My Way

Feb. 25, 2010

Getting Web Content and pdf-Output from One Source

Thomas A. Schmitz

ConTEXt's capabilities of typesetting xml allow you to use the same source document for producing both a web page and typeset output. This tutorial will explain the basics of how to use a ConTEXt environment that will process your xml file.

Sometimes, documents that you create will have to "live" in different formats. One common requirement will be that you want to publish their content on the web and have a beautifully typeset version for printing and easier reference. ConTEXt can handle xml files, and with the advent of MkIV, it has sophisticated features to filter and manipulate xml documents.[1]

In this MyWay, I will describe the process of setting up a relatively simple xhtml document so that it can be typeset by ConTEXt. This article is the by-product of something I had to set up at my university department: we wanted to publish a document with reading assignments and bibliographical information for our students. This document will be published on our department's website,[2] but I also wanted a pdf-version that students could print out for easier reference. Maintaining and syncing two different source files (one in html for the website, one in TEX for typesetting) is terribly inefficient and error-prone, so I decided that I wanted to set up a process to typeset the xhtml file with ConTEXt. The document itself is rather simple: it contains text, a few tables, and a few images. It is given as an example that should allow and motivate you to delve further into this subject.

Our source document is coded in "strict" xhtml since the specs for this format (esp. that all elements be properly nested and closed) make it easier to process documents with ConTEXt than "pure" html. We will look at the structure of this xhtml document step by step. After the DOCTYPE declaration and the required <head> and <body> elements, our document begins with a heading and an introduction which contains just text:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

  <head>
    <title>
      Our Document
    </title>
  </head>

  <body>

    <h1>Important Advice</h1>

    <h2>Introduction</h2>
```

[1] If you are interested in the details, chapters XVII and XXIII of the mk manual (http://www.pragma-ade.nl/general/manuals/mk.pdf) contain lots of fascinating background information. A manual for xml in MkIV can be found at http://www.pragma-ade.com/general/manuals/xml-mkiv.pdf.

[2] Where it will be part of our university's CMS system, but this is irrelevant here.

```
    <p>The first paragraph. It contains <q>quoted text</q>,
    <em>emphasized text</em> which should be rendered in
    <em>italics</em>, and <b>bold text</b>.</p>
  <body>
<html>
```

If we want to process this file, we will need to tell ConTEXt what to do with the different elements and with the document as a whole. For this, we need to write an environment file. If you have ever written something in html, you can think of this file as the equivalent of an external css file. As you may have heard, ConTEXt makes use of LuaTEX, which will completely replace pdfTEX in time. Many parts of ConTEXt now exist in an older version (for good old pdfTEX), which is called MkII, and a newer version (for LuaTEX), which is called MkIV. In most areas, there are no big differences in the user interface, but since LuaTEX is far superior in this area, Hans Hagen has rewritten the entire xml handling mechanism from scratch. The new code allows more control over what to do with different xml elements, and it is much faster for complex documents. For the time being, there is not enough documentation for beginners – hence this MyWay. It will describe how to code an environment for use with MkIV. Our environment will basically contain two parts:

1. Setups for handling the different xhtml elements, tags, and attributes,

2. and the setup for typesetting our document, i.e., the information which is normally contained in the preamble of your ConTEXt documents.

With that in mind, let us begin by looking at the different elements of our environment file. I will explain what they do as we go.

```
\startxmlsetups xml:oursetups
  \xmlsetsetup{\xmldocument}{*}{-}
  \xmlsetsetup{\xmldocument}{html|body|h1|h2|p|em|q|b}{xml:*}
\stopxmlsetups

\xmlregistersetup{xml:oursetups}
```

We begin by defining our setups. The line `\startxmlsetups xml:oursetups` defines an environment for our set and names it `oursetups`. The two lines within this environment tell ConTEXt what to do with the different elements: the line `\xmlsetsetup{\xmldocument}{*}{-}` tells it to disregard everything; this way, only elements that we name explicitly will be typeset. In our case, this is useful because we do not want the "title" element of the header to be typeset. The next line lists all the elements which we *do* want to be processed and typeset. As we will define further elements, we will have to remember to add them to this line, or they will be silently disregarded! Finally, we "register" our setup under its name.

Next, we will tell ConTEXt what it should do with the different elements:

