This Way

ConTEXt magazine #8
November 2004

Dealing with XML
Hans Hagen
PRAGMA ADE

In this issue I will drop an idea that has been buzzing in my mind for a long time: how to comfortably deal with typesetting documents coded in XML, while using the manipulative power of XSLT on the one hand and the typesetting capabilities of CONTEXT at the other hand, without seeing TEX code.

Introduction

When implementing an XSL-FO engine, I found out that it's not that trivial to implement robust mechanisms, i.e. to guarantee output that conforms to what we normally want to achieve with ConTeXt.

So far I encountered roughly two applications where XSL-FO are used:

- short (few page) documents, whose appearance resembles forms
- large documents with a relatively simple layout

The first category demands a simple page builder as well as means to position data onto the page(body). The data is rather static. The second category is more demanding in terms of typesetting, and needs ways to deal with mixed page buildup.

The XSL-FO model is not that far from cascading stylesheets, the way to deal with styling text on the web. As with much in the XML world, it has the promise of being the ultimate solution to all problems. But getting something on paper is not the same as getting it on screen, if only because (most noticably in educational documents) complex page layouts may be used when typesetting books. A realistic view on its XSL-FO applicability depends on the place someone has in the workflow from XML to (for instance) PDF. In this context I often remark that "The problem hasn't changed.". What I mean with this is that, at the level where the final mapping from one onto the other has to be done, it's the (quality of) the design that determines the tools to be used. One time a solution using XSL-FO is nice, clean and robust, another time, mapping onto a full featured typesetting engine like ConTeXt makes more sense. And, when in the process the designs evolve (often becoming less consistent over time) both solution may end up in messy source code, and all claims for clean, portable solutions become invalid. But, given that most documents —at the moment using XML based solutions make most sense when applied to series of documents— are one–time shots, this is not that big an issue.

To come back to the two applications mentioned, it is interesting to notice that when XML is used in large scale applications where many pages are to be typeset that all look more or less the same, direct mappings onto a typesetting language like TeX is still way more efficient that using all kind of transformations. However, who wants to learn TeX, with its not always that intuitive interface (a mixture between typesetting and programming features)?

The other application I mentioned, simple documents (that can have hundreds of pages), are often so simple to typeset, that using TeX sounds like overkill. This may be one of the reasons why TeX has always gained more interest from its original audience (mathematicians) than from those who need to typeset for instance novels. Anyhow, setting up a style for a document with just text, some sectioning, a few tables and some itemized lists, is not that complex.

So, what then are the problems with using XSL-FO for these documents? It is no secret that an author can mess up a TEX document by using font switches and spacing at will. Scientific publishers spend much time cleaning up source code, i.e. removing such fine tuning. The same is true for a TEX document style: it's no problem to make it look like a real mess, and it's often hard to decide if it is indeed a mess or just a complex solution for a complex problem. Sometimes users copy code from existing styles, and as a result, much of the code in styles is not doing anything or all, or worse, interfering in unexpected ways.

Although an XSL-FO source is not supposed to be read by humans, some of the test files that I used as testbed gave me a kind of familiar feeling of such a mess. The positive side is that one can clean up the mess by redesigning the XSLT conversion script. But even then, in the end there is this huge file with lots of font switches and spacing directives at each level. When done right, this is no problem (apart from being time consuming for a formatter). However, in order to let a typesetting engine do a proper job, it needs to know what it's dealing with, and since this information is not present in the file anymore, it ends up gambling. It goes beyond this MyWay to discuss this in depth, but it's this observation that brings me to a slightly different approach. As an implementer of a rather large typesetting system, it gives an uncomfortable feeling to cook up imperfect solutions, knowing that a little bit more information already would give better results. Typesetting on grids is for instance a rather special business, as is typesetting in columns and tables. There is not one solution. A nice example of a comparable situation is in MATHML: in many cases content based markup gives better results than presentational markup. Alas coding in structure happens less than coding by pasting together pieces like in presentational MATHML.

So, here we are. We have XML document sources. We have a powerfull language for manipulating such sources called XSLT. Sometimes the quality of the XML source may demand some cleanup beforehand, using for instance the text processing capabilities of modern scripting languages. The question we need to answer is: will we use XSL-FO, and one of the available processors, or will we use a specialized formatting engine, or will we use a combination of both? (Imagine for instance an XSL-FO document instance being used as placed XML.)

Here I will discuss a mixture between structured and generated style markup. When users like it, this may become a thread in the CONTEXT story. Also, it may provide those who now use XSLT to map onto TEX code to map onto XML instead. In the end we may end up with style libraries coded in XML instead of TEX. But above all, we can get high quality output, using complex page models and using features provided by CONTEXT in a different way.

