

# Processing “Computed” Texts

Jean-Michel Hufflen

## Sommario

This article is a comparison among methods that may be used to derive texts to be typeset by a word processor. By ‘derive’, we mean that such texts are extracted from a larger structure. The present standard for such a structure uses XML-like format, and we give an overview of the available tools for this derivation task.

**Keywords** Typesetting computed texts,  $\text{T}_{\text{E}}\text{X}$ ,  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ ,  $\text{ConT}_{\text{E}}\text{Xt}$ ,  $\text{X}_{\text{Q}}\text{T}_{\text{E}}\text{X}$ ,  $\text{LuaT}_{\text{E}}\text{X}$ , XML, XSLT, character maps, XQuery, XSL-FO.

## Sommario

Questo articolo è una comparazione dei metodi che possono essere usati per derivare, tramite un *word processor*, testi da comporre. Per ‘derivare’ intendiamo che questi testi saranno estratti da una struttura più grande. Lo standard corrente per tale struttura usa un formato simile a XML, e daremo una panoramica degli strumenti disponibili per la derivazione.

**Parole chiave** Composizione di testi elaborati,  $\text{T}_{\text{E}}\text{X}$ ,  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ ,  $\text{ConT}_{\text{E}}\text{Xt}$ ,  $\text{X}_{\text{Q}}\text{T}_{\text{E}}\text{X}$ ,  $\text{LuaT}_{\text{E}}\text{X}$ , XML, XSLT, mappe di caratteri, XQuery, XSL-FO.

## 1 Introduction

Formats based on the  $\text{T}_{\text{E}}\text{X}$  typesetting engine—e.g., *Plain T<sub>E</sub>X* (Knuth, 1984), or  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  (Lamport, 1994), or  $\text{ConT}_{\text{E}}\text{Xt}$  (Hagen, 2001)—are known as wonderful tools to get high-quality print outputs. Of course, they have been designed to typeset texts directly written by end-users, at first. But other texts may be *computed*, in the sense that they result from computation applied to some data, in particular, items belonging to data bases. A very simple example is given by a bill computed by means of a spreadsheet program like Microsoft Excel: the master file is an .xls file—that is, the whole information about data is centralised into this .xls file—but we may wish such a bill to be typeset nicely using a word processor comparable with  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ . We personally experienced a more significant example: at the University of Franche-Comté, we manage the projects proposed to Computer Science students in several degrees located at Besançon, in the East of France. That is, we collect the proposals of projects, control the assignment of student groups to projects. Then, at the semester’s end, we organise the projects’ oral defences, and rate students from information transmitted by juries. During projects, information is transmitted

to students and projects’ supervisors either on the Web, or by means of printed documents. Managing only a list of project specifications and enriching it progressively is insufficient: it is better for oral defences’ announcement to be shown w.r.t. defences’ chronological order, and this order is unknown when projects are proposed. Likewise, we may wish to give the grades got by students according to the decreasing order of these grades, as well as students’ alphabetical order. In these cases, we have to perform a sort operation before typesetting the result. These examples are not limitative: other operations may be needed if we are only interested in a subset of the information concerning projects.

It is well-known that  $\text{T}_{\text{E}}\text{X}$ ’s language is not very suitable for tasks directly related to programming. Readers interested in putting a sort procedure into action using this language can refer to (Roegel, 1998): this *modus operandi* may be viewed as a worthwhile exercise, but is unusable in practice, especially when it is not immediate to get sort keys from items to be sorted. A better idea is to use a format suitable for information management, with interface tools serving several purposes. Presently, the indisputable standard to model such formats is XML<sup>1</sup>, providing a rich toolbox for this kind of task. But these tools have advantages and drawbacks: we are going thoroughly into these points in this article. Reading it requires only basic knowledge about ( $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  and the tools related to XML.

## 2 A simple example

The examples given in the introduction are ‘real’ applications, but for sake of simplicity, we consider an easier example<sup>2</sup>, pictured in Figure 1. This XML text describes some items of a series of stories—*Doc Savage*—first published as pulps in the 1930’s, then republished as pocket books in the 1960’s<sup>3</sup>. As it can be noticed in the given example, the original publication order—for pulps—was not followed by the series of pocket books. In addition, some stories were unpublished as pulps (e.g., *The Red Spider*) or retitled when published as pocket books (e.g., *The Deadly Dwarf*, previously titled *Repel*), in which

1. eXtensible Markup Language. Readers interested in a general introductory book to this formalism can refer to (Ray, 2001).

