . .	2
4.2	generatedtd . . . . .	2
4.3	validatexml . . . . .	3
4.4	processtopics . . . . .	3
4.5	processjournal . . . . .	3
4.6	processrecords . . . . .	3
4.7	processtasks . . . . .	3
4.8	tutorial1 . . . . .	3
<b>5</b>	<b>Documents</b>	<b>3</b>
5.1	template, ampmmanual, perltour, practicalxml, pcsdevelopment, rffilesystem . . . . .	4
5.2	pcstopics . . . . .	4
5.3	pcsjournal, pcsrecords, pcstasks . . . . .	4
<b>6</b>	<b>DocBook Extensions</b>	<b>4</b>
6.1	File Inclusion . . . . .	4
6.2	Maths . . . . .	4

## 1 Introduction

PRELIMINARY DOCUMENTATION.

Practical XML is a project of John Storrs, Laboratory for Micro Enterprise (LME), aimed at promoting the wider use of xml, particularly for non-Web applications. The open-source Practical XML package contains a parser, documentation generator, dtd generator, xml validator, with sample applications and associated xml documents. The applications include a simple tutorial which shows how easy it is to build and process an in-memory tree representing an xml document. This is a 'fusion community'

version of the project, which includes example documents written at Culham Laboratory as part of my work.

XML is a curate's egg of a standard, and the Practical XML project currently only handles the good bits. It focuses on the definition of hierarchical data structures using element and attribute markup. It handles the insertion of standard entities, but not user-defined entity definitions. Though it only covers a subset of the xml standard, the Practical XML toolchain is applicable to a wide range of applications as the examples show.

The project software is written in Perl, an ideal language for text processing. The parser and system tools make extensive use of Perl regular expressions and other useful features of the language.

The package files are organised in 3 subdirectories: lib, containing the parser libraries, app containing system and demo applications, and doc containing associated xml documents, all discussed below.

## 2 Installation

The code in this package is copyright under the Free Software Foundation General Public License by John Storrs, Laboratory for Micro Enterprise.

Install the tarball anywhere. Set environment variable PXML\_HOME to point to the installation directory. Perl 5.8 or later is required. The pdf generator uses Latex as its output engine, so one of the current Tex/Latex packages is needed. The ImageMagick convert-6.0.0 or higher is needed for graphics conversion, and the graphviz package is used in a DocBook extension. The software tool requirements should be satisfied by any recent Linux distribution. Once you are set up you should be able to process xml documents in the doc area, by running make in the xml subdirectories. The results are placed in the docbook, html and pdf subdirectories. Expect some error messages: this is a beta release.

## 3 Libraries

The xml parser is in two perl modules: lib/LME/Xmlparser.pm and lib/LME/XmlNode.pm. It is an incremental parser, allowing memory-efficient processing of very large xml files.

## 4 Applications

The app subdirectory contains system tools publishxml, generatedtd and validatedtd, some demo applications process\*, and a coding tutorial. Example xml documents processed by the demo applications are in the doc subdirectory.

### 4.1 publishxml

This is a documentation generator for a core subset of DocBook, with some useful extensions. It currently generates html and pdf output. It has a clean, extensible design, and is much faster than some other open source Docbook tools.

### 4.2 generatedtd

This generates a dtd from a collection of xml documents expressing a single schema.

### 4.3 `validatexml`

This validates an xml document against a dtd.

### 4.4 `processttopics`

`processjournal`, `processtasks` and `processrecords` are topic-based applications illustrating the transformation of a user-defined xml vocabulary to DocBook. A topic tree is defined using a simple xml vocabulary. This can be processed by `processttopics` to generate topic tree documentation. Other documents can structure their contents implicitly by reference to this tree.

### 4.5 `processjournal`

This processes a topic-based journal written in a simple vocabulary with embedded DocBook.

### 4.6 `processrecords`

This processes a topic-based structured document written in a simple vocabulary with embedded DocBook. PDF output is generated recursively for all levels in the topic hierarchy, making it easy to view any sub-tree of the document

### 4.7 `processtasks`

This processes a topic-based project task description, generating project management documentation.

### 4.8 `tutorial1`

This shows how easy it is to build and process an in-memory tree representation of an xml file. Try it with your own application-specific vocabulary.

## 5 Documents

The `doc` subdirectory contains example documents associated with applications described above. `ampmmanual`, `practicalxml` (this document), `pcsdevelopment`, and `rffilesystem` are straight DocBook documents, processed directly by `publishxml`. The other documents are first processed by the appropriate application to transform them to DocBook, then that is processed by `publishxml`.

Practical XML documents occupy a directory sub-tree. The `xml` subdirectory contains the source, the `graphics` and `imports` subdirectories contain any images and other imported files, the `docbook` subdirectory contains generated DocBook if any, and the `html` and `pdf` subdirectories contain the viewable representations of the document after processing. In all cases, to process the document run `make` in the `xml` subdirectory. View the `html` by opening `html/index.html` in a web browser. View the `pdf` output(s) using `acroread` in the `pdf` directory. Potentially many `pdfs` are generated for topic-based documents; The `pdf` containing the full document starts with a lower case letter. A default `html` stylesheet is provided in the `import` subdirectories.

In real life, making an xml document would usually result in it being published to a web site. Most `makefiles` contain a target for this, though it is not called as part of the normal `make` process in the released package. Any links in the generated `html` to the `pdfs` will not work as a result.

## 5.1 `template`, `ampmanual`, `perltour`, `practicalxml`, `pcsdevelopment`, `rffilesystem`

These DocBook documents provide examples of many common DocBook features. You can use the `template` as a starting point for your own Docbook documents. `Perltour` is a short Perl tutorial which may be of interest, as it highlights some perl features on which the Practical XML project depends.

## 5.2 `pcstopics`

This defines the topic tree on which `pcsjournal`, `pcsrecords` and `pcstasks` are based.

## 5.3 `pcsjournal`, `pcsrecords`, `pcstasks`

These topic-based documents use application-specific markup with embedded DocBook.

# 6 DocBook Extensions

DocBook extensions provided in Practical XML include:

## 6.1 File Inclusion

Simple xml file inclusion is achieved like this:

```
<include file="introduction.xml"/>
<include file="developmenttools.xml"/>
<include file="hardware.xml"/>
<include file="firmware.xml"/>
<include file="software.xml"/>
<include file="hardwaretesting.xml"/>
<include file="softwareupdates.xml"/>
```

## 6.2 Maths

Latex maths can be embedded in DocBook documents in a `math` element. This is an experimental extension, which currently only works for pdf output. Here are some examples:

```
<math>\left[ 2\sum_{i=1}^n a_i \int_a^b f_i(x)g_i(x)\,dx \right]</math>
```

produces this:

$$2 \sum_{i=1}^n a_i \int_a^b f_i(x)g_i(x) dx$$

and this:

```
<math>$$$ 2\sum_{i=1}^n a_i \int_a^b f_i(x)g_i(x)\,dx $$$</math>
```

produces this:

$$2 \sum_{i=1}^n a_i \int_a^b f_i(x)g_i(x) dx$$

Here is some in-line maths:  $x^{2n-1}$  and  $\sqrt[3]{8}$ .