

How To Write A Minimal L^AT_EX XML Binding

Hang Yuan, Jinbo Zhang, Michael Kohlhase
Computer Science, Jacobs University Bremen

May 14, 2015

Abstract

Writing L^AT_EX XML bindings for L^AT_EX classes is more difficult than bindings for L^AT_EX packages, since the class bindings need to supply a lot of the processing infrastructure. Most L^AT_EX classes inherit that from `article.cls`, and correspondingly most L^AT_EX XML class bindings inherit from `article.cls.ltxml`, but some L^AT_EX classes are standalone.

We explain how to write a L^AT_EX XML binding for a L^AT_EX class from scratch based on a minimal example. This note intends to complement the L^AT_EX XML documentation help beginners get started.

Contents

1	Introduction	2
2	Using LaTeXXML	2
2.1	Things We Need	2
3	The mockDoc Format	3
3.1	A minimal Document Format	3
3.2	The RelaxNG Schema	4
4	How to Create a LaTeXXML Binding	4
4.1	Basic structure	5
4.2	Configure namespace	5
4.3	Linebreaks	5
4.4	Sectioning	5
4.5	The Document Environment	5
4.6	Auto-opening for Paragraphs	6
5	Postprocessing for Web Workflow	6
5.1	XSL Stylesheet	6
6	Conclusion	6
A	Appendix	9
A.1	The mockDoc Class	9
A.2	The mockDoc Class Binding	10
A.3	mockDoc RelaxNG schema	11
A.4	XSL stylesheet	12
A.5	Generated HTML	13
B	A Makefile for Automation	14

1 Introduction

L^AT_EX has been widely used as a document processor among scholars, especially when one needs to use large quantities of mathematical representations. L^AT_EX is also a good choice for those who are meticulous about typographical quality of documents.

As a page formatting tool, the primary output format of the L^AT_EX formatter is PDF; which – with fixed page formats and limited interaction features – is only partially suited for usage in the modern web. The DLMF (Digital Library of Mathematical Functions) developed L^AT_EX_{ML}, a flexible, semantics-preserving L^AT_EX to XML converter to fix this.

However, for every L^AT_EX class and package used in a document L^AT_EX_{ML} we need a **L^AT_EX_{ML} binding** – a configuration file that specifies the XML counterpart of the L^AT_EX command sequences provided by the respective class or package.

Even though the L^AT_EX_{ML} distribution provides bindings for the most commonly used classes and packages, the availability of bindings is still the most severe bottleneck for L^AT_EX_{ML}. The L^AT_EX_{ML} documentation [Milb] is mostly written for developers and quite impenetrable for beginners.

To encourage binding development, this how-to tutorial goes through the steps and pitfalls of creating a L^AT_EX_{ML} class binding from scratch. This tutorial does not cover advanced topics related to L^AT_EX_{ML}, for which we refer to the L^AT_EX_{ML} manual [Milb].

We have developed a minimal document class `mockDoc` as an example for this how-to and will go through in detail. All necessary files (and the development version of this tutorial) are available from [Moc], but are also included in the appendix of this document for reference.

This how-to tutorial is structured as follows: section 2 briefly reviews L^AT_EX_{ML} workflows and the files involved; section 3 introduces a minimal T_EX and its schema; section 4 gives a basic view of how to write L^AT_EX_{ML} binding; section 5 talks about postprocessing for web workflow; section 6 concludes the tutorial.

2 Using LaTeXML

In this tutorial we assume a working installation of L^AT_EX_{ML} – see [Mila] for instructions – on a Unix-like system (Linux, Mac OS, etc.).

Given that, we use the command

```
latexmlc mockDoc.tex --format=XML --destination=mockDoc.xml --log=mockDoc.xml.log
latexmlpost --stylesheet=mockDoc.xsl --destination=mockDoc.html mockDoc.xml
```

for converting `mockDoc.tex` into `mockDoc.xml` and `mockDoc.xml` into `mockDoc.html`

The conversion from L^AT_EX to XML is processed by L^AT_EX_{ML}. Basically L^AT_EX_{ML} maps the L^AT_EX markups to the XML markups, more specifically: macros, primitives and constructors. The post-processing mechanism such as conversion to HTML and XHTML is done by passing the documents through the post-processing filter modules.

2.1 Things We Need

Source Here we use `mockDoc.tex` as a minimal example see section 3.1.