The process

Remark: for simplicity we use structural components that are familiar to ConTeXt *users, but in the end we may end up with more generic constructs. There are many ways leading to Rome and it would be stupid to think that we can use the same vehicle on each trip. Consider it an experiment.*

Our first document is a real simple one.

```
<document>
  <p>
    One line of text.
  </p>
</document>
```

This document can be converted into an intermediate format, where structure is replaced by more rigourous structural components. However, a paragraph of text remains what it is. Watch the definition of the layout. It looks familiar for ConTeXt users.

```
<xsl:stylesheet version='1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
xmlns:fx='http://www.pragma-ade.com/rlg/foxet.rng'>
  <xsl:output method='xml' charset='utf-8'/>
  <xsl:template name='definitions'>
    <fx:definelayout label='main' paper='A6,landscape'
    page='A6,landscape' backspace='2cm' topspace='1cm'
    width='middle' height='middle'/>
    <fx:enablelayout label='main'/>
  </xsl:template>
  <xsl:template match='document'>
    <fx:text>
      <xsl:call-template name='definitions'/>
      <xsl:apply-templates/>
    </fx:text>
  </xsl:template>
  <xsl:template match='p'>
    <fx:p strut='yes'>
      <xsl:apply-templates/>
    </fx:p>
  </xsl:template>
</xsl:stylesheet>
```

We can feed our document to XSLTPROC, using this script:

```
xsltproc --output sample-1.fx sample-1.xsl sample-1.xsl
```

This gives:

```
<fx:text xmlns:fx="http://www.pragma-ade.com/rlg/foxet.rng">
  <fx:definelayout label="main" paper="A6,landscape" page=
  "A6,landscape" backspace="2cm" topspace="1cm" width="middle"
  height="middle"/>
  <fx:enablelayout label="main"/>
  <fx:p strut="yes">
    One line of text.
  </fx:p>
</fx:text>
```

We process this with:

```
texexec --pdf --use=fx --xml sample-1.fx
```

The resulting file is shown in figure 1: indeed just a line of text.

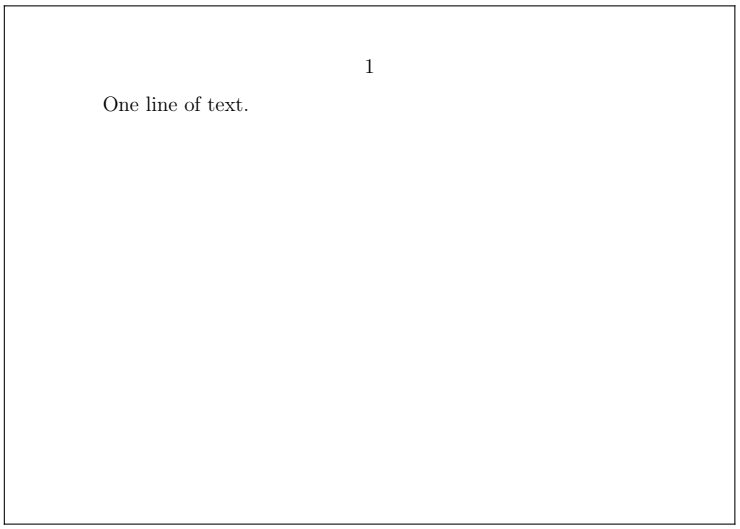

Figure 1 `sample-1.fx`

We will enhance our document with a title. This means that we need to deal with a typographical construct that needs special treatment when encountered in sequence and in relation to the following text.

```
<document>
  <title>Just a title</title>
  <p>
    One line of text.
  </p>
</document>
```

Those who have used ConTeXt for typesetting XML directly, may think of:

```
\defineXMLenvironment[document] {\starttext} {\stoptext}
\defineXMLargument    [title]    {\chapter}
```