2. Concerning the first example we have mentioned—the example related to a spreadsheet program—readers interested in it can refer to (Boulanger, 2001).

3. All the source texts mentioned throughout this article can be downloaded *in extenso* from <http://lifc.univ-fcomte.fr/home/~jmhufflen/texts/>.

```

<story-list>
  <story>
    <title>The Deadly Dwarf</title>
    <pulp nb="56">Repel</pulp>
    <pocket-book nb="28"/>
  </story>
  <story>
    <title>The Land of Terror</title>
    <pulp nb="2"/>
    <pocket-book nb="8"/>
  </story>
  <story>
    <title>The Lost Oasis</title>
    <pulp nb="7"/>
    <pocket-book nb="6"/>
  </story>
  <story>
    <title>The Man of Bronze</title>
    <pulp nb="1"/>
    <pocket-book nb="1"/>
  </story>
  <story>
    <title>The Red Spider</title>
    <pocket-book nb="95"/>
  </story>
  <story>
    <title>World's Fair Goblin</title>
    <pulp nb="74"/>
    <pocket-book nb="39"/>
  </story>
</story-list>

```

FIGURE 1: Master file using XML-like syntax.

case, the pulp’s title is given as the contents of the pulp element<sup>4</sup>. More precisely, if the contents of a pulp element is empty, that means that the pulp’s title was the same as the pocket book’s.

Now we propose to search the information given in Figure 1, extract the items published as pulps, sorts them according to the publication order<sup>5</sup>. The title given is the pulp’s; if the corresponding pocket book has been retitled, a footnote must give the ‘new’ title. Of course, we wish the result to be typeset nicely, as (LA)TEX is able to do. That is, a good solution should look like the source text, processable by *Plain TEX*, given in Figure 2. As above-mentioned, a TEX-based solution could use TEX commands for the elements `story-list`,

4. In this case, such a story is more commonly known under the pocket book’s title, because pocket books are easier to get than pulps, very rare. That is why our title elements always refer to pocket books’, the contents of `pocket-book` elements being always empty. The Web page mentioned above includes specifications of the taxonomy of these texts.

5. This example seems to us to be pertinent, because there is no order ‘better’ than others: the original order is based on pulps, but—as above-mentioned—some stories are unpublished as pulps, whereas sorting stories according to pocket books’ order allows us to sort all the stories, but this is not really chronological.

```

\item{1} The Man of Bronze
\item{2} The Land of Terror
\item{7} The Lost Oasis
\item{56} Repel\footnote*{Book’s title of
  pulp no.~56: The Deadly Dwarf}
\item{74} World’s Fair Goblin

\end

```

FIGURE 2: Stories’ titles sorted by pulp numbers.

`story`, `title`, `pulp`, and `pocket-book`, but would lead to complicated programming.

### 3 Using tools related to xml

#### 3.1 xslt producing TEX sources

XSLT<sup>6</sup> (W3C, 2007b) is the language commonly used for transformations of XML texts. The text of a stylesheet that may be used to get Figure 2 from Figure 1 is given *in extenso* in Figure 3. This stylesheet takes as much advantage as possible of the features introduced by XPath’s<sup>7</sup> and XSLT’s new version<sup>8</sup> (2.0): for example, we have made precise the types of the used variables, by means of `as` attributes—these types being provided by the library of XML Schema<sup>9</sup> (W3C, 2008)—such information allows an XSLT processor to perform type-checking. Likewise, we have made precise the type of templates’s results, as far as possible.

Let us remark that this stylesheet implicitly uses the fact that the strings of our input file—the elements’ contents—do not contain any special character of TEX: for example, a ‘#’ character must be replaced by ‘\#’ in order to be processed correctly by TEX. If we are interested in building (LA)TEX texts, *character maps* (W3C, 2007b, § 20.1) allow the replacement of a single character by a string, as shown in Figure 4 where we show how to process most of *Plain TEX*’s special characters. Another solution may be the generation of Unicode texts (The UNICODE CONSORTIUM, 2006) and the use of a Unicode-compliant TEX-like engine<sup>10</sup>, e.g. XTTEX

6. eXtensible Stylesheet Language Transformations. Introductions to this language have been given in BachoTEX conferences: (Hufflen, 2005, 2006, 2008a).