L^AT_EX class We provide a L^AT_EX class `mockDoc.cls` for reference; sometimes it is useful to generate PDF for proofreading. The normal situation in developing L^AT_EX_{ML} bindings is that the class/package pre-exists. This file won't be illustrated in this tutorial - see appendix A.1 for source.

L^AT_EX_{ML} binding The core issue of this tutorial. We use `mockDoc.cls.ltxml` – see section 4 for a step-by-step explanation and appendix A.2 for the end result.

RelaxNG schema L^AT_EX_{ML} needs a RelaxNG schema to infer the output structure. We supply it in compact form `mockDoc.rnc` - see appendix A.3 for source and section 3.2 for explanation,

which can assist the XML conversion, and we note that \LaTeX ML needs `mockDoc.rng` via `trang.jar`. The reason for writing `mockDoc.rnc` instead of `mockDoc.rng` is that, `mockDoc.rnc` is much shorter and easier to maintain.

XSL stylesheet To customize our output in the web workflow, we can provide \LaTeX ML with `mockDoc.xsl`, showing a general idea of how postprocessing works - see appendix A.5 for the effects and section 5 for a detailed description.

After we have finished writing all the documents above, we should be able to see the converted XML file for `mockDoc.tex`. In the following chapters we will explain how to construct `mockDoc.rnc` and `mockDoc.cls.ltxml`

These workflows can be automated via a Unix `makefile` (see appendix B), which re-generates everything when source files have changed. Then only need to issue the command:

```
make
```

3 The mockDoc Format

3.1 A minimal Document Format

Actually our `mockDoc` format is probably the smallest one in the world. It is only intended for this tutorial.

The \LaTeX class only provides one environment: `document` and four macros: `\section`, `\subsection`, `\paragraph`, and `\newline`. A minimal example would be

Listing 1: A Minimal \LaTeX Document

```
\documentclass{mockDoc}
\begin{document}
\section{First section}
Here is some text.
\section{Second section}
\subsection{Subsection I}
\paragraph{Paragraph 1}
Here is some text.
\paragraph{Paragraph 2}
Line 1 \newline
We try to test line break and paragraph II here.
\subsection{Subsection II}
\paragraph{Paragraph 1}
We try to test subsections II here.
\end{document}
%%% Local Variables:
%%% mode: latex
%%% TeX-master: t
%%% End:
```

We want to use this document class for generating XML document, which has five elements `document`, `\section`, `\subsection`, `\paragraph`, and `\newline`. The XML document corresponding to the \LaTeX document from Listing 1 is

Listing 2: The Generated XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<?latexml searchpaths="/Users/yuancalvin/xmltrans/howto"?>
<?latexml class="mockDoc"?>
<?latexml RelaxNGSchema="mockDoc.rng"?>
<mock:document xmlns:mock="https://kwarc.info/projects/mockDoc">
  <mock:section>
    <mock:title>First section</mock:title>
    <mock:p>Here is some text.
  </mock:p>
</mock:section>
```

```

<mock:section>
  <mock:title>Second section</mock:title>
  <mock:subsection>
    <mock:title>Subsection I</mock:title>
    <mock:paragraph>
      <mock:title>Paragraph 1</mock:title>
      <mock:p>
Here is some text.
</mock:p>
      </mock:paragraph>
    <mock:paragraph>
      <mock:title>Paragraph 2</mock:title>
      <mock:p>
Line 1 <mock:break/>We try to test line break and paragraph II here.
</mock:p>
      </mock:paragraph>
    </mock:subsection>
  <mock:subsection>
    <mock:title>Subsection II</mock:title>
    <mock:paragraph>
      <mock:title>Paragraph 1</mock:title>
      <mock:p>
We try to test subsections II here.
</mock:p>
    </mock:paragraph>
  </mock:subsection>
</mock:section>
</mock:document>

```

Note the typical format-specific differences between the content-oriented \LaTeX and more data-oriented XML formats. The sectioning is conveyed by macros in \LaTeX – only giving the start cues (here the numbered section headings) – whereas the XML has start and end tags. In addition, \LaTeX markup is less strict than XML markup, meaning \LaTeX more suitable for somewhat messy human readable documents, whilst in XML not everything is allowed.

3.2 The RelaxNG Schema

Schema is a crucial document that decides how `mockDoc.xml` is constructed. When one is creating his own schema, the [Rel] and [Xml] are two good documentations to get started. One convenient approach to test this is to create our expected `mockDoc_sample.xml` by hand, according to our `mockDoc.tex`, then compare `mockDoc_sample.xml` with the generated `mockDoc.xml`. We can easily accomplish this by using *emacs nxml mode* [Ema], in which we have the freedom to write our expected `mockDoc.xml`, while validating our `mockDoc.xml` at the same time. If validation fails, we can see the error message instantly, such that we can debug our `mockDoc.xml` or schema accordingly.

In our `mockDoc.rnc`:

