Zpravodaj Československého sdružení uživatelů TeXu

Hans Hagen

Exporting XML and ePub from ConTeXt

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 27 (2017), No. 1-2, 55–63

Persistent URL: http://dml.cz/dmlcz/150269

Exporting XML and ePub from ConTEXt

Hans Hagen

The article describes the xml backend of ConTEXt, which can be used to produce structured xml documents out of a TEX input. One of the many applications of the xml backend is the conversion to ePub e-book format, which the article covers in detail.

Keywords: ConTEXt, xml, ePub

Introduction

There is a pretty long tradition of typesetting math with TEX and it looks like this program will dominate for many more years. Even if we move to the web, the simple fact that support for MathML in some browsers is suboptimal will drive those who want a quality document to use pdf instead.

I'm writing this in 2014, at a time when xml is widespread. The idea of xml is that you code your data in a very structured way, so that it can be manipulated and (if needed) validated. Text has always been a target for xml which is a follow-up to sgml that was in use by publishers. Because html is less structured (and also quite tolerant with respect to end tags) we prefer to use xhtml but unfortunately support for that is less widespread.

Interestingly, documents are probably among the more complex targets of the xml format. The reason is that unless the author restricts him/herself or gets restricted by the publisher, tag abuse can happen. At Pragma ADE we mostly deal with education-related xml and it's not always easy to come up with something that suits the specific needs of the educational concept behind a school method. Even if we start out nice and clean, eventually we end up with a polluted source, often with additional structure needed to satisfy the tools used for conversion.

We have been supporting xml from the day it showed up and most of our projects involve xml in one way or the other. That doesn't mean that we don't use TEX for coding documents. This manual is for instance a regular TEX document. In many ways a structured TEX document is much more convenient to edit, especially if one wants to add a personal touch and do some local page make-up. On the other hand, diverting from standard structure commands makes the document less suitable for output other than pdf. There is simply no final solution for coding a document, it's mostly a matter of taste.

doi: 10.5300/2017-1-2/55

So we have a dilemma: if we want to have multiple output, frozen PDF as well as less-controlled HTML output, we can best code in XML, but when we want to code comfortably we'd like to use TEX. There are other ways, like Markdown, that can be converted to intermediate formats like TEX, but that is only suitable for simple documents: the more advanced documents get, the more one has to escape from the boundaries of (any) document encoding, and then often TEX is not a bad choice. There is a good reason why TEX survived for so long.

It is for this reason that in ConTEXt MkIV we can export the content in a reasonable structured way to XML. Of course we assume a structured document. It started out as an experiment because it was relatively easy to implement, and it is now an integral component.

The output

The regular output is an XML file but as we have some more related data it gets organized in a tree. We also export a few variants. An example is given below:

```
./test-export
./test-export/images
./test-export/images/...
./test-export/styles
./test-export/styles/test-defaults.css
./test-export/styles/test-images.css
./test-export/styles/test-styles.css
./test-export/styles/test-templates.css
./test-export/test-raw.xml
./test-export/test-raw.lua
./test-export/test-tag.xhtml
./test-export/test-div.xhtml
```

Say that we have this input:

```
\setupbackend
  [export=yes]

\starttext
  \startsection[title=First]
    \startitemize
      \startitem one \stopitem
      \startitem two \stopitem
    \stopitemize
  \stopsection
\stoptext
```

The main export ends up in the `test-raw.xml` export file and looks like the following (we leave out the preamble and style references):

```
<document> <!-- with some attributes -->
  <section detail="section" chain="section" level="3">
    <sectionnumber>1</sectionnumber>
    <sectiontitle>First</sectiontitle>
    <sectioncontent>
      <itemgroup detail="itemize"
        chain="itemize" symbol="1" level="1">
        <item>
          <itemtag><m:math ..><m:mo>•</m:mo></m:math></itemtag>
          <itemcontent>one</itemcontent>
        </item>
        <item>
          <itemtag><m:math ..><m:mo>•</m:mo></m:math></itemtag>
          <itemcontent>two</itemcontent>
        </item>
      </itemgroup>
    </sectioncontent>
  </section>
</document>
```

This file refers to the stylesheets and therefore renders quite well in a browser like Firefox that can handle XHTML with arbitrary tags.