and then use the normal definition and setup commands to tune the look and feel. This could be a problem for those who don't like backslashes, so again we will kick in XSLT.

```
<xsl:stylesheet version='1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
xmlns:fx='http://www.pragma-ade.com/rlg/foxet.rng'>
  <xsl:output method='xml' charset='utf-8'/>
  <xsl:template name='definitions'>
    <fx:definelayout label='main' paper='A6,landscape'
    page='A6,landscape' backspace='2cm' topspace='1cm'
    width='middle' height='middle'/>
    <fx:enablelayout label='main'/>
    <fx:definehead label='chapter'/>
    <fx:definelayer label='chapter' width='10cm' method='fit'/>
    <fx:definefont label='titlefont' spec='SerifBold at 24pt'/>
    <fx:definecolor label='titlecolor' r='1' g='.25' b='.5'/>
  </xsl:template>
  <xsl:template match='document'>
    <fx:text>
      <xsl:call-template name='definitions'/>
      <xsl:apply-templates/>
    </fx:text>
  </xsl:template>
  <xsl:template match='p'>
    <fx:p strut='yes'>
      <xsl:apply-templates/>
    </fx:p>
  </xsl:template>
  <xsl:template match='title'>
    <fx:page command='yes'/>
    <fx:head label='chapter'>
      <fx:setlayer label='chapter' preset='lefttop' hoffset='0cm'>
        <fx:font label='titlefont'>
          <fx:color label='titlecolor'>
            <fx:framed>
            <xsl:number/>
            </fx:framed>
          </fx:color>
        </fx:font>
      </fx:setlayer>
      <fx:setlayer label='chapter' preset='lefttop' hoffset='2cm'>
```

```
        <fx:font label='titlefont'>
          <fx:color label='titlecolor'>
            <fx:framed>
            <xsl:apply-templates/>
            </fx:framed>
          </fx:color>
        </fx:font>
      </fx:setlayer>
      <fx:flushlayer label='chapter'/>
    </fx:head>
  </xsl:template>
</xsl:stylesheet>
```

Now watch this. We don't use the ConTeXt numbering mechanism, but directly generate the numbers with XSLT! Figure 2 shows how this looks after processing. Since we're just demonstrating a way of processing XML, we don't tune spacing now.

```
<fx:text xmlns:fx="http://www.pragma-ade.com/rlg/foxet.rng">
  <fx:definelayout label="main" paper="A6,landscape" page=
  "A6,landscape" backspace="2cm" topspace="1cm" width="middle"
  height="middle"/>
  <fx:enablelayout label="main"/>
  <fx:definehead label="chapter"/>
  <fx:definelayer label="chapter" width="10cm" method="fit"/>
  <fx:definefont label="titlefont" spec="SerifBold at 24pt"/>
  <fx:definecolor label="titlecolor" r="1" g=".25" b=".5"/>
  <fx:page command="yes"/>
  <fx:head label="chapter">
    <fx:setlayer label="chapter" preset="lefttop" hoffset="0cm">
      <fx:font label="titlefont">
        <fx:color label="titlecolor">
          <fx:framed>1</fx:framed>
        </fx:color>
      </fx:font>
    </fx:setlayer>
    <fx:setlayer label="chapter" preset="lefttop" hoffset="2cm">
      <fx:font label="titlefont">
        <fx:color label="titlecolor">
          <fx:framed>Just a title</fx:framed>
        </fx:color>
      </fx:font>
    </fx:setlayer>
    <fx:flushlayer label="chapter"/>
  </fx:head>
  <fx:p strut="yes">
```

```
    One line of text.
  </fx:p>
</fx:text>
```

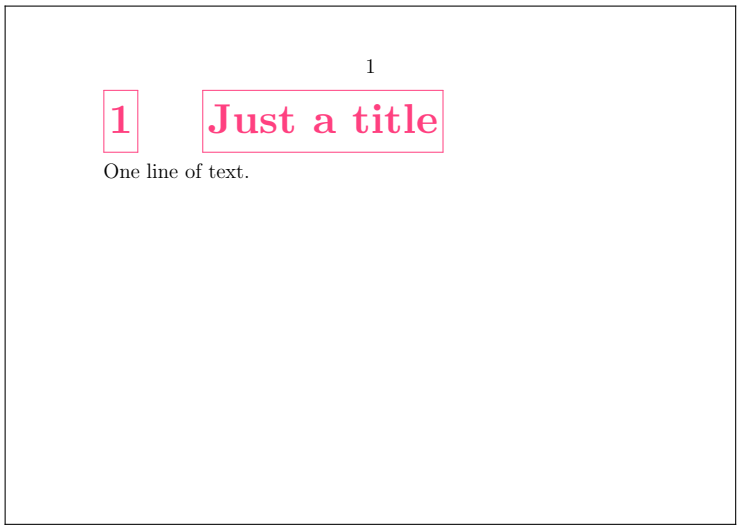

Figure 2 `sample-2.fx`

In XSL-FO we've seen such a title being generated using a item construct, a table, or just a combination of blocks. There one uses (combinations of) attributes to communicate if we're dealing with something that flows (a paragraph) or should be considered kind of fixed.