```

document = element document {p*, section*}
section = element section {title,p*,subsection*}

```

We can easily see that, under a `document`, there can be either `p` or `section`, and under a `section` there can be a `title` followed by either `p` or `subsection`. This is due to the first section in `mockDoc.tex`:

```

\section{First section}
Here is some text.

```

has no subsection but text, and in the other sections, there are subsections. We need to consider all possible hierarchies of our elements in the schema.

4 How to Create a \LaTeX XML Binding

We now come to the central part of this tutorial: writing the \LaTeX XML binding itself. Generally, a \LaTeX XML binding file is a Perl module – and therefore underlies Perl syntax, but special high-level commands simplify expressing the \LaTeX -to-XML relation.

4.1 Basic structure

Since \LaTeX binding is a perl module, we need to initialise a binding file by adding the followings in the beginning of `mockDoc.cls.ltxml`:

```
package LaTeXXML::Package::Pool;
use strict;
use LaTeXXML::Package;
use warnings;
```

At the end of `mockDoc.cls.ltxml`, don't forget to include

```
1;
```

to make sure that perl works properly.

4.2 Configure namespace

With:

```
RegisterNamespace('mock'=>"https://kwarc.info/projects/mockDoc");
RelaxNGSchema("mockDoc.rng", 'mock'=>"https://kwarc.info/projects/mockDoc");
```

We declare the namespace associated the prefix `mock` with the namespace, and thus we can use the prefix when defining new macros to avoid name conflicts. The second lines tells \LaTeX XML that the generated XML should fit in our schema.

4.3 Linebreaks

The next task is to teach \LaTeX XML new commands used in `mockDoc.tex`. Here is an example:

```
DefConstructor('\newline', "<mock:break/>");
```

This line defines how \LaTeX XML interprets `\newline`, as we see, \LaTeX XML will translate `\newline` to `<mock:break/>` in `mockDoc.xml`.

4.4 Sectioning

When dealing with `section`, things get a little tricky, with:

```
DefConstructor('\section{ }', "<mock:section><mock:title>#1</mock:title>");
```

we already define `\section`. But, think about the closing tags. In `mockDoc.tex`, we declare where the `\section` starts and where the next `\section` starts, nevertheless, we never write something like “Now close this section”. This schema file tells \LaTeX XML what the structure of our document, and with:

```
Tag('mock:section', autoClose=>1);
```

\LaTeX XML will close the section tags (i.e, adding `</mock:section>`) whenever needed.

4.5 The Document Environment

We may think something like:

```
DefEnvironment('{document}', "<mock:document>#body</mock:document>");
```

is enough for defining `document` environment. We can try it, but we will find that all spaces disappear. What we actually write in `mockDoc.cls.ltxml` is:

```
DefEnvironment('{document}', "<mock:document>#body</mock:document>", beforeDigest
=> sub { AssignValue(inPreamble => 0); });
```

This code can prevent the error mentioned before, however, the mechanism of the `beforeDigest` part is out of our discussion in this tutorial.

For an environment, we don't need care about auto-closing, since an environment is always like

```
\begin{*environment-name*}
content...
\end{*environment-name*}
```

where `\end{*environment-name*}` will indicate where to close the tags.

4.6 Auto-opening for Paragraphs

Since we also want to write some texts directly under `document`, without any `section`. At this circumstance, we need auto-open for `p`:

```
Tag('mock:p', autoOpen=>1);
```

which will surround such texts.

We now have a complete set of tiles to generate our XML file. Simply by using `makefile`, we should be able to see the generated `mockDoc.xml` in our current directory. It should be something similar to the expected `mockDoc\sample.xml`.

5 Postprocessing for Web Workflow

After we obtain `mockDoc.xml`, we can further utilize the power of `LATEXML` to convert it into some other useful formats such as HTML, HTML5 and XHTML. `LATEXML` by default provides us with stylesheets for this conversion, however we are given the freedom to customize this process by creating our own XSL and CSS stylesheets.

5.1 XSL Stylesheet

Similar to how we create RelaxNG schema, in `mockDoc.xsl`

```
<xsl:template match="mock:section|mock:subsection|mock:paragraph">
  <xsl:apply-templates/>
</xsl:template>
```

In case of the section template, we let `LATEXML` generate a section in HTML and then apply the templates for the macros that appear in `mock:section` from our `mockDoc.xml` file. Our `mockDoc.xsl` should be consistent with our `mockDoc.rnc` because they both define the structure of our documents. As we can see from the section definition in our `mockDoc.rnc`, they both contain: `title`, `p` and `subsection` elements.