```
\startxmlsetups xml:html
  \xmlflush{#1}
\stopxmlsetups

\startxmlsetups xml:body
  \xmlflush{#1}
\stopxmlsetups

\startxmlsetups xml:h1
  \chapter{\xmlflush{#1}}
\stopxmlsetups

\startxmlsetups xml:h2
  \section{\xmlflush{#1}}
\stopxmlsetups

\startxmlsetups xml:p
  \xmlflush{#1}\par
\stopxmlsetups

\startxmlsetups xml:em
  \dontleavehmode{\em \xmlflush{#1}}
\stopxmlsetups

\startxmlsetups xml:q
  \quotation{\xmlflush{#1}}
\stopxmlsetups

\startxmlsetups xml:b
  \dontleavehmode{\bf\xmlflush{#1}}
\stopxmlsetups
```

These different setup elements are the most important part of our environment file. They tell ConTEXt how to translate xhtml tags into ConTEXt commands. If you look at these definitions, you will see that they are not difficult to understand: for every element you want processed, you need a setup command. Every element is prefixed by the xml: namespace; the name of the element follows. The first two commands tell ConTEXt to simply "flush," i.e., transmit the content of the `<html>` and `<body>` elements to the typesetting engine. Things become more interesting with the different headers: here, we want headings at the level of `<h1>` to be typeset as chapter headings in ConTEXt. That's what the line `\chapter{\xmlflush{#1}}` does: it takes the content between the `<h1></h1>` tags and "flushes" it as the argument of the `\chapter` command. `<p>` elements are paragraphs; hence, they are flushed and a `\par` is added at the end. You'll see easily what the other setups do. Since switches such as `\em` and `\bf` need to be inside groups, we add an extra pair of braces; since

they might make problems if they start a paragraph, we have to be cautious and add a `\dontleavehmode` at the beginning.

After these xml setups, the second part of our environment file contains just the normal setup for typesetting. If you are a little bit familiar with ConTEXt, this should be easy to understand, and I won't go into the details here:

```
\usetypescript[termes]
\setupbodyfont[termes,11pt]

\setupbodyfontenvironment[default][em=italic]

\setuphead[chapter][page=yes,
                    header=empty,
                    align=middle,
                    after={\blank[line]}]

\setuphead[section][page=no,
                    align=middle,
                    number=no,
                    before={\blank[2*line]},
                    after={\blank[line]}]

\setupindenting[medium,yes]
```

This, then, is all you need if you want to process normal text in paragraphs and headings. We can now typeset our file (which we name `sample.xml`) with this environment (which we call `ourenvironment.tex`) with this command:

```
context --environment=ourenvironment sample.xml
```

The output will be saved as `sample.pdf`, and it should show all the elements we have defined. Things become a bit more complex when we want to build tables. In the document I was writing, there were two types of table, one with two columns, one with three columns. In html, this is not problematic since the browser will reflow the text according to the width of the window. In a printed version, however, we want more control over the relative width of the single table columns. In order to achieve this, we need to distinguish between the two types of tables. We assign them two different `class` attributes in our xhtml code:

```
<table class="threecol">
  <tr>
    <th>
      Heading One
    </th>
    <th>
      Heading Two
```

```
      </th>
      <th>
        Heading Three
      </th>
    </tr>
    <tr>
      <td>
        A Paragraph
      </td>
      <td>
        <em>A Title</em>
      </td>
      <td>
        <b>An Explanation</b>
      </td>
    </tr>
  </table>
  <table class="twocol">
    <tr>
      <td>
        A
      </td>
      <td>
        A lengthy paragraph, with <q>text in quotation marks</q>
        and all sorts of other stuff.
      </td>
    </tr>
    <tr>
      <td>
        B
      </td>
      <td>
        And yet another paragraph.
      </td>
    </tr>
  </table>
```

The first thing we will have to remember is to add these elements to the top of our environment so they will get processed:

```
\xmlsetsetup{\xmldocument}{html|body|h1|h2|p|em|q|b|table|tr|th|td}{xml:*}
```

This is important for all elements that we will use; I will assume that you remember this from now on. But how can we typeset these tables? ConTEXt offers "Natural Tables."[3] They are quite similar in their setup to html tables, so it is relatively easy

[3] For details, see the Natural Tables manual at http://pragma-ade.com/general/manuals/enattab.pdf.

to map this code to ConTEXt code. We will use the "class" attributes to define two different setups:

```
\startxmlsetups xml:table
  \doifelse {\xmlatt{#1}{class}} {threecol} {
    \setupTABLE[c][1][align=right,width=.2\textwidth]
    \setupTABLE[c][2,3][align=right,width=.4\textwidth]
    \bTABLE[frame=on,split=yes]
    \xmlflush{#1}
    \eTABLE }
    {
    \setupTABLE[c][1][align=right,width=.05\textwidth]
    \setupTABLE[c][2][align=right,width=.95\textwidth]
    \bTABLE[frame=on,split=yes]
    \xmlflush{#1}
    \eTABLE
    }
\stopxmlsetups