7. XPath (W3C, 2007a) is the language used to address parts of an XML text.

8. A study of these new features can be found in Hufflen (2008a).

9. Schemas allow users to define *document types*, such a document type can be viewed as taxonomy common to some XML texts. There exist several schema languages, but only XML Schema is interfaced with XSLT 2.0. The prefix ‘`xsd:`’ gets access to XML Schema’s library, whereas the prefix ‘`xs1:`’ characterises XSLT constructs. Readers interested in more details about XML namespaces can consult (Ray, 2001, pp. 41–45). An example using XML Schema is given at the Web page mentioned above.

10. Engines are not formats: a *format* is a set of pre-loaded definitions based on primitives of a TEX-like engine, whereas an *engine* is TEX or is derived from TEX by adding

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="2.0" id="pulp" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" extension-element-prefixes="xsd">
  <xsl:output method="text" encoding="ISO-8859-1"/>
  <xsl:strip-space elements="*" />
  <!-- (The best way to specify an End-Of-Line character:) -->
  <xsl:variable name="eol" select="'&#xA;'" as="xsd:string"/>
  <xsl:template match="story-list">
    <xsl:apply-templates select="story[pulp]">
      <xsl:sort select="xsd:integer(pulp/@nb)"/> <!-- Numerical sort. -->
    </xsl:apply-templates>
    <xsl:value-of select="$eol,'&end',$eol" separator=""/>
  </xsl:template>
  <xsl:template match="story">
    <xsl:variable name="pulp-0" select="pulp" as="element(pulp)"/>
    <xsl:variable name="pulp-nb-0" select="$pulp-0/@nb" as="xsd:integer"/>
    <xsl:variable name="pulp-title-0" select="data(pulp-0)" as="xsd:string"/>
    <xsl:variable name="title-processed" as="xsd:string">
      <xsl:apply-templates select="title"/>
    </xsl:variable>
    <xsl:value-of
      select="'\item{",$pulp-nb-0,"} ",
        if ($pulp-title-0) then
          $pulp-title-0,"\footnote*{Book&apos;s title of pulp no.~",$pulp-nb-0," : ",
            $title-processed,"}" else
            $title-processed,
          $eol'
      separator=""/>
    </xsl:template>
    <xsl:template match="title" as="xsd:string"><xsl:apply-templates/></xsl:template>
  </xsl:stylesheet>

```

FIGURE 3: Getting a source text for *Plain T<sub>E</sub>X* by means of an XSLT stylesheet.

(Kew, 2007) or LuaT<sub>E</sub>X (Hagen, 2008a).

XSLT is not limited to the generation of texts, the `method` attribute may also be set to `html` or `xhtml`<sup>11</sup> (W3C, 2007b, § 20), in which case it allows Web pages to be generated. Likewise, this `method` attribute set to `xml` means that XML texts are to be generated. Using these output methods provides a great advantage: since any XSLT stylesheet is an XML text, an XSLT processor checks that the final document is well-formed, in particular, opening and closing tags must be balanced. The generation of (L<sup>A</sup>)T<sub>E</sub>X texts lacks analogous check: an XSLT processor cannot ensure that opening and closing braces are balanced, as in T<sub>E</sub>X; likewise, when L<sup>A</sup>T<sub>E</sub>X texts are generated, it is impossible to ensure that environments—e.g., `\begin{document}... \end{document}`—are bal-

or redefining some primitives. *Plain T<sub>E</sub>X* and L<sup>A</sup>T<sub>E</sub>X are formats, X<sub>Y</sub>T<sub>E</sub>X and LuaT<sub>E</sub>X are engines.

11. (eXtensible) HyperText Markup Language. XHTML is a reformulation of HTML—the original language of *Web* pages—using XML conventions. (Musciano and Kennedy, 2002) is a good introduction to these languages.

anced. Since such errors are not detected statically, they just appear when generated texts are typeset by a word processor. Let us notice that such check would be more difficult about the texts generated for the ConT<sub>E</sub>Xt format, because the opening and closing commands for ConT<sub>E</sub>Xt's environments are `\start...` and `\stop...`, e.g., the equivalent formulation for `\begin{document}... \end{document}` in L<sup>A</sup>T<sub>E</sub>X is `\starttext... \stoptext` in ConT<sub>E</sub>Xt.

### 3.2 xslt producing xsl-fo texts

Since deriving XML texts by means of XSLT provides a better check level, an alternative idea is to get XSL-FO<sup>12</sup> (W3C, 2006) texts. Such texts use an XML-like syntax that aim to describe high-quality print outputs. As shown in (Hutfien, 2007), there are some similarities between L<sup>A</sup>T<sub>E</sub>X and XSL-FO, the latter providing, of course, more systematic markup. This language is verbose, but it is not devoted to direct use: XSL-FO texts usually result

12. eXtensible Stylesheet Language—Formatting Objects.