Here we use the dedicated head construct in combination with the rather fixed layers and framed constructs instead. It is no problem to isolate code and reuse it in more than one situation.

```
<xsl:stylesheet version='1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
xmlns:fx='http://www.pragma-ade.com/rlg/foxet.rng'>
  <xsl:output method='xml' charset='utf-8'/>
  <xsl:template name='definitions'>
    <fx:definelayout label='main' paper='A6,landscape'
    page='A6,landscape' backspace='2cm' topspace='1cm'
    width='middle' height='middle'/>
    <fx:definehead label='chapter'/>
    <fx:enablelayout label='main'/>
    <fx:definelayer label='chapter' width='10cm' method='fit'/>
    <fx:definefont label='titlefont' spec='SerifBold at 24pt'/>
    <fx:definecolor label='titlecolor' r='1' g='.25' b='.5'/>
  </xsl:template>
  <xsl:template name='title-type-1'>
```

```
    <xsl:param name='number'/>
    <xsl:param name='text'/>
    <fx:page command='yes'/>
    <fx:head label='chapter'>
      <fx:setlayer label='chapter' preset='lefttop' hoffset='0cm'>
        <fx:framed> <xsl:value-of select="$number"/> </fx:framed>
      </fx:setlayer>
      <fx:setlayer label='chapter' preset='lefttop' hoffset='2cm'>
        <fx:framed> <xsl:value-of select="$text"/> </fx:framed>
      </fx:setlayer>
      <fx:flushlayer label='chapter'/>
    </fx:head>
  </xsl:template>
</xsl:stylesheet>
```

The translation now looks like:

```
<xsl:stylesheet version='1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
xmlns:fx='http://www.pragma-ade.com/rlg/foxet.rng'>
  <xsl:output method='xml' charset='utf-8'/>
  <xsl:include href='sample-0.xsl'/>
  <xsl:template match='document'>
    <fx:text>
      <xsl:call-template name='definitions'/>
      <xsl:apply-templates/>
    </fx:text>
  </xsl:template>
  <xsl:template match='p'>
    <fx:p strut='yes'>
      <xsl:apply-templates/>
    </fx:p>
  </xsl:template>
  <xsl:template match='title'>
    <xsl:call-template name='title-type-1'>
      <xsl:with-param name='number'>
      <xsl:number/>
      </xsl:with-param>
      <xsl:with-param name='text'>
      <xsl:apply-templates/>
      </xsl:with-param>
    </xsl:call-template>
  </xsl:template>
</xsl:stylesheet>
```

This time we left out the colors (figure 3).

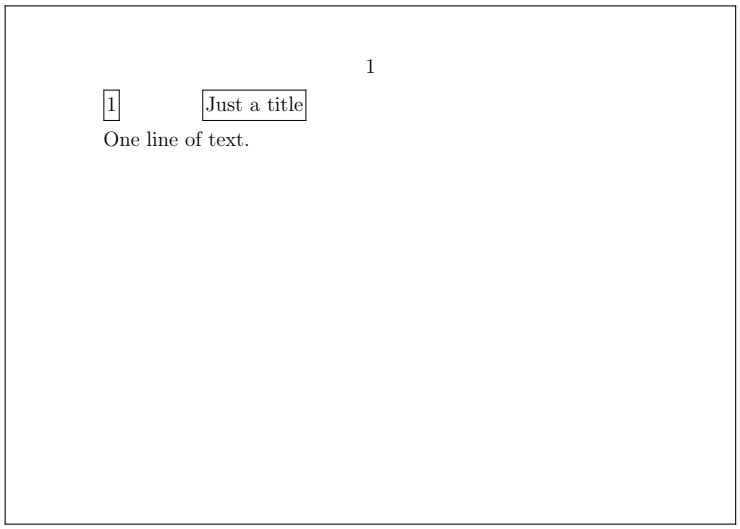

Figure 3 `sample-3.fx`

We further enhance our document, this time with some tabular information:

```
<document>
  <title>Just a title</title>
  <p>
    One line of text.
  </p>
  <addresses>
    <address>
      <name>Not Me</name>
      <street>Nice Lane</street>
      <number>2</number>
      <city>Smalltown</city>
    </address>
    <address>
      <name>Just You</name>
      <street>Famous Street</street>
      <number>73</number>
      <city>Worldcity</city>
    </address>
  </addresses>
</document>
```