\startxmlsetups xml:tr
  \bTR \xmlflush{#1} \eTR
\stopxmlsetups

\startxmlsetups xml:th
  \bTD [align=middle,style=bold] \xmlflush{#1} \eTD
\stopxmlsetups

\startxmlsetups xml:td
  \bTD \xmlflush{#1} \eTD
\stopxmlsetups
```

Let us look at this code in detail: first, we tell ConTEXt that we want to process `<table>` elements:

```
\startxmlsetups xml:table
```

Then, we use a condition to process this element. The syntax for this conditional in ConTEXt is `\doifelse {string1} {string2} {then ...} {else ...}`:[4] we compare "string1" to "string2." If they are identical, the "then" branch is executed; if they are different, the "else" branch is executed. The command

```
\doifelse {\xmlatt{#1}{class}} {threecol}
```

[4] There is an excellent article by Taco Hoekwater on system macros at http://tex.aanhet.net/context/syst-gen-doc.pdf; the same material is available on the ConTEXt wiki (http://wiki.contextgarden.net/System_Macros) as well.

thus compares the value of the attribute `class` of the current element (that's what `\xmlatt{#1}{class}` expands to) with the string "threecol." So: if the "class" attribute is set to "threecol," we set up a table in which the first column occupies 20 % of the textwidth, columns two and three 40 %, respectively. If it is set to any other value, we set up a table in which the first column holds 5 % of the textwidth and the second column the remaining 95 %. (If we need more different types of tables, we would have to nest such `\doifelse` macros).

The rest is straightforward: `<th>` elements are wrapped in `\bTD \eTD` pairs, and are formatted as bold, centered text; `<tr>` and `<td>` elements are wrapped in the corresponding commands for table rows and table cells for natural tables.

Let us lok at one further point: in my tables, I wanted some cells to span several rows. How is this done? In xhtml, there is the `rowspan` attribute:

```
<table class="threecol">
  <tr>
    <td>
      A
    </td>
    <td>
      1
    </td>
    <td rowspan="3">
      Three rows
    </td>
  </tr>
  <tr>
    <td>
      B
    </td>
    <td>
      2
    </td>
  </tr>
  <tr>
    <td>
      C
    </td>
    <td>
      3
    </td>
  </tr>
</table>
```

A similar effect can be achieved in a natural table in ConTEXt. The syntax here is `\td [nr=3]`. So all we have to do is extract the value of the attribute of `rowspan`

and "feed" it to the nr argument in our ConTEXt table. But there is one further problem: if a <td> element does not have a rowspan attribute, its value does not exist, of course. We must make sure that such a non-existent value is not transmitted to the nr argument, or ConTEXt will complain about a "missing number." We modify our definition of the <td> element: at first, we test whether rowspan does have a numerical value; if it does, we feed this number to ConTEXt. Again, we use one of the nifty system conditionals that ConTEXt provides:

```
\startxmlsetups xml:td
\doifnumberelse
  {\xmlatt{#1}{rowspan}}
  {\bTD [nr=\xmlatt{#1}{rowspan},align=lohi] \xmlflush{#1} \eTD}
  {\bTD \xmlflush{#1} \eTD}
\stopxmlsetups
```

You have probably understood what this code does: the command \doifnumberelse takes three arguments. It checks whether the first argument is a number; here this first argument is the attribute rowspan of the current element. If this is a number, it will use this number as assignment for the nr attribute in ConTEXt's table and flush the content of the element between the table commands \bTD and \eTD. If it isn't a number (because the attribute doesn't exist), it builds a "normal" table cell without any additional arguments.

So much for tables. Let us now take a look at another interesting aspect of html: embedding images. Here's a typical way an image is embedded in html:

```
<p style="text-align:middle">
  <img src="hacker.jpg" width="50%" alt="hacker"></img>
</p>
```

As you see, the <img> element takes attributes which define the image to be included, its width, and an alternative text which should appear in case the image does not load. We can use this text for our image caption, and it is clear that we will need the image name as well. However, there is a problem with the width parameter: in xhtml, it can be given either in pixels, in which case it will be given as a number only, or in percent of the containing element. These cases need a special treatment: if the width is given in pixels, we can easily use this number to give the size in points, but we will have to add the unit pt. If it is given in percent, we will have to get rid of the % sign (which would confuse the TEX engine) and convert it to a format that ConTEXt uses, which is usually in the form 0.x\textwidth. This conversion could be done in TEX, but since we are using LuaTEX, we have the convenience of the Lua language, which we will use here. At first, we write a Lua function that converts the value of the width attribute:[5]

[5] I'm grateful to Taco Hoekwater who provided help with the lua code.

