How To Write A Minimal LaTeXML Binding

Hang Yuan Jinbo Zhang Michael Kohlhase

Computer Science, Jacobs University Bremen

LaTeX has been widely used as a word processing tool among scholars, especially when one needs to use large quantities of mathematical representations. LaTeX is also a good choice for those who are meticulous about typographical quality of documents. However, LaTeX lacks a converter tool to XML. The DLMF (Digital Library of Mathematical Functions) developed LaTeXML, trying to make a new typesetting system that allows users to be able to focus more on the contents, not the style, by providing extensive ways of customizations. In order to achieve this goal, building up the bindings is 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. We will use *mockDoc* [1] as a sample in this tutorial. This tutorial does not cover advanced topics related to LaTeXML, and thus if you are interested in the general theories, please explore the LaTeXML Manual [1] to better comprehend how the theories are implemented.

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 LaTeXML binding. In this tutorial, we use the command:

```
latexmlc mockDoc.tex --format=XML --destination=mockDoc.xml --log=
    mockDoc.xml.log
```

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

Note: Regarding LaTeXML installation, when you think you have finished installing LaTeXML, run a simple command:

```
latexml your_sample.tex
```

to test it. You should be able to see an XML interpretation of `your_sample.tex` in screen immediately. Under some circumstances LaTeXML doesn't seem to work, maybe you fail to install the prerequisites such as `libxml2` or `libxslt` [2].

[1] mockDoc project in Github: `https://github.com/angerhang/mockDoc`

[2] Please visit `http://dlmf.nist.gov/LaTeXML/get.html` for more information.

2 How to Create a LaTeXML Binding

The conversion from LaTeX to XML is processed by LaTeXML. Basically LaTeXML maps the LaTeX markups to the XML markups, more specifically: macros, primitives and constructors.

2.1 Things We Need

`mockDoc.tex` As your source file. You can write down whatever you want.

`doc.cls` For XeLaTeX, which essentially helps you to see what `mockDoc.tex` file looks like in a pdf format. This file won't be illustrated in this tutorial.

`doc.cls.ltxml` LaTeXML binding, the core file of this tutorial. `doc.cls.ltxml` is similar to `doc.cls` , but used for the conversion to other formats.

`mockDoc.rnc` The schema in compact form, which defines the structure of `mockDoc.tex`, crucial for executing tasks like placing the tags correctly and auto closing the tags when needed.

`trang.jar` LaTeXML cannot process the compact form schema, therefore you need `trang.jar` to convert `mockDoc.rnc` into `mockDoc.rng`. The reason for writing `mockDoc.rnc` instead of `mockDoc.rng` is that, `mockDoc.rnc` is much shorter and easier to maintain.

After you have finished writing all the documents above, run the command mentioned before, and then you 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 `doc.cls.ltxml`.

2.2 RelaxNG Schema

Schema is a crucial document that decides how `mockDoc.xml` is constructed. When you are creating your own schema[3], one good approach to test this is to create your expected `mockDoc_sample.xml` by hand, according to your `mockDoc.tex`, then compare `mockDoc_sample.xml` with the generated `mockDoc.xml`. You can easily accomplish this by using *emacs nxml mode*[4], in which you have the freedom to write your expected `mockDoc.xml`, while validating your `mockDoc.xml` at the same time. If validation fails, you can see the error message instantly, such that you can debug your `mockDoc.xml` or schema accordingly.

In our `mockDoc.rnc`:

```
1 document = element document {p, section*}
2 section = element section {title,(p |subsection)*}
```

[3]Before you write your expected xml and schema, having a look at the links below can be beneficial: `http://relaxng.org/compact-tutorial-20030326.html`; `http://www.w3schools.com/xml/`.

[4]Here is a tutorial about Emacs nxml mode: `http://www.emacswiki.org/emacs/NxmlMode`

you 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 `p` or a `title` followed by a `subsection`. This is because in the first section in `mockDoc.tex`:

```
\section{A brief introduction about Shelley}
    Percy Bysshe Shelley (4 August 1792 -- 8 July 1822)...
```

there is no `subsection` but texts directly. But in the other `section`s, there are `subsection`s. In your schema you need to consider all kinds of possible hierarchy of your elements.

2.3 Minimal LaTeXML

Actually this binding is not the smallest one in the world, in `doc.cls.ltxml` we covered:

1 environment: `document`
4 control sequences: `\section, \subsection, \paragraph, \newline`

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

2.3.1 Basic structure

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

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

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

```
1;
```

to make sure that perl works properly.

2.3.2 Configure namespace

With:

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

We declared the namespace associated the prefix `mock` with the namespace.

2.3.3 Define \newline

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

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

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

2.3.4 Define \section

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

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

we defined `\section`. But, think about the closing tags. In `mockDoc.tex`, we declared where the `\section` starts and where the next `\section` starts, nevertheless, we never wrote something like "Now close this section". Here is why we need `mockDoc.rnc`. This schema file tells LaTeXML what the structure of our document, and with:

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

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

2.3.5 Define document

You may think something like:

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

is enough for defining `document` environment. You can try it, you will find that all spaces disappear. What we actually wrote in `doc.cls.ltxml` is:

```
1 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 dicussion in this tutorial.

For an environment, we don't need care about autoclosing, since an environment is always like

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

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

2.3.6 Autoopen for p

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

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

which will surround such texts.

3 Conclusion

Thank you for following this tutorial to the end. After processing the `makefile`, with command:

```
1 make
```

you should be able to see the generated `mockDoc.xml` in your current directory. It should be something similar to your expected `mockDoc_sample.xml`.

References

[1] Bruce R. Miller. LaTeXML The Manual. `http://dlmf.nist.gov/LaTeXML/manual.pdf`.