```
<xsl:character-map name="some-special-characters">
  <xsl:output-character character="#" string="\#"/>
  <xsl:output-character character="%" string="\%"/>
  <xsl:output-character character="$" string="\$"/>
  <xsl:output-character character="&" string="\&"/>      <!-- &  -->
  <xsl:output-character character="\ " string="\backslash"/>
  <xsl:output-character character="{ " string="\{">      <!-- In Plain TeX, the commands  -->
  <xsl:output-character character="}" string="\}">      <!-- \{ and \} are only usable in  -->
  <!-- in math mode (cf. (Knuth, 1984, Exercise 16. 12)).  -->
  <xsl:output-character character="~" string="\char'7E"/> <!-- Using hexadecimal code.  -->
</xsl:character-map>
```

FIGURE 4: Using a character map in XSLT 2.0.

from applying an XSLT stylesheet<sup>13</sup>.

The advantage of this approach is clear: generated texts in XSL-FO are well-formed. However, XSL-FO lacks *document classes*, as in L<sup>A</sup>T<sub>E</sub>X. Some elements allow the description of *page models*, but end-users are entirely responsible for this definition. XSL-FO provides much expressive power about placement of *blocks*<sup>14</sup>, but is very basic on other points. For example, let us consider footnotes, end-users are responsible of choosing footnote marks. Figure 5 gives the result of applying an XSLT stylesheet providing an XSL-FO text for our example. The first child of the `fo:footnote` element gives the footnote reference, the second child is the actual footnote’s contents (W3C, 2006, § 6.12.3). This `fo:footnote` element seems to be low-level in comparison with L<sup>A</sup>T<sub>E</sub>X’s `\footnote` command<sup>15</sup>. Of course, if you want footnotes to be numbered automatically, XSLT addresses this problem. However, another point seems to us to be more debatable: end-users are responsible for putting a leader separating footnotes from the text’s body<sup>16</sup> (we show how to proceed in Figure 5). So some footnotes may be preceded by a leader, some may not. This point may seem anecdotal, but for L<sup>A</sup>T<sub>E</sub>X users, some features of XSL-FO can be viewed as low-level and be difficult to handle.

Last but not least, most present XSL-FO processors do not implement the whole of this language, even if they can successfully process most of XSL-FO texts used in practice<sup>17</sup>. So some features may be unusable, whereas an equivalent construct will work in (L<sup>A</sup>)T<sub>E</sub>X. XSL-FO has been designed to deal with the whole of Unicode, so it shows how the

Unicode bidirectional algorithm (UNICODE CONSORTIUM, 2008) is put into action (Hufflen, 2009), but this point may also be observed with X<sub>F</sub>T<sub>E</sub>X.

### 3.3 XQuery producing T<sub>E</sub>X sources

XQuery (W3C, 2007c) is a query language for data stored in XML form, as SQL<sup>18</sup> does for relational data bases<sup>19</sup>. XQuery can be used to search documents and arrange the result, as an XML structure or a simple text (possibly suitable for a T<sub>E</sub>X-like engine). An XQuery program processing our example in order to get Figure 2’s text is given in Figure 6. Like XSLT 2.0, XQuery uses XPath 2.0 expressions and the datatype library provided by XML Schema<sup>20</sup>. Such programs, using FLWOR<sup>21</sup> expressions, are more compact than equivalent ones in XSLT. However, XQuery is suitable only for generating simple texts: advanced features like character maps in XSLT are provided by some XQuery processors, but are not portable. A simple example of an implementation-dependent feature is given in Figure 6: we declare that the result is not an XML text by a non-portable option, `saxon:output`<sup>22</sup>. Of course, if XQuery is used to generate XML texts, they are well-formed, but no analogous check can be done about texts generated for a T<sub>E</sub>X-like engine. In other words, XQuery has the same drawback than XSLT.

### 3.4 A curiosity: dsssl

DSSSL<sup>23</sup> (dsssl, 1996) was initially designed as the stylesheet language for displaying SGML<sup>24</sup> texts.

18. **Structured Query Language**. A good introductory book about it is (Melton and Simon, 1993).

19. A short introduction to XQuery is given in (?).

20. ... although the type declarations using the `as` keyword are optional. We put them in Figure 6 for sake of clarity and for taking as much advantage as possible of XQuery’s type-checker.

21. ‘For, Let, Where, Order by, Return’, the keywords used throughout such expressions.

22. The XQuery processor we have used for this example is *Saxon* (Kay, 2008).

23. **Document Style Semantics Specification Language**.

24. **Standard Generalised Markup Language**. Now it is only of a historical interest. Readers interested in this metalanguage can refer to (Bradley, 1997).

13. The use of several prefixes for namespaces—usually ‘`xsl:`’ and ‘`fo:`’—clearly distinguish elements belonging to XSLT and XSL-FO.

14. Roughly speaking a *block* in XSL-FO is analogous to a *minipage* in L<sup>A</sup>T<sub>E</sub>X (Hufflen, 2007, § 1.2).

15. In fact, XSL-FO’s footnotes can be compared with *Plain TeX*’s `\footnote` command, as shown in Figure 2.

16. That is done in standard L<sup>A</sup>T<sub>E</sub>X class, but not universal: as an example, the *arstexnica* class, used for articles of the *ArXiv* journal, does not put it.

17. We personally use the **A**pache **F**OP (**F**ormating **O**bjects **P**rocessor) (`apachefop`).