The `detail` attribute tells us what instance of the element is used. Normally the `chain` attribute is the same but it can have more values. For instance, if we have:

```
\definefloat[graphic][graphics][figure]

.....

\startplacefigure[title=First]
    \externalfigure[cow.pdf]
\stopplacefigure

.....

\startplacegraphic[title=Second]
    \externalfigure[cow.pdf]
\stopplacegraphic
```

we get this:

```
<float detail="figure" chain="figure">
```

```
  <floatcontent>...</floatcontent>
  <floatcaption>...</floatcaption>
</float>
<float detail="graphic" chain="figure graphic">
  <floatcontent>...</floatcontent>
  <floatcaption>...</floatcaption>
</float>
```

This makes it possible to style specific categories of floats by using a (combination of) `detail` and/or `chain` as filters.

The body of the `test-tag.xhtml` file looks similar but it is slightly more tuned for viewing. For instance, hyperlinks are converted to a way that CSS and browsers like more. Keep in mind that the raw file can be the base for conversion to other formats, so that one stays closest to the original structure.

The `test-div.xhtml` file is even more tuned for viewing in browsers as it completely does away with specific tags. We explicitly don't map onto native HTML elements because that would make everything look messy and horrible, if only because there seldom is a relation between those elements and the original. One can always transform one of the export formats to pure HTML tags if needed.

```
<body>
  <div class="document">
    <div class="section" id="aut-1">
      <div class="sectionnumber">1</div>
      <div class="sectiontitle">First</div>
      <div class="sectioncontent">
        <div class="itemgroup itemize symbol-1">
          <div class="item">
            <div class="itemtag"><m:math ...>
              <m:mo>•</m:mo></m:math></div>
            <div class="itemcontent">one</div>
          </div>
          <div class="item">
            <div class="itemtag"><m:math ...>
              <m:mo>•</m:mo></m:math></div>
            <div class="itemcontent">two</div>
        </div>
      </div>
      <div class="float figure">
        <div class="floatcontent">...</div></div>
        <div class="floatcaption">...></div>
      </div>
      <div class="float figure graphic">
```

```
        <div class="floatcontent">...</div></div>
        <div class="floatcaption">...></div>
      </div>
    </div>
  </div>
</body>
```

The default CSS file can deal with tags as well as classes. The file of additional styles contains definitions of so-called highlights. In the ConTeXt source one is better off using explicit named highlights instead of local font and color switches because these properties are then exported to the CSS. The images style defines all images used. The templates file lists all the elements used and can be used as a starting point for additional CSS styling.

Keep in mind that the export is *not* meant as a one-to-one visual representation. It represents structure so that it can be converted to whatever you like.

In order to get an export you must start your document with:

```
\setupbackend
  [export=yes]
```

So, we trigger a specific (extra) backend. In addition you can set up the export:

```
\setupexport
  [svgstyle=test-basic-style.tex,
   cssfile=test-extras.css,
   hyphen=yes,
   width=60em]
```

The `hyphen` option will also export hyphenation information so that the text can be nicely justified. The `svgstyle` option can be used to specify a file where math is set up; normally this would only contain a `bodyfont` setup, and this option is only needed if you want to create an ePub file afterwards which has math represented as SVG.

The value of `cssfile` ends up as a style reference in the exported files. You can also pass a comma-separated list of names (between curly braces). These entries come after those of the automatically generated CSS files so you need to be aware of default properties.

Images

Inclusion of images is done in an indirect way. Each image gets an entry in a special image related stylesheet and then gets referred to by `id`. Some extra information is written to a status file so that the script that creates ePub files

can deal with the right conversion, for instance from PDF to SVG. Because we can refer to specific pages in a PDF file, this subsystem deals with that too. Images are expected to be in an `images` subdirectory and because in CSS the references are relative to the path where the stylesheet resides, we use `../images` instead. If you do some postprocessing on the files or relocate them you need to keep in mind that you might have to change these paths in the image-related CSS file.

Epub files

At the end of a run with exporting enabled you will get a message to the console that tells you how to generate an ePub file. For instance:

```
mtxrun --script epub --make --purge test
```

This will create a tree with the following organization:

```
./test-epub
./test-epub/META-INF
./test-epub/META-INF/container.xml
./test-epub/OEBPS
./test-epub/OEBPS/content.opf
./test-epub/OEBPS/toc.ncx
./test-epub/OEBPS/nav.xhtml
./test-epub/OEBPS/cover.xhtml
./test-epub/OEBPS/test-div.xhtml
./test-epub/OEBPS/images
./test-epub/OEBPS/images/...
./test-epub/styles
./test-epub/styles/test-defaults.css
./test-epub/styles/test-images.css
./test-epub/styles/test-styles.css
./test-epub/mimetype
```

Images will be moved to this tree as well and if needed they will be converted, for instance into SVG. Converted PDF files can have a `page-<number>` in their name when a specific page has been used.