```
function getmeas(s)
   if string.find(s, "[^0-9]") then
      s = s:sub(1,-2)
      s = s / 100
      s = s.."\\textwidth"
      tex.sprint(tex.ctxcatcodes, s)
   else
      s = s.."pt"
      tex.sprint(s)
   end
end
```

Providing an introduction to the Lua language is beyond the scope of this MyWay; I give just a few short explanations: Since the xhtml attribute `width` can either be a number or a number with a percent sign, we know that any value which contains more than just digits must be a percentage. The function `getmeas` takes a string `s`. It then tests whether this string contains anything but digits (that's what the line `if string.find(s, "[^0-9]")` does). If it contains anything but digits (i.e., digits and a percent sign), the `then` branch is executed: first, we extract a substring from our string `s` which extends from the first character to the last but one character with the code `s = s:sub(1,-2)`. This will thus give us the number, without the % sign. We then divide this number by 100 (`s = s / 100`) and append the TEX string `\textwidth` to it. Finally, we pass this new string (which now has the form `0.25\textwidth`) to the LuaTEX engine. If our string `s` contains only digits, we simply append the unit `pt` to it and pass it to LuaTEX; it now has the form `25pt`.

We wrap this Lua function in a pair of `\startluacode` `\stopluacode` delimiters. We can now finally write the setup for `img` element:

```
\startxmlsetups xml:img
  \placefigure
    [here]
    [\xmlatt{#1}{src}]
    {\xmlatt{#1}{alt}}
    {\externalfigure[\xmlatt{#1}{src}]
                    [width=\ctxlua{getmeas("\xmlatt{#1}{width}")}]}
\stopxmlsetups
```

So: when TEX finds an `img` element, it will place a `\placefigure` command. It will use the name of the image (which is given in the `src` attribute) as the identifier of this figure and the content of the `alt` attribute for the caption. Finally, it will place the image itself as an `\externalfigure`, again using the content of the `src` attribute and the content of the `width` attribute to calculate the width. One last word about images: as you know, html can include both local images and images retrieved from the web via URIs. You will be relieved to know that the same is possible with ConTEXt: both `\externalfigure[nameoflocalfigure]` and `\externalfigure[http://www.someplace/someimage.jpg]` will work.

As you see, ConTEXt MkIV offers rich possibilities of processing and manipulating xml content. It is even possible to filter the content of the xml data and only typeset content which matches certain criteria. Here's an example:

```
\startxmlsetups xml:p
  \xmltext{#1}{q}
\stopxmlsetups
```

What this setup does is: it looks at the element <p> and then only typesets subelements of type <q> within this element. This may come in handy if you want to select only certain elements from your file. A command that is even more powerful is \xmlfilter; it can filter your xml data and only process it if it meets certain conditions (only elements which have a certain attribute, or whose text contains a certain string).

This MyWay was meant to whet your appetite. ConTEXt MkIV offers many sophisticated options to filter, manipulate, and typeset xml files. This brief tutorial was meant to give beginners a point where to start exploring these opportunities. If writing, editing, and maintaining documents which will end up on the web and which should also be typeset is part of your workflow, you should definitely have a look at these possibilities.

To make it easier for you to experiment, I have included the xml file and the environment here. First the file `sample.xml`:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

  <head>
    <title>
      Our Document
    </title>
    <style type="text/css">
      table
      {
      width:100%;
      border:2px solid black;
      }
      th
      {
      height:75px;
      border:1px solid black
      }
      tr
      {
      height:50px;
      }
      td
      {
      border:1px solid black
      }
    </style>
   </head>

  <body>

    <h1>Important Advice</h1>

    <h2>Introduction</h2>

    <p>The first paragraph. It contains <q>quoted text</q>,
        <em>emphasized text</em> which should be rendered in
       <em>italics</em>, and <b>bold text</b>.</p>
    <table class="threecol">
      <tr>
       <th>
```

```
      Heading One
    </th>
    <th>
      Heading Two
    </th>
    <th>
      Heading Three
    </th>
  </tr>
  <tr>
    <td>
      A Paragraph
    </td>
    <td>
      <em>A Title</em>
    </td>
    <td>
      <b>An Explanation</b>
    </td>
  </tr>
</table>
<table class="twocol">
  <tr>
    <td>
      A
    </td>
    <td>
       A lengthy paragraph, with <q>text in quotation marks</q>
       and all sorts of other stuff.
    </td>
  </tr>
  <tr>
    <td>
      B
    </td>
    <td>
      And yet another paragraph.
    </td>
  </tr>
</table>
<table class="threecol">
  <tr>
    <td>
      A
    </td>
    <td>
```

```
          1
         </td>
         <td rowspan="3">
          Three rows
         </td>
        </tr>
        <tr>
         <td>
          B
         </td>
         <td>
          2
         </td>
        </tr>
        <tr>
         <td>
          C
         </td>
         <td>
          3
         </td>
        </tr>
       </table>

       <p style="text-align:center">
        <img src="hacker.jpg" width="80%" alt="The hacker sample image"></img>
       </p>

       <p style="text-align:center">
        <img src="http://wiki.contextgarden.net/skins/common/images/logo_circ_small
width="100" alt="The ConTeXt logo"></img>
       </p>
     </body>
</html>
```