In ConTeXt we have several mechanisms for dealing with tables, and one of them is well suited for tabular information that ends up in the text flow. Althoug it is technically possible to combine all specific table features in one mechanism, it is pretty hard to let it give the desired output without the user providing clues. Therefore it's more convenient to use dedicated mechanisms. (Also, using dedicated

mechanisms is less error prone due to changes and updates in the table handler. The way web browsers deal with tables over time demonstrates that it's far from trivial to suit all needs.)

```
<xsl:stylesheet version='1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
xmlns:fx='http://www.pragma-ade.com/rlg/foxet.rng'>
  <xsl:output method='xml' charset='utf-8'/>
  <xsl:include href='sample-0.xsl'/>
  <xsl:template match='document'>
    <fx:text>
      <xsl:call-template name='definitions'/>
      <xsl:apply-templates/>
    </fx:text>
  </xsl:template>
  <xsl:template match='p'>
    <fx:p strut='yes'>
      <xsl:apply-templates/>
    </fx:p>
  </xsl:template>
  <xsl:template match='subtitle'>
    <fx:margintext>
    <xsl:apply-templates/>
    </fx:margintext>
  </xsl:template>
  <xsl:template match='title'>
    <xsl:call-template name='title-type-1'>
      <xsl:with-param name='number'>
      <xsl:number/>
      </xsl:with-param>
      <xsl:with-param name='text'>
      <xsl:apply-templates/>
      </xsl:with-param>
    </xsl:call-template>
  </xsl:template>
  <xsl:template match='addresses'>
    <fx:tabulate>
      <fx:preamble>
        <fx:e align='flushleft'/>
        <fx:e align='flushleft' type='paragraph'/>
        <fx:e align='flushright'/>
        <fx:e align='flushleft'/>
      </fx:preamble>
      <fx:tbody>
        <xsl:apply-templates/>
```

```
      </fx:tbody>
    </fx:tabulate>
  </xsl:template>
  <xsl:template match='addresses/address'>
    <fx:tr>
      <fx:td><xsl:value-of select='name'/></fx:td>
      <fx:td><xsl:value-of select='street'/></fx:td>
      <fx:td><xsl:value-of select='number'/></fx:td>
      <fx:td><xsl:value-of select='city'/></fx:td>
    </fx:tr>
  </xsl:template>
</xsl:stylesheet>
```

The result is shown below and in Figure 4.

```
<fx:text xmlns:fx="http://www.pragma-ade.com/rlg/foxet.rng">
  <fx:definelayout label="main" paper="A6,landscape" page=
  "A6,landscape" backspace="2cm" topspace="1cm" width="middle"
  height="middle"/>
  <fx:definehead label="chapter"/>
  <fx:enablelayout label="main"/>
  <fx:definelayer label="chapter" width="10cm" method="fit"/>
  <fx:definefont label="titlefont" spec="SerifBold at 24pt"/>
  <fx:definecolor label="titlecolor" r="1" g=".25" b=".5"/>
  <fx:page command="yes"/>
  <fx:head label="chapter">
    <fx:setlayer label="chapter" preset="lefttop" hoffset="0cm">
      <fx:framed>1</fx:framed>
    </fx:setlayer>
    <fx:setlayer label="chapter" preset="lefttop" hoffset="2cm">
      <fx:framed>Just a title</fx:framed>
    </fx:setlayer>
    <fx:flushlayer label="chapter"/>
  </fx:head>
  <fx:p strut="yes">
    One line of text.
  </fx:p>
  <fx:tabulate>
    <fx:preamble>
      <fx:e align="flushleft"/>
      <fx:e align="flushleft" type="paragraph"/>
      <fx:e align="flushright"/>
      <fx:e align="flushleft"/>
    </fx:preamble>
    <fx:tbody>
```

```
      <fx:tr>
        <fx:td>Not Me</fx:td>
        <fx:td>Nice Lane</fx:td>
        <fx:td>2</fx:td>
        <fx:td>Smalltown</fx:td>
      </fx:tr>
      <fx:tr>
        <fx:td>Just You</fx:td>
        <fx:td>Famous Street</fx:td>
        <fx:td>73</fx:td>
        <fx:td>Worldcity</fx:td>
      </fx:tr>
    </fx:tbody>
  </fx:tabulate>
</fx:text>
```

