

# **How To Write A Minimal LaTeXML Binding**

Author: Hang Yuan, Jinbo Zhang

Supervisor: Michael Kohlhase

February 16, 2015



# Chapter 1

## Introduction

**L**<sup>A</sup>T<sub>E</sub>X has been widely used as the word processing tool among scholars, especially when one needs to use large quantities of mathematical representations. L<sup>A</sup>T<sub>E</sub>X is also good for those who are meticulous about typographical quality of the documents. However, L<sup>A</sup>T<sub>E</sub>X lacks of conversion tool to XML which Digital Library of Mathematical Functions uses for delivery. DLMF developed L<sup>A</sup>T<sub>E</sub>XML, 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 L<sup>A</sup>T<sub>E</sub>XML seems fairly unfathomable for beginners. We want to make it easier for those who want to pick up using L<sup>A</sup>T<sub>E</sub>XML in the future, by going through how to construct a minimal L<sup>A</sup>T<sub>E</sub>XML binding step by step.

**T**his document does not cover advanced topics related L<sup>A</sup>T<sub>E</sub>XML, and thus if you are interested in understanding how this and how that work, very likely the L<sup>A</sup>T<sub>E</sub>XML manual will serve your needs better.



## Chapter 2

# Understanding L<sup>A</sup>T<sub>E</sub>X<sub>ML</sub>

### 2.1 Using LaTeXML

The first thing we want to talk about is, what aspects of L<sup>A</sup>T<sub>E</sub>X<sub>ML</sub> we are going to cover, and then we will move onto the workflow of creating your first L<sup>A</sup>T<sub>E</sub>X document class binding. In this tutorial, we use command:

**latexml** for converting T<sub>E</sub>X document into \*.XML

The general command for conversion is

**latexml** options -destination=doc.xml doc

Or simply you only supply with the T<sub>E</sub>X file and the result will be standard output which is totally fine as well, based on your needs. One quick note here about L<sup>A</sup>T<sub>E</sub>X<sub>ML</sub> installation, when you think you have finished installing L<sup>A</sup>T<sub>E</sub>X<sub>ML</sub>, 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. If you have something that differs from the expected and you have already checked your L<sup>A</sup>T<sub>E</sub>X<sub>ML</sub> package multiple times, maybe you have overlooked some prerequisites such as libxml2 and libxslt.

**Note:** Now in order to make better use of our document class binding in the future, we need to know how L<sup>A</sup>T<sub>E</sub>X<sub>ML</sub> operates in

different stages. The stages are like the following: Digestion , construction, rewriting, math parsing and serialization. However we are not going to discuss this in detail here. For the interested users, the L<sup>A</sup>T<sub>E</sub>X<sub>XML</sub> manual is a good source.

## 2.2 LaTeXML Binding

The conversion from T<sub>E</sub>X to XML is processed by L<sup>A</sup>T<sub>E</sub>X<sub>XML</sub>. Basically L<sup>A</sup>T<sub>E</sub>X<sub>XML</sub> maps the T<sub>E</sub>X makeup to the XML markup, more specifically macros, primitives and constructors. That's why you are able to customize the conversion between T<sub>E</sub>X 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 T<sub>E</sub>X style and L<sup>A</sup>T<sub>E</sub>X binding which is exactly the goal of this tutorial.

### 2.2.1 Minimal LaTeXML Structure

Since L<sup>A</sup>T<sub>E</sub>Xbinding is essentially a Perl module, we need to initialize a binding file by add the followings in the beginning of the document class:

```
package LaTeXML::Package::Pool; //load text package
use strict; // catch errors and stop when encounter one
use LaTeXML::Package;

//your customization here

use warnings; // give warnings
1; //make sure Perl work properly
```

We always need to load Tex.pool binding and possibly LaTeX.pool as well in the beginning. L<sup>A</sup>T<sub>E</sub>X<sub>XML</sub> packages are just like the style and class files in T<sub>E</sub>X and they have an extension of *.ltxml*. We load our L<sup>A</sup>T<sub>E</sub>Xbinding juts like how we load our class file in T<sub>E</sub>X. In our case we would like to load *doc.cls.ltxml*, and therefore we use `\documentclass{doc}`, similarly, if you would like to load a doc type *doc.sty.ltxml*, you only need to include `\use package{doc}` in your .tex file.

## 2.3 Construction & Constructors

As we are interested in the conversion to XML, we need to understand how constructors works.

**DefConstructor** (*\$prototype*, *\$replacement*, *\$options*)

**DefConstructor**(‘\section’, “<mock:section><mock:title>#1</mock:title>”);

The prototype is the control sequence you have defined in your *.tex* file, and the replacement is what you want it to be built in your XML file. There are options that need to be passed. In our *doc.cls.ltxml*, it is important to include

*beforeDigest =<sub { AssignValue(inPreamble =<0);}*

for it makes sure that there spaces between words in the generated XML file. For other similar options, you might not necessarily be able to find them in the manual, albeit, you can go one step further by looking at the other built in bindings such as *article.cls.ltxml* and the *pool* package.

One other error you might encounter when creating a bidding from scratch is the constructor for new line. The macro for new line in T<sub>E</sub>Xis `\\`.

It does not necessarily mean you are able to customize `\\`, due to some predefinition in the *pool* package, which explains why we change our the conventional `\\` in *mockDoc* into `\newline`. The same method can be considered as one possible solution for some *malformed* errors.

## 2.4 Document Model

After customizing how T<sub>E</sub>Xis translated into XML. There are three more schema to include: RelaxNGSchema, RegisterNamespace, and Tag.

### 2.4.1 RelaxNGSchema

The constructors tell *l<sup>A</sup>T<sub>E</sub>X*XML to add a replacement into XML when it detects a prototype in the *.tex* file. The question is it doesn't tell where the tag should be closed, and therefore we introduce

*Tag(\$Tag,\$properties)*

meaning whenever there is new paragraph, the last paragraph tag will be closed, before the new paragraph tag can be added. RelaxNGSchema tells how the whole document class is constructed, such that when we want to create our own document structure, RelaxNGSchema is quite important.

*l<sup>A</sup>T<sub>E</sub>X*XML is unable to process the compact form which has an extension of *.rnc* but *.rng*. It makes your life easier to write in compact form nevertheless. The trick is to convert your *.rnc* into *.rng* using *trang*.

In a schema file, pretty much you are defining in what elements can be included in different bodies. As you can see in *mockDoc.rnc* file. We need to register our namespace first, which we need to do later in *l<sup>A</sup>T<sub>E</sub>X*XML file as well. The definition of paragraph tells *l<sup>A</sup>T<sub>E</sub>X*XML, in a paragraph it has two elements that are title and p. Likewise, section contains a title followed by two possible structures either title or subsection.

Finally we come to last part of our *l<sup>A</sup>T<sub>E</sub>X*XML, namespace. Namespace differentiates our customized macros with others by adding a prefix to our *l<sup>A</sup>T<sub>E</sub>X*XML constructor patterns.

*RegisterNamespace(\$prefix, \$url)*

Congratulations for being able to follow this tutorial to the end. You should have everything you need to create a minimal document *l<sup>A</sup>T<sub>E</sub>X*XML binding. After the processing the makefile, you should be able to see the generated XML in your current directory!

### Reference

I. Bruce R. Miller, November 21, 2014 *l<sup>A</sup>T<sub>E</sub>X*XML The Manual



II. 2006 - 2012 Philipp Lehman, *biblatex*