And here the environment `ourenvironment.tex`:

```
\startluacode
function getmeas(s)
   if string.find(s, "[^0-9]") then
      s = s:sub(1,-2)
      s = s / 100
      s = s.."\\textwidth"
      tex.sprint(tex.ctxcatcodes, s)
   else
      s = s.."pt"
```

```
      tex.sprint(s)
   end
end
\stopluacode

\startxmlsetups xml:oursetups
       \xmlsetsetup{\xmldocument}{*}{-}
       \xmlsetsetup{\xmldocument}{html|body|h1|h2|p|em|q|b|table|tr|th|td|img}{xml:*}
\stopxmlsetups

\xmlregistersetup{xml:oursetups}

\startxmlsetups xml:html
       \xmlflush{#1}
\stopxmlsetups

\startxmlsetups xml:body
       \xmlflush{#1}
\stopxmlsetups

\startxmlsetups xml:h1
       \chapter{\xmlflush{#1}}
\stopxmlsetups

\startxmlsetups xml:h2
       \section{\xmlflush{#1}}
\stopxmlsetups

\startxmlsetups xml:p
       \xmlflush{#1}\par
\stopxmlsetups

\startxmlsetups xml:em
       \dontleavehmode{\em \xmlflush{#1}}
\stopxmlsetups

\startxmlsetups xml:q
       \quotation{\xmlflush{#1}}
\stopxmlsetups

\startxmlsetups xml:b
       \dontleavehmode{\bf\xmlflush{#1}}
\stopxmlsetups

\startxmlsetups xml:table
       \doifelse {\xmlatt{#1}{class}} {threecol} {
```

```
                \setupTABLE[c][1][align=right,width=.2\textwidth]
                \setupTABLE[c][2,3][align=right,width=.4\textwidth]
                \bTABLE[frame=on,split=yes]
                \xmlflush{#1}
                \eTABLE }
                {
                \setupTABLE[c][1][align=right,width=.05\textwidth]
                \setupTABLE[c][2][align=right,width=.95\textwidth]
                \bTABLE[frame=on,split=yes]
                \xmlflush{#1}
                \eTABLE
                }
\stopxmlsetups

\startxmlsetups xml:tr
       \bTR \xmlflush{#1} \eTR
\stopxmlsetups

\startxmlsetups xml:th
       \bTD [align=middle,style=bold] \xmlflush{#1} \eTD
\stopxmlsetups

\startxmlsetups xml:td
\doifnumberelse {\xmlatt{#1}{rowspan}}
                {\bTD [nr=\xmlatt{#1}{rowspan},align=lohi] \xmlflush{#1}
\eTD}
                {\bTD \xmlflush{#1} \eTD}
\stopxmlsetups

\startxmlsetups xml:img
       \placefigure[here]
                   [\xmlatt{#1}{src}]
                    {\xmlatt{#1}{alt}}
                   {\externalfigure[\xmlatt{#1}{src}][width=\ctxlua{getmeas("\xml
\stopxmlsetups

\usetypescript[termes]
\setupbodyfont[termes,11pt]

\setupbodyfontenvironment[default][em=italic]

\setuphead[chapter][page=yes,
                    header=empty,
                    align=middle,
                    after={\blank[line]}]
```

```
\setuphead[section][page=no,
                    align=middle,
                    number=no,
                    before={\blank[2*line]},
                    after={\blank[line]}]

\setupindenting[medium,yes]
```