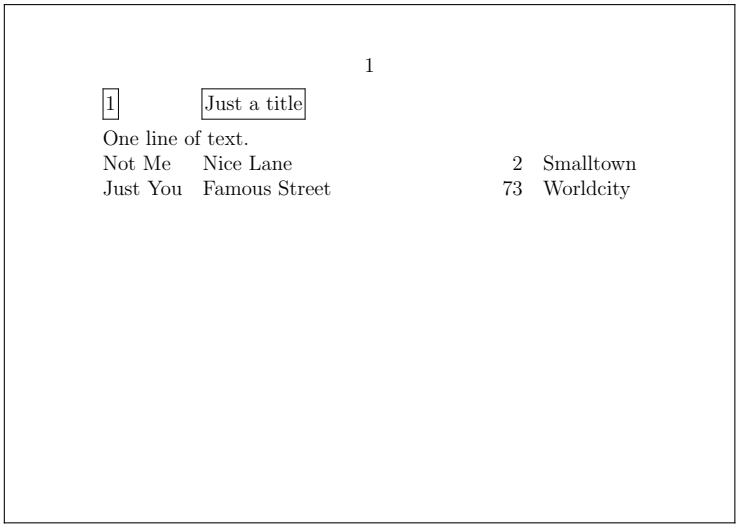

Figure 4 `sample-4.fx`

An alternative mapping can be:

```
<xsl:stylesheet version='1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
xmlns:fx='http://www.pragma-ade.com/rlg/foxet.rng'>
  <xsl:output method='xml' charset='utf-8'/>
  <xsl:include href='sample-0.xsl'/>
  <xsl:template match='document'>
    <fx:text>
      <xsl:call-template name='definitions'/>
      <xsl:apply-templates/>
    </fx:text>
```

```
  </xsl:template>
  <xsl:template match='p'>
    <fx:p strut='yes'>
      <xsl:apply-templates/>
    </fx:p>
  </xsl:template>
  <xsl:template match='subtitle'>
    <fx:margintext>
    <xsl:apply-templates/>
    </fx:margintext>
  </xsl:template>
  <xsl:template match='title'>
    <xsl:call-template name='title-type-1'>
      <xsl:with-param name='number'>
      <xsl:number/>
      </xsl:with-param>
      <xsl:with-param name='text'>
      <xsl:apply-templates/>
      </xsl:with-param>
    </xsl:call-template>
  </xsl:template>
  <xsl:template match='addresses'>
    <fx:table>
      <fx:tbody>
        <xsl:apply-templates/>
      </fx:tbody>
    </fx:table>
  </xsl:template>
  <xsl:template match='addresses/address'>
    <fx:tr>
      <fx:td><xsl:value-of select='name'/></fx:td>
      <fx:td><xsl:value-of select='street'/></fx:td>
      <fx:td><xsl:value-of select='number'/></fx:td>
      <fx:td><xsl:value-of select='city'/></fx:td>
    </fx:tr>
  </xsl:template>
</xsl:stylesheet>
```

This becomes:

```
<fx:text xmlns:fx="http://www.pragma-ade.com/rlg/foxet.rng">
  <fx:definelayout label="main" paper="A6,landscape" page=
  "A6,landscape" backspace="2cm" topspace="1cm" width="middle"
  height="middle"/>
  <fx:definehead label="chapter"/>
```

```
  <fx:enablelayout label="main"/>
  <fx:definelayer label="chapter" width="10cm" method="fit"/>
  <fx:definefont label="titlefont" spec="SerifBold at 24pt"/>
  <fx:definecolor label="titlecolor" r="1" g=".25" b=".5"/>
  <fx:page command="yes"/>
  <fx:head label="chapter">
    <fx:setlayer label="chapter" preset="lefttop" hoffset="0cm">
      <fx:framed>1</fx:framed>
    </fx:setlayer>
    <fx:setlayer label="chapter" preset="lefttop" hoffset="2cm">
      <fx:framed>Just a title</fx:framed>
    </fx:setlayer>
    <fx:flushlayer label="chapter"/>
  </fx:head>
  <fx:p strut="yes">
    One line of text.
  </fx:p>
  <fx:table>
    <fx:tbody>
      <fx:tr>
        <fx:td>Not Me</fx:td>
        <fx:td>Nice Lane</fx:td>
        <fx:td>2</fx:td>
        <fx:td>Smalltown</fx:td>
      </fx:tr>
      <fx:tr>
        <fx:td>Just You</fx:td>
        <fx:td>Famous Street</fx:td>
        <fx:td>73</fx:td>
        <fx:td>Worldcity</fx:td>
      </fx:tr>
    </fx:tbody>
  </fx:table>
</fx:text>
```

And shows up as in Figure 5.

As said, the tabulate mechanism is well suited for tabular information that is part of the text flow. This is quite noticable when one has to deal with typesetting on a grid.