```

<fo:list-block ...>
  ...
  <fo:list-item>
    <fo:list-item-label ...><fo:block>56</fo:block></fo:list-item-label>
    <fo:list-item-body ...>
      <fo:block>
        Repel<fo:footnote>
          <fo:inline font-size="x-small" vertical-align="super">*</fo:inline>
          <fo:footnote-body>
            <fo:block text-align-last="justify"><fo:leader leader-pattern="rule"/></fo:block>
            <fo:block font-size="'xx-small'">
              <fo:inline font-size="xx-small" vertical-align="super">*</fo:inline>
              Book's title for pulp no. 56: The Deadly Dwarf
            </fo:block>
          </fo:footnote-body>
        </fo:footnote>
      </fo:block>
    </fo:list-item-body>
  </fo:list-item>
  ...
</fo:list-block>

```

---

FIGURE 5: How to put a footnote in XSL-FO (see the equivalent *Plain T<sub>E</sub>X* source text in Figure 2).

DSSSL includes a core expression language that is a side-effect free subset of the Scheme programming language (Kelsey et al., 1998). XML being a subset of SGML, stylesheets written using DSSSL can be applied to XML texts. DSSSL is rarely used now, that is why we just cite it<sup>25</sup>, an example of using it for a bibliography style can be found in (Hufften, 2008b, § 3.5). End-users do not put down (L<sup>A</sup>)T<sub>E</sub>X commands when they develop a DSSSL stylesheet, but the *jade*<sup>26</sup> program, as shown in (Goossens et al., 1999, § 7.5.2), can generate T<sub>E</sub>X-like texts, the typesetting engine usable to process such results being JadeT<sub>E</sub>X.

## 4 Enriched T<sub>E</sub>X engines

If we go back to programs based on T<sub>E</sub>X-like typesetting engines, there are two other possible methods, based on T<sub>E</sub>X-engines ‘enriched’ by using a ‘more classical’ programming language. In both cases, XML texts are pre-processed by procedures belonging to a programming language, and the result is sent to a T<sub>E</sub>X-like typesetting engine. Of course, such *modus operandi* is suitable only if we want to generate (L<sup>A</sup>)T<sub>E</sub>X texts, it would be of little interest to get XML texts or pages written using (X)HTML<sup>27</sup>.

PyT<sub>E</sub>X (Fine, 2005) is written using Python (Martelli, 2006) and uses T<sub>E</sub>X as a daemon. Getting the components of a ‘computed’ text is left to Python—more precisely, to the Python functions

25. A DSSSL stylesheet generating a text equivalent to Figure 2’s is given at the Web page mentioned above.

26. James Clark’s Awesome DSSSL Engine.

27. Unless a converter from (L<sup>A</sup>)T<sub>E</sub>X to (X)HTML is used, of course.

dealing with XML texts—and successive results are sent to T<sub>E</sub>X, in turn.

LuaT<sub>E</sub>X (Hagen, 2008a) is a T<sub>E</sub>X-like typesetting engine that is able to call functions written using Lua (Ierusalimschy, 2006). On another point, this engine can process texts using XML-like syntax, as shown in (Hagen, 2008b). So Lua can be used to extract and sort the right information from an XML text, and this information is typeset by the engine. Figure 7 gives an implementation of our example in ConT<sub>E</sub>Xt MkIV: it uses Lua to define an interface with sorting functions, the other functionalities being put into action using T<sub>E</sub>X-like commands. As in ConT<sub>E</sub>Xt, the layout is controlled by set-up commands:

```

\start...setup
...
\stop...setup

```

—for example, `title` elements’ contents are just displayed, processing `pulp` elements yields displaying the number or the title, depending on a mode—then our XML file is processed ‘atomically’, by means of the `\xmlprocessfile` command. This approach is promising, but let us recall that LuaT<sub>E</sub>X and ConT<sub>E</sub>Xt Mk2IV have not reached yet stable state: that is planned for the year 2010. Another important drawback: as shown by the examples using the `\xmlfilter` command in Figure 7, this command uses path expressions, very close to XPath expressions, but not identical. For example, `command`—used to connect a selected item to the set-up command that processes it—obviously does not belong to XPath. On the contrary, some XPath expressions are not recognised, even if ‘simple’ paths are processed. Some tricks may be used

```

declare namespace saxon = "http://saxon.sf.net/" ;
declare namespace xsd = "http://www.w3.org/2001/XMLSchema" ;

declare option saxon:output "omit-xml-declaration=yes" ;

declare variable $eol as xsd:string := "&#xA;" ;
declare variable $filename as xsd:string external ;

if (doc-available($filename)) then
  string-join((for $story-0 as element(story) in doc($filename)/story-list/story[pulp]
    let $pulp-nb-0 as xsd:untypedAtomic := data($story-0/pulp/@nb),
        $pulp-nb-0-int as xsd:integer := xsd:integer($pulp-nb-0),
        $pulp-nb-0-string as xsd:string := xsd:string($pulp-nb-0),
        $pulp-title-0 as xsd:string := xsd:string(data($story-0/pulp)),
        $story-title-0 as xsd:string := xsd:string(data($story-0/title))
    order by $pulp-nb-0-int
    return ("`item{",$pulp-nb-0-string,"} ",
        if ($pulp-title-0) then
          $pulp-title-0,"`footnote*{Book's title of pulp no.-",
          $pulp-nb-0-string," : ",$story-title-0,"}" else
          $story-title-0,$eol),
        $eol,"`end", $eol),
    "")) else
  ()

```

FIGURE 6: Getting a source text for *Plain T<sub>E</sub>X* by means of XQuery.

as workarounds, but we think that complete compatibility with XPath should be reached.

## 5 Conclusion

If we sum up the methods shown throughout this article, the approaches that seem suitable are XQuery for simple examples, XSLT for more ambitious ones. The use of LuaT<sub>E</sub>X could be interesting in a near future, too.

However, we think that there are two directions that should be explored. The first would be a modern implementation of XSL-FO using a T<sub>E</sub>X-like typesetting engine. Such an implementation has begun: *Passive T<sub>E</sub>X* (Carlisle et al., 2000), but this project is presently stalled. We think that processing XSL-FO could re-use the experience got by (L<sup>A</sup>)T<sub>E</sub>X developers, even if syntaxes are very different<sup>28</sup>. The second direction would be the definition and implementation of an output mode of XSLT suitable for (L<sup>A</sup>)T<sub>E</sub>X; some additional services could be performed: for example, checking that braces and environments are balanced. Such an output mode already exists in *nbst*<sup>29</sup>, the language of bibliography styles close to XSLT and used by MIBiBT<sub>E</sub>X<sup>30</sup> (Hufflen, 2003), but it concerns only the way to process L<sup>A</sup>T<sub>E</sub>X's special characters.

28. Besides, it is well-known that T<sub>E</sub>X recognises only its own formats, what complicates cooperation between T<sub>E</sub>X and other programs.

29. New Bibliography STyles.

30. MultiLingual BiBT<sub>E</sub>X.

## Acknowledgements

I wish to thank Hans Hagen who greatly helped me debug and improve LuaT<sub>E</sub>X source texts. Thanks to Karl Berry, who made precise some terminology notions. Last but not least, thanks to Gianluca Pignalberi for his patience and his Italian translations of the abstract and keywords.

## References

- apachefop. *Apache FOP*. <http://xmlgraphics.apache.org/fop/>, March 2009.
- Frédéric Boulanger. L<sup>A</sup>T<sub>E</sub>X au pays des tableurs. *Cahiers GUTenberg*, 39–40:7–16, May 2001. In *Actes du Congrès GUTenberg 2001*, Metz.
- Neil Bradley. *The Concise SGML Companion*. Addison-Wesley, 1997.
- David Carlisle, Michel Goossens, and Sebastian Rahtz. De XML à PDF avec *xmltex*, XSLT et *PassiveT<sub>E</sub>X*. *Cahiers GUTenberg*, 35–36:79–114, May 2000. In *Actes du congrès GUTenberg 2000*, Toulouse.
- Jonathan Fine. T<sub>E</sub>X forever! In *Proc. EuroT<sub>E</sub>X 2005*, pages 150–158, Pont-à Mousson, France, March 2005.
- Michel Goossens, Sebastian Rahtz, Eitan M. Gurari, Ross Moore, and Robert S. Sutor. *The L<sup>A</sup>T<sub>E</sub>X Web Companion*. Addison-Wesley Longmann, Inc., Reading, Massachusetts, May 1999.

