How To Write A Minimal LaTeXML Binding

Author: Hang Yuan, Jinbo Zhang

Supervisor: Michael Kohlhase

February 19, 2015

Chapter 1

Introduction

LaTeX has been widely used as the word processing tool among scholars, especially when one needs to use large quantities of mathematical representations. LaTeXis also good for those who are meticulous about typographical quality of the documents. However, LaTeX lacks of conversion tool to XML which Digital Library of Mathematical Functions uses for delivery. DLMF developed LaTeXML , trying to make a new typesetting system that allows users to be able to focus more on the content, but not the style, by providing extensive ways of customizations. In order to achieve this goal, building up the document class binding seems crucial, and yet LaTeXML seems fairly unfathomable for beginners. We want to make it easier for those who want to pick up using LaTeXML in the future, by going through how to construct a minimal LaTeXML binding step by step.

This document does not cover advanced topics related LaTeXML, and thus if you are interested the general theories, probably you can have the manual hand in hand with this document to have a better comprehension between the theories and the application. In addition, I will refer you to the particular chapters in the manual, when needed.

Chapter 2

Your First LaTeXML Document Class

2.1 Using LaTeXML

We are going to talk about various aspects of LaTeXML, and then we will move onto the workflow of creating your first LaTeX document class binding. In this tutorial, we use the command:

```
1 latexml
```

for converting TeX document into *.sml One quick note in regards to LaTeXML installation, when you think you have finished installing LaTeXML, run a simple conversion command within mockDoc.tex's directory. You should be able to see an XML interpretation of mockDoc.tex either in a form a standard output or a newly-generated XML file. It is totally fine to see tons of mysterious error messages at this point, because we have created anything yet. Under some circumstances when your LaTeXML doesn't seem to function, maybe you have overlooked the prerequisites such as libxml2 and libxslt.

> *For more information about how to use LaTeXML, please have a look at the LaTeXMLmanual chapter 2:Using LaTeXML.*

2.2 How to Create A LaTeXML Binding

The conversion from TEX to XML is processed by LATEXML. Basically LATEXML maps the TEX markups to the XML markups, more specifically macros, primitives and constructors. That's why you are able to customize the conversion between TEX and XML, in three ways, modifying the bindings used by **latexml**, adding your own bindings that has not been implemented, and even creating your own TEX style and LATEX binding which is exactly the goal of this tutorial.

2.2.1 Things We need

It probably would be a good idea to name every file after the same prefix which will make your life easier in the future. We need to have:

*.tex as your source file, so you can have something to convert from. You can write down whatever you want and based on this .tex file, your other files will vary. Feel free to define your own macros into something unusual such that, even if you accidentally load the TEX binding in LATEXML, the conversion will fail, ensuring all the conversion is done by our LATEXML binding;

*.cls for LATEXpdf which essentially helps you to see what our .tex file is like in a pdf format, since pdflatex is unable to process the customized macros and things alike;

*.cls.ltxml your LATEXML binding, similar to the *.doc.cls you have for LATEX, but used for the conversion to other formats ;

*.rnc The RelaxNG schema compact form, which defines the structure of your .tex, crucial for executing tasks like placing the tags correctly and auto closing the tags when needed;

trang.jar(optional): LATEXML cannot process the compact form scheme, and therefore you need trang to convert your .rnc into .rng,

unless you want to write your scheme in .rng from the first, albeit this approach is not recommended for lack of efficiency and diffculty of maintenance;

After you have finished writing all the documents above, run LaTeXML, and then you should be able to see the converted xml of your .tex. In the following chapters I will explain how to construct your *.doc.ltxml and *.rnc and the dos and don'ts detail

2.2.2 Minimal LaTeXML

Since LaTeXbinding is a Perl module, we need to initialize a binding file by add the followings in the beginning of the document class:

```
1 package LaTeXML::Package::Pool;
2 use strict;
3 use LaTeXML::Package;
4 use warnings;
```

At the end of LaTeXML don't forget to include

```
1 1;
```

to make sure Perl work properly. These are just the must dos, one has to follow. If you don't understand them, it is OK for now. Just use them and they work. The meat of LaTeXML bindings come form the construction and constructors.

> *It will be good idea to read the manual Chapter 4: Customization, before your proceed and come back to see how the theories are implemented.*

Assuming you have read chapter 4 thoroughly, and get some feelings about how things work. Now you want to teach LaTeXML the new commands you created in your .tex file. Let's look at an example below:

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

The reason why I use the break as an example is because you might encounter problems dealing with break in LaTeXML. The two backlashes macro is preserved in pool package, that's why if you still use the regular newline break macro, your LaTeXML will have a malformed error. Renaming your newline macro in your .tex will solve the problem for you.

After you link your .tex file and .cls.ltxml file by changing your document class in your .tex into your LaTeXML binding name, in our case "doc". LaTeXML will load your binding file, when it tries to do the conversion.

You might be wondering how LaTeXML reads your biding. To put it in a simple way, during the conversion process, whenever LaTeXML encounters a macro or control sequence, it will look for its replacement in your binding and then put the replacement in xml. This is where things get a little tricky. How about the closing tag? Just like section macro, you declare where the section starts and were the next section starts, nevertheless, you never write now close section, so LaTeXML will never close the section tags? Yes and no. Indeed LaTeXML will have no clue of where to close the declared tags if we don't tell it when to do so. I solve the problem by using auto->which has something to do with your scheme.

2.2.3 RelaxNG Schema

Schema is a crucial document that decides how the xml is constructed. When you are creating your own schema, it is a good idea to have your .tex document open side by side to make sure your scheme works well with your .tex file.

One good approach to test this is to create your expected xml output according to your .tex and then validate the test xml with your scheme. You can easily accomplish this by using emacs nxml mode in which you have the freedom to write your expected xml, while validating your xml at the same time. If validation fails, you can see the error message instantly, such that you can debug your xml or schema accordingly.

Tutorial: Emacs: Nxml Mode

In our mockDoc.rnc, you can easily see under a document, there can be either p or section and under a section there can be the possibilities of having a title followed by p or a title followed by a subsection. The reason for this is because in mockDoc.tex in the first section, there is no subsection but text directly but in the other sections, there are subsections. What I am trying to say is, in your schema you need to consider all kinds of possible hierarchy of your elements.

> Before you write your expected xml and RelaxNG schema, having a look at the links below can be beneficial:
>
> I.RelaxNG Syntax Tutorial;
>
> II. XML tutorialsl.

Some more improvements: If you have followed what I said, very likely you still have many errors when you use LaTeXML to compile your files. Don't be frustrated by this, when I tried to make my first binding, this "HowTo" didn't exit at all. The success is within a reach. We only need to deal with two more things, namespace and putting spaces in your text.

We have a default namespace in the schema and we need to declare the schema in the binding and associate the prefix with the namespace. That's an easy step. Then we come to the obscure command of putting spaces between two words. It is related to the architecture of LaTeXML, which is far beyond the scope of this tutorial. So you can just do what is in the doc.cls.ltxml.

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

Now you should have a minimal setup of what is required for a LaTeXML biding.

Congratulations for being able to follow this tutorial to the end. After the processing the makefile, you should be able to see the generated XML in your current directory which hopefully should look something similar to your expected XML!

Reference

I. Bruce R. Miller, November 21, 2014 *LaTeXML The Manual*
II. 2006 - 2012 Philipp Lehman, *biblatex*