```
section = element section {title,p*,subsection*}
```

Also, we want to allow each children of a class to be processed whenever a template matches, to deal with situations where we have a subsection, that contains more than one paragraph can exist, so we use

```
<xsl:apply-templates/>
```

to check template matches.

6 Conclusion

This note explains how to write a `LATEXML` binding for a `LATEX` class from scratch based on a minimal example. It serves as a complement the `LATEXML` documentation to help help beginners get started.

The case of bindings for `LATEX` classes is special, since we need to get familiar with all the necessary components that are required for different processing steps. For instance if we want to customize the conversion from XML to HTML, we need to know about `LATEXML` schema, XSLT and XHTML. `LATEXML` allows a large degree of customizations which maximizes the convenience of format conversion, particularly the conversion from `tex` to `xml`, as our prefer to use `TEX` for

production and XML for delivery. For a web workflow, where the ultimate goal is to generate HTML5, writing a document class from scratch may not be the most common workflow, since the majority of document classes in \LaTeX is derived in some way from `article.cls` and therefore the \LaTeX XML bindings can inherit the from `article.cls.ltxml`, but in some cases we want to use \LaTeX XML to generate other XML-based format. There we need the techniques in this tutorial. Examples are generating OMDoc from \LaTeX [Kohlhase:ulsmf08; URL:sTeX:github], `letter.cls`, `amsart.cls`, etc. – actually there are surprisingly few classes that do not inherit from `article.cls`.

References

- [Ema] *Emacs nxml*. URL: <http://www.emacswiki.org/emacs/NxmlMode> (visited on 03/12/2015).
- [Mila] Bruce R. Miller. *Get LaTeXXML*. URL: <http://dlmf.nist.gov/LaTeXML/get.html> (visited on 02/22/2015).
- [Milb] Bruce R. Miller. *LaTeXML The Manual*. URL: <http://dlmf.nist.gov/LaTeXML/manual.pdf> (visited on 02/22/2015).
- [Moc] *mockDoc, a minimal LaTeXML class binding and HowTo*. URL: <https://github.com/angerhang/mockDoc> (visited on 03/07/2015).
- [Rel] *RelaxNG Tutorial*. URL: <http://relaxng.org/compact-tutorial-20030326.html> (visited on 03/12/2015).
- [Xml] *XML Tutorial*. URL: <http://www.w3schools.com/xml/> (visited on 03/12/2015).

A Appendix

A.1 The mockDoc Class

```
% File: doc.cls
% Author: Jinbo Zhang
% Date: 3 Feb, 2015

\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{mockDoc}
\RequirePackage{ifthen}

\renewcommand\normalsize{\fontsize{10pt}{12pt}\selectfont}
\setlength{\textwidth}{6.5in}
\setlength{\textheight}{8in}

\newcommand\large{\@setfontsize\large\@xiipt{14}}
\newcommand\Large{\@setfontsize\Large\@xivpt{18}}

% define \paragraph
\newcommand{\paragraph}[1]{
  \newline\newline
  \bfseries #1
  \normalfont
}

% define \section
\newcounter{SectionCount}
\newcommand{\section}[1]{
  \ifthenelse{\value{SectionCount}=0}{\{\}\newline\newline\newline}
  \Large
  \stepcounter{SectionCount}
  \noindent\bfseries\arabic{SectionCount}\hspace{4mm} #1
  \normalfont
  \newline\newline
}

% define \subsection
\newcounter{SubCount}[SectionCount]
\newcommand{\subsection}[1]{
  \ifthenelse{\value{SubCount}=0}{\{\}\newline\newline}
  \large
  \stepcounter{SubCount}
  \bfseries\arabic{SectionCount}.\arabic{SubCount}\hspace{3mm} #1
  \normalfont
}

\endinput
```

A.2 The mockDoc Class Binding

```
package LaTeXML::Package::Pool;
use strict;
use LaTeXML::Package;
use warnings;