```

\enablemode[ds:pulp]
\startluacode      % Definitions using Lua.
...
function lxml.sorters.reset(name)
  ...
end      -- (... and other definitions.)
\stopluacode

\def\xmlresetsorter#1{\cxtlua{lxml.sorters.reset("#1")}}
...      % (Other definitions for LuaTEX.)

\startxmlsetups xml:ds:base
  \xmlsetsetups{ds}{*}{-}
  \xmlsetsetups{ds}{story-list|story|title|pulp|pocket-book}{xml:ds:*}
\stopxmlsetups

\xmlregisterdocumentsetup{ds}{xml:ds:base}

\startxmlsetups xml:ds:story-list
  \xmlfilter{story-list}{story/pulp/../../command(xml:ds:stories-plus)}
\stopxmlsetups

\startxmlsetups xml:ds:story-list-plus
  \xmlresetsorter{story-list} \xmlfilter{#1}{story/command(xml:story-list:getkeys)}
  \blank sortkeys: \blank\xmlshowsorter{ds}\blank \xmlsortentries{story-list}
  \xmlflushsorter{story-list}{xml:story-list:flush}
\stopxmlsetups

\startxmlsetups xml:story-list:getkeys
  \xmladdsortentry{story}{#1}{\xmlatt{\xmlfirst{#1}{pulp}}{nb}}
\stopxmlsetups

\startxmlsetups xml:story-list:flush \startitemize \xmlflush{#1} \stopitemize \stopxmlsetups

\startxmlsetups xml:ds:story
  \sym{\xmlfirst{#1}{pulp}} \xmldoifelse{#1}{pulp}{
    {\disablemode[ds:pulp] \xmlfirst{#1}{pulp}\footnote{Book's title: \xmlfirst{#1}{title}}}{
    \xmlfirst{#1}{title}}
\stopxmlsetups

\startxmlsetups xml:ds:title \xmlflush{#1} \stopxmlsetups

\startxmlsetups xml:ds:pulp
  \doifmodeelse[ds:pulp]{\xmlatt{#1}{nb}}{\xmlflush{#1}}
\stopxmlsetups