You can pass the option `--svgmath` in which case math will be converted to SVG. The main reason for this feature is that we found out that MathML support in browsers is not currently as widespread as might be expected. The best bet is Firefox which natively supports it. The Chrome browser had it for a while but it got dropped and math is now delegated to JavaScript and friends. In Internet Explorer MathML should work (but I need to test that again).

This conversion mechanism is kind of interesting: one enters TEX math, then gets MathML in the export, and that gets rendered by TEX again, but now as a standalone snippet that then gets converted to SVG and embedded in the result.

Styles

One can argue that we should use native HTML elements but since we don't have a nice guaranteed-consistent mapping onto that, it makes no sense to do so. Instead, we rely on either explicit tags with details and chains or divisions with classes that combine the tag, detail and chain. The tagged variant has some more attributes and those that use a fixed set of values become classes in the division variant. Also, once we start going the (for instance) `H1`, `H2`, etc. route we're lost when we have more levels than that or use a different structure. If an `H3` can reflect several levels it makes no sense to use it. The same is true for other tags: if a list is not really a list than tagging it with `LI` is counterproductive. We're often dealing with very complex documents so basic HTML tagging becomes rather meaningless.

If you look at the division variant (this is used for ePub too) you will notice that there are no empty elements but `div` blocks with a comment as content. This is needed because otherwise they get ignored, which for instance makes table cells invisible.

The relation between `detail` and `chain` (reflected in `class`) can best be seen from the next example.

```
\definefloat[myfloata]
\definefloat[myfloatb][myfloatbs][figure]
\definefloat[myfloatc][myfloatcs][myfloatb]
```

This creates two new float instances. The first inherits from the main float settings, but can have its own properties. The second example inherits from the `figure` so in fact it is part of a chain. The third one has a longer chain.

```
<float detail="myfloata">...</float>
<float detail="myfloatb" chain="figure">...</float>
<float detail="myfloatc" chain="figure myfloatb">...</float>
```

In a CSS style you can now configure tags, details, and chains as well as classes (we show only a few possibilities). Here, the CSS element on the first line of each pair is invoked by the CSS selector on the second line.

```
div.float.myfloata { }            float[detail='myfloata'] { }
div.float.myfloatb { }            float[detail='myfloatb'] { }
div.float.figure { }              float[detail='figure']   { }
div.float.figure.myfloatb { }     float[chain~='figure']
                                    [detail='myfloata'] { }
div.myfloata { }                  *[detail='myfloata'] { }
div.myfloatb { }                  *[detail='myfloatb'] { }
div.figure { }                    *[chain~='figure'] { }
div.figure.myfloatb { }           *[chain~='figure']
                                    [detail='myfloatb'] { }
```

The default styles cover some basics but if you're serious about the export or want to use ePub then it makes sense to overload some of it and/or provide additional styling. You can find plenty about CSS and its options on the Internet.

Coding

The default output reflects the structure present in the document. If that is not enough you can add your own structure, as in:

```
\startelement[question]
Is this right?
\stopelement
```

You can also pass attributes:

```
\startelement[question][level=difficult]
Is this right?
\stopelement
```

But these will be exported only when you also say:

```
\setupexport
  [properties=yes]
```

You can create a namespace. The following will generate attributes like `my-level`.

```
\setupexport
  [properties=my-]
```

In most cases it makes more sense to use highlights:

```
\definehighlight
  [important]
  [style=bold]
```

This has the advantage that the style and color are exported to a special CSS file.

Headers, footers, and other content that is part of the page builder are not exported. If your document has cover pages you might want to hide them too. The same is true when you create special chapter title rendering with a side effect that content ends up in the page stream. If something shows up that you don't want, you can wrap it in an `ignore` element:

```
\startelement[ignore]
Don't export this.
\stopelement
```

Acknowledgement

The above article is available through the ConTeXt distribution. It can be found in `/texmf-context/doc/context/sources/general/manuals/epub`.

We would like to thank Karl Berry for proofreading and corrections.

Export dokumentů ve formátu ConTeXt do XML a ePub

Článek popisuje výstupní modul pro ConTeXt, který slouží pro generování strukturovaných xml dokumentů z TeXového vstupu. Jednou z aplikací tohoto modulu, které se článek věnuje blíže, je export do formátu ePub využívaného čtečkami elektronických knih.

Klíčová slova: ConTeXt, xml, ePub

Hans Hagen, pragma@wxs.nl