Our last example concerns another sometimes tricky feature, marginal notes.

```
<document>
  <title>Just a title</title>
  <subtitle>a short subtitle</subtitle>
  <p>
```

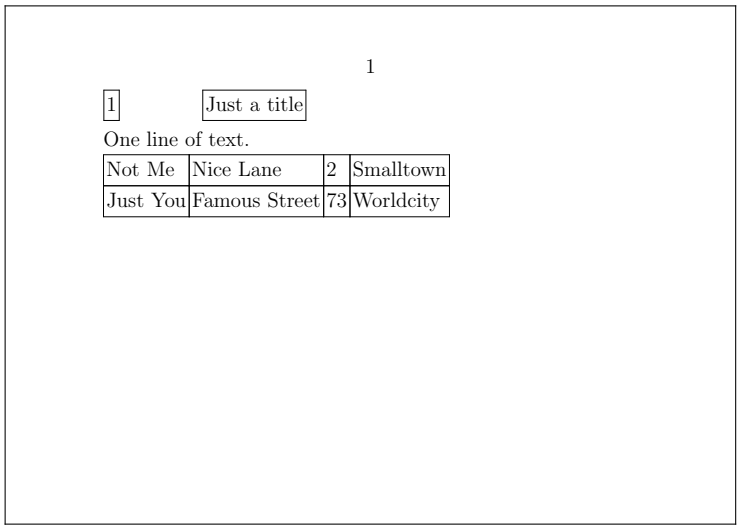

Figure 5 `sample-5.fx`

```
    One line of text.
  </p>
  <addresses>
    <address>
      <name>Not Me</name>
      <street>Nice Lane</street>
      <number>2</number>
      <city>Smalltown</city>
    </address>
    <address>
      <name>Just You</name>
      <street>Famous Street</street>
      <number>73</number>
      <city>Worldcity</city>
    </address>
  </addresses>
</document>
```

The mapping is simple:

```
<xsl:stylesheet version='1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
xmlns:fx='http://www.pragma-ade.com/rlg/foxet.rng'>
  <xsl:output method='xml' charset='utf-8'/>
  <xsl:include href='sample-0.xsl'/>
  <xsl:template match='document'>
    <fx:text>
      <xsl:call-template name='definitions'/>
```

```
    <xsl:apply-templates/>
  </fx:text>
</xsl:template>
<xsl:template match='p'>
  <fx:p strut='yes'>
    <xsl:apply-templates/>
  </fx:p>
</xsl:template>
<xsl:template match='subtitle'>
  <fx:margintext>
  <xsl:apply-templates/>
  </fx:margintext>
</xsl:template>
<xsl:template match='title'>
  <xsl:call-template name='title-type-1'>
    <xsl:with-param name='number'>
    <xsl:number/>
    </xsl:with-param>
    <xsl:with-param name='text'>
    <xsl:apply-templates/>
    </xsl:with-param>
  </xsl:call-template>
</xsl:template>
<xsl:template match='addresses'>
  <fx:tabulate>
    <fx:preamble>
      <fx:e align='flushleft'/>
      <fx:e align='flushleft' type='paragraph'/>
      <fx:e align='flushright'/>
      <fx:e align='flushleft'/>
    </fx:preamble>
    <fx:tbody>
      <xsl:apply-templates/>
    </fx:tbody>
  </fx:tabulate>
</xsl:template>
<xsl:template match='addresses/address'>
  <fx:tr>
    <fx:td><xsl:value-of select='name'/></fx:td>
    <fx:td><xsl:value-of select='street'/></fx:td>
    <fx:td><xsl:value-of select='number'/></fx:td>
    <fx:td><xsl:value-of select='city'/></fx:td>
  </fx:tr>
</xsl:template>
```

```
</xsl:stylesheet>
```

And the result looks familiar as well:

```
<fx:text xmlns:fx="http://www.pragma-ade.com/rlg/foxet.rng">
  <fx:definelayout label="main" paper="A6,landscape" page=
  "A6,landscape" backspace="2cm" topspace="1cm" width="middle"
  height="middle"/>
  <fx:definehead label="chapter"/>
  <fx:enablelayout label="main"/>
  <fx:definelayer label="chapter" width="10cm" method="fit"/>
  <fx:definefont label="titlefont" spec="SerifBold at 24pt"/>
  <fx:definecolor label="titlecolor" r="1" g=".25" b=".5"/>
  <fx:page command="yes"/>
  <fx:head label="chapter">
    <fx:setlayer label="chapter" preset="lefttop" hoffset="0cm">
      <fx:framed>1</fx:framed>
    </fx:setlayer>
    <fx:setlayer label="chapter" preset="lefttop" hoffset="2cm">
      <fx:framed>Just a title</fx:framed>
    </fx:setlayer>
    <fx:flushlayer label="chapter"/>
  </fx:head>
  <fx:margintext>a short subtitle</fx:margintext>
  <fx:p strut="yes">
    One line of text.
  </fx:p>
  <fx:tabulate>
    <fx:preamble>
      <fx:e align="flushleft"/>
      <fx:e align="flushleft" type="paragraph"/>
      <fx:e align="flushright"/>
      <fx:e align="flushleft"/>
    </fx:preamble>
    <fx:tbody>
      <fx:tr>
        <fx:td>Not Me</fx:td>
        <fx:td>Nice Lane</fx:td>
        <fx:td>2</fx:td>
        <fx:td>Smalltown</fx:td>
      </fx:tr>
      <fx:tr>
        <fx:td>Just You</fx:td>
        <fx:td>Famous Street</fx:td>
        <fx:td>73</fx:td>
```

```
        <fx:td>Worldcity</fx:td>
      </fx:tr>
    </fx:tbody>
  </fx:tabulate>
</fx:text>
```