\starttext \xmlprocessfile{ds}{ds.xml} \stoptext

```

FIGURE 7: Processing a master file by means of ConT<sub>E</sub>Xt Mk IV.

- 
- |   |   |
|---|---|
| <p>Hans Hagen. <i>ConT<sub>E</sub>Xt, the Manual</i>. <a href="http://www.pragma-ade.com/general/manuals/cont-enp.pdf">http://www.pragma-ade.com/general/manuals/cont-enp.pdf</a>, November 2001.</p> <p>Hans Hagen. The luafication of T<sub>E</sub>X and ConT<sub>E</sub>Xt. In <i>Proc. BachoT<sub>E</sub>X 2008 Conference</i>, pages 114–123, April 2008a.</p> <p>Hans Hagen. Dealing with XML in ConT<sub>E</sub>Xt MkIV. MAPS, 37:25–39, 2008b.</p> <p>Jean-Michel Hufflen. MIBIBT<sub>E</sub>X's version 1.3. <i>TUGboat</i>, 24(2):249–262, July 2003.</p> <p>Jean-Michel Hufflen. Introduction to XSLT. <i>Biuletyn GUST</i>, 22:64, April 2005. in <i>BachoT<sub>E</sub>X 2005 conference</i>.</p> | <p>Jean-Michel Hufflen. Advanced techniques in XSLT. <i>Biuletyn GUST</i>, 23:69–75, April 2006. In <i>BachoT<sub>E</sub>X 2006 conference</i>.</p> <p>Jean-Michel Hufflen. Introducing L<sup>A</sup>T<sub>E</sub>X users to XSL-FO. <i>TUGboat</i>, 29(1):118–124, 2007. EuroBachoT<sub>E</sub>X 2007 proceedings.</p> <p>Jean-Michel Hufflen. XSLT 2.0 vs XSLT 1.0. In <i>Proc. BachoT<sub>E</sub>X 2008 Conference</i>, pages 67–77, April 2008a.</p> <p>Jean-Michel Hufflen. Languages for bibliography styles. <i>TUGboat</i>, 2008(3):401–412, July 2008b. TUG 2008 proceedings, Cork, Ireland.</p> <p>Jean-Michel Hufflen. Multidirectional typesetting in XSL-FO. In Tomasz Przechlewski, Karl Berry,</p> |
|---|---|

- and Jerzy B. Ludwiczowski, editors, *Proc. BachoT<sub>E</sub>X 2009 Conference*, pages 37–40, April 2009.
- Roberto Ierusalimsky. *Programming in Lua*. Lua.org, 2 edition, March 2006.
- dsssl. DSSSL. International Standard ISO/IEC 10179:1996(E), 1996.
- Michael H. Kay. *Saxon. The XSLT and XQuery Processor*. <http://saxon.sourceforge.net>, March 2008.
- Richard Kelsey, William D. Clinger, Jonathan A. Rees, Harold Abelson, Norman I. Adams iv, David H. Bartley, Gary Brooks, R. Kent Dybvig, Daniel P. Friedman, Robert Halstead, Chris Hanson, Christopher T. Haynes, Eugene Edmund Kohlbecker, Jr, Donald Oxley, Kent M. Pitman, Guillermo J. Rozas, Guy Lewis Steele, Jr, Gerald Jay Sussman, and Mitchell Wand. Revised<sup>5</sup> report on the algorithmic language Scheme. HOSC, 11(1):7–105, August 1998.
- Jonathan Kew. X<sub>Y</sub>T<sub>E</sub>X in T<sub>E</sub>X live and beyond. *TUGboat*, 29(1):146–150, 2007. EuroBachO<sub>T</sub>E<sub>X</sub> 2007 proceedings.
- Donald Ervin Knuth. *Computers & Typesetting. Vol. A: The T<sub>E</sub>Xbook*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1984.
- Leslie Lamport. *L<sup>A</sup>T<sub>E</sub>X: A Document Preparation System. User’s Guide and Reference Manual*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.
- Alex Martelli. *Python in a Nutshell*. O’Reilly, 2 edition, July 2006.
- Jim Melton and Alan R. Simon. *Understanding the new SQL*. Morgan Kaufmann, 1993.
- Chuck Musciano and Bill Kennedy. *HTML & XHTML: The Definitive Guide*. O’Reilly & Associates, Inc., 5 edition, August 2002.
- Erik T. Ray. *Learning XML*. O’Reilly & Associates, Inc., January 2001.
- Denis B. Roegel. Anatomie d’une macro. *Cahiers GUTenberg*, 31:19–27, December 1998.
- The UNICODE CONSORTIUM. *The Unicode Standard Version 5.0*. Addison-Wesley, November 2006.
- Unicode Bidirectional Algorithm*. The UNICODE CONSORTIUM, <http://unicode.org/reports/tr9/>, March 2008. Unicode Standard Annex #9.
- W3C. *Extensible Stylesheet Language (XSL). Version 1.1*. <http://www.w3.org/TR/2006/REC-xsl11-20061205/>, December 2006. W3C Recommendation. Edited by Anders Berglund.
- W3C. *XML Path Language (XPath) 2.0*. <http://www.w3.org/TR/2007/WD-xpath20-20070123>, January 2007a. W3C Recommendation Draft. Edited by Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael H. Kay, Jonathan Robie and Jérôme Siméon.
- W3C. *XSL Transformations (XSLT). Version 2.0*. <http://www.w3.org/TR/2007/WD-xslt20-20070123>, January 2007b. W3C Recommendation. Edited by Michael H. Kay.
- W3C. *XQuery 1.0 and XPath 2.0 Functions and Operators*. <http://www.w3.org/TR/2007/REC-xpath-functions-20070123>, January 2007c. W3C Recommendation. Edited by Ashok Malhotra, Jim Melton, and Norman Walsh.
- W3C. *XML Schema*. <http://www.w3.org/XML/Schema>, December 2008.

▷ Jean-Michel Hufflen  
LIFC (EA CNRS 4157)  
University of Franche-Comté  
16, route de Gray  
25030 Besançon Cedex  
France  
jmhufflen at lifc dot  
univ-fcomte dot fr