#Document Structure
RegisterNamespace('mock'=>"https://kwarc.info/projects/mockDoc");
RelaxNGSchema("mockDoc.rng", 'mock'=>"https://kwarc.info/projects/mockDoc");

# -----
DefEnvironment('{document}', "<mock:document>#body</mock:document>", beforeDigest
=> sub { AssignValue(inPreamble => 0); });
DefConstructor('\section{ }', "<mock:section><mock:title>#1</mock:title>");
DefConstructor('\subsection{ }', "<mock:subsection><mock:title>#1</mock:title>");
DefConstructor('\paragraph{ }', "<mock:paragraph><mock:title>#1</mock:title><mock:p>
");
DefConstructor('\newline', "<mock:break/>");

#autoClose
Tag('mock:paragraph', autoClose=>1);
Tag('mock:section', autoClose=>1);
Tag('mock:subsection', autoClose=>1);
Tag('mock:p', autoClose=>1);
Tag('mock:p', autoOpen=>1);

#make sure Perl work
1;
```

A.3 mockDoc RelaxNG schema

```
default namespace md = "https://kwarc.info/projects/mockDoc"

start = document
document = element document {p*, section*}
section = element section {title,p*,subsection*}
subsection = element subsection {title,paragraph*}
paragraph = element paragraph {title, p+}
title = element title { text }
p = element p { (text|break)*}
break = element break { empty }
```

A.4 XSL stylesheet

```
<?xml version="1.0" encoding="utf-8"?>
<!-- customized for mockDoc conversion -->

<!-- style sheet declration -->
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:mock = "https://kwarc.info/projects/mockDoc"
    exclude-result-prefixes="mock">

<xsl:template match="/">
    <html>
        <head><xsl:comment>automatically generated, handle with care</xsl:comment></head>
        <xsl:apply-templates/>
    </html>
</xsl:template>

<xsl:template match="mock:document">
    <body><xsl:apply-templates/></body>
</xsl:template>

<!-- sectioning commands do not leave a trace, we deal with the headers below -->
<xsl:template match="mock:section|mock:subsection|mock:paragraph">
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="mock:section/mock:title">
    <h1><xsl:apply-templates/></h1>
</xsl:template>

<xsl:template match="mock:subsection/mock:title">
    <h2><xsl:apply-templates/></h2>
</xsl:template>

<xsl:template match="mock:paragraph/mock:title">
    <h3><xsl:apply-templates/></h3>
</xsl:template>

<!-- caution, we are changing the namespace here -->
<xsl:template match="mock:p">
    <p><xsl:apply-templates/></p>
</xsl:template>

<xsl:template match="mock:break"><br/></xsl:template>
</xsl:stylesheet>
```

A.5 Generated HTML

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<!--automatically generated, handle with care-->
</head>
<body>

  <h1>First section</h1>
  <p>Here is some text.
</p>

  <h1>Second section</h1>

  <h2>Subsection I</h2>

    <h3>Paragraph 1</h3>
    <p>
Here is some text.
</p>

    <h3>Paragraph 2</h3>
    <p>
Line 1 <br>We try to test line break and paragraph II here.
</p>

    <h2>Subsection II</h2>

    <h3>Paragraph 1</h3>
    <p>
We try to test subsections II here.
</p>

</body>
</html>
```

B A Makefile for Automation

```
#makefile for using latexml and pdflatex to generate *.pdf and *.xml
#declaration of variables
#set .tex as source. In our case only mockDoc is available
#name .xml and .pdf based on .tex
SRC = mockDoc.tex
XML = $(SRC:%.tex=%.xml)
PDF = $(SRC:%.tex=%.pdf)
HTML = $(SRC:%.tex=%.html)

all: $(XML) $(PDF) $(HTML) howTo.pdf

mockDoc.rnc: mockDoc.rnc
    java -jar trang.jar -I rnc -O rng mockDoc.rnc mockDoc.rng

#the codes below follow the usage of variables mentioned above
# $@ : object filename. $< :source file name
$(XML): %.xml: %.tex mockDoc.rng mockDoc.cls.ltxml
    latexmlc $< --format=XML --destination=$@ --log=$@.log

$(HTML): %.html: %.xml mockDoc.xsl
    xsltproc -o $@ mockDoc.xsl $<

$(PDF): %.pdf: %.tex mockDoc.cls
    xelatex $<

howTo.pdf: howTo.tex $(XML) $(PDF) $(HTML) howTo.bib
    pdflatex howTo
    bibtex howTo
    pdflatex howTo
    pdflatex howTo
```