And figure 6 shows that the text shows up in the margin indeed.

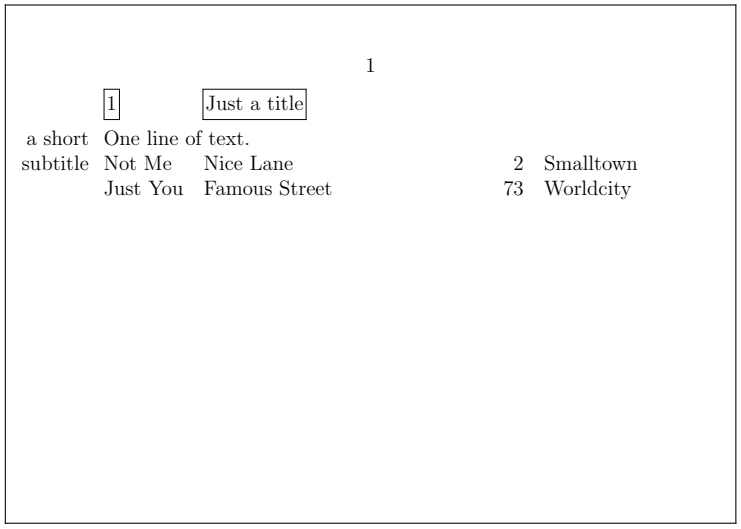

Figure 6 `sample-6.fx`

If you know a bit of ConTeXt, you can imagine that processing such files is rather trivial. Even more interesting is that one has the full machinery available as well. However, the idea is that we limit ourselves to only part of ConTeXt's functionality and let XSLT do the dirty work. The code shown here is just the result of an simple experiment and should be considered as such. However, depending on user demand and input, we may end up with a rather full blown implementation. Just to give you an idea of how this is done:

```
\defineXMLenvironment
  [fx:framed] [fox] []
  {\initializefox
   \expanded{\framed[\foxarguments]}\bgroup\ignorespaces}
  {\removeunwantedspaces\egroup}
```

with a few definition common to such mappings:

```
\def\foxarguments
  {\theXMLarguments{fox}}
\def\initializefox
  {\expanded{\getrawparameters[fox][\theXMLarguments{fox}]}}
```

We realized that some constructs may demand extensions to CONTEXT, for instance, it makes no sense to use the current head mechanisms for some related XML variant. On the other hand, layers and such can be used directly, and some —yet undocumented but rather interesting— other mechanisms as well. Another topic for discussion is how to deal with defaults, for instance those used in the standard layout definition. We may as well start with no defaults at all. The same is true for handling page numbers, headers and footers. In the end it may even result in a lightweight version of CONTEXT. Fonts are yet another matter. Several methods need to be supported there, if only because we want the goodies that PDFTEX and XETEX provide.

How well this can be used in combination with XSL-FO is to be seen. In principle combining the technologies is possible, but it may be needed to disable some of the XSL-FO features that spoil the game.

This means that currently we have the following methods (either or not used in combination) to go from XML to a typeset document:

- using the builtin XML parser to map onto CONTEXT code
- using XSLT to map onto CONTEXT TEX code
- using XSLT to map onto CONTEXT XML code
- using XSLT to produce the XSL-FO as accepted by CONTEXT

In the third case we do a rather direct mapping, and so we benefit from the interface documentation already available. If this method, of which we demonstrated an example here, makes sense to users, we may even end up with a large repository of transformation scripts for common DTD's and interesting layouts. The challenge is to determine what functionality is needed, so . . . use the CONTEXT mailing list to let your voice be heard.