Abstract

L'idée centrale de X\LaTeX est de changer le processus de préparation de document à la \LaTeX, en migrant vers XML (avec toujours la possibilité d'inclure du code \LaTeX dans le code XML en passant par des instructions de traitement). Pour ce faire, nous allons utiliser une DTD ou un schéma XML, très proches de la syntaxe de \LaTeX, de manière à ce que les utilisateurs de X\LaTeX n'aient pas à apprendre de nouveau les mots-clés. Ainsi, les mots-clés utilisés, jusqu'à maintenant, dans les commandes et environnements \LaTeX seront dorénavant des éléments, attributs et espaces de nommage XML (ainsi qu'éventuellement des entités, tant que l'on utilise encore des DTD). L'avantage de cette méthode est que l'on puisse utiliser toute la panoplie d'outils XML pour valider et traiter de documents avant leur composition.

Our main idea is to change the \LaTeX document preparation process by migrating to XML input (with eventual \LaTeX code insertions via XML processing instructions). To do this, we need a DTD or an XML Schema which is very close to \LaTeX syntax, so that users do not need to learn keywords anew. The keywords which used to be \LaTeX commands and environments now become elements, attributes, namespaces (and eventually entities, as long as we still deal with DTDs). The advantage of this method is that one can use XML tools to validate and process documents before typesetting.
XLaTeX, a DTD/Schema Which is Very Close to \LaTeX

Yannis Haralambous  
Département Informatique  
École Nationale Supérieure des Télécommunications de Bretagne  
CS 83 818, 29238 Brest Cédex, France  
http://omega.enstb.org/yannis  
Yannis.Haralambous@enst-bretagne.fr

John Plaice  
School of Computer Science and Engineering  
The University of New South Wales  
UNSW SYDNEY NSW 2052, Australia  
http://www.cse.unsw.edu.au/plaice  
plaice@cse.unsw.edu.au

2003

1 SGML and XML

When author YH first heard about SGML at the DANTE’91 meeting in Vienna, where group of SGML evangelists were presenting it to the \TeX community, Klaus Thull, sitting next to him, reacted by “What do we need another markup system for? We already have \LaTeX...”

\LaTeX is indeed a markup system, and one can even say it is easy to parse: when writing
\begin{center}...
\end{center}

it is quite clear where the “centered” block of text starts and where it ends—also it is quite easy to extract the “tag name:” center, since this name is made out of letters, delimited by braces.

But \LaTeX has a major drawback: it is based upon \TeX, and the latter is one of the hardest programming languages to parse, especially when one is playing with weird macro definitions, or with changing catcodes.

As with most “drawbacks,” there are always people for whom they are rather advantages than drawbacks. In some cases it is a quite interesting feature of \LaTeX to be able to go back to \TeX for obtaining special effects, or for doing things that occur only once “manually.” One can hardly prevent people from using \TeX code in a \LaTeX document. In fact there is no way to tell if a \LaTeX is a “good one” (in SGML jargon, a “valid one”), besides compiling it and seeing if it produces errors or warnings.

SGML has such a mechanism: there are programs called “validators” which can assert either a given document is “valid” or not, for a given set of rules, called a “document type definition” (DTD).

But \LaTeX has other drawbacks: it uses escape characters which can be different in each document (a phenomenon similar to the change of catcodes in \TeX), and, worse, it uses “optional” tags: the DTD can declare that the existence of a given tag can be deduced from its context, and hence it does need to be explicitly written. Both of these features make the presence of a DTD absolutely necessary to be able to parse SGML documents correctly (and not only for validating purposes). These features make SGML document hard to parse and hard to process.

A solution to these problems came with XML [7, 1]. It is easy to underestimate the difference between the two systems. Indeed, when comparing SGML and XML documents, they seem quite
alike. In fact the philosophy is entirely different: in XML no part of the markup is ever hidden or ambiguous. Escape characters are uniquely defined and no tags can ever be optional. DTDs, and their successors XML schemas, are not necessary to parse an XML document, they are only needed to validate it.

2 The Principles of \LaTeX X

When one has used \LaTeX X for more than 15 years, as have the authors, command and environment names, as well as the general syntax, become as automatic as driving a car, putting clothes on, eating and drinking, etc. Braces and backslashes are part of everyday life, and we can hardly imagine it without them.

For author YH, living in France and using a Macintosh, it is quite funny to note that none of these symbols is available on his keyboard (or any of the Macintosh keyboards he has used in the last 20 years). The braces are obtained by 2-key combinations (alt + parenthesis), and the backslash by a 3-key combination (alt + shift + slash).

The backslash is quite a strange symbol: in most languages it does not even have a real name (in French it is usually called “barre oblique inverse,” lately a name has been invented for it: “contre-oblique”). Legend [3, p. 29] says it was introduced into ASCII only as a graphic complement of “/” in order to obtain symbols $\wedge$ (= “/” + “\”) and $\vee$ (= “\” + “/”). As all legends, se non è vero, è ben trovato, since the authors could not find any rational use of the backslash symbol before its use in programming language syntax (besides its use in set theory, which is quite limited).

Is the \TeX X community a community of backslash worshippers? Not necessarily, although a document full of backslashes certainly feels “like home” for many of us.

Nevertheless \TeX Xists are aware of the disadvantages of \TeX X syntax. While it is a general rule that commands start with a backslash, some commands, such as ~, do not have a backslash, while for others, as in \#, a backslash is not used as an escape character. Command arguments are generally delimited by braces, but sometimes by brackets—and sometimes commands have no arguments at all and must be included into groups, as is the case for small. Some commands, such as \verb, can even use any character as argument delimiter (and when we say any, in the case of paired delimiters, like parentheses or braces, one must use the left one on the left and the right one on the right: \texttt{\verb=bla=} is right, \texttt{\verb{bla{ is wrong, \texttt{\verb{bla} is right).

This is the only the tip of the iceberg of \TeX X's syntax problems. On a more philosophical level, a big disadvantage of \LaTeX X, considered as a markup language, is the fact that there is no clear distinction between data and markup. When writing \texttt{\begin{center} is clear that \texttt{\begin{center is part of the markup (the “tag name”), while in \texttt{\emph{hello}}, hello is data. But what about:

\texttt{\textcolor{red}{green}}

Will this produce the word “green” in red color, or the word “red” in green color? The author of the color package decided that the first argument is markup and the second data, but there is no way for parsing software to guess it. Not to mention the fact that there are commands producing data (like \texttt{\today} and others changing the status of data (like the \texttt{\%} character, or the comment environment, or—in a more \TeX -like fashion—the \texttt{\bdef} and \texttt{\edef} commands which convert a string into a command, or the \texttt{\token} command which converts a command into a string. . . ).

Using software such as \texttt{latex2html} one quickly realizes that parsing \TeX code is a perilous daredevil project. In fact only \TeX can parse \TeX code well. This is quite fair in a world where \TeX files are written for the sole purpose of being compiled, but in the current era of electronic documents this hardly anymore the case. Nowadays documents are used in many different ways: they can be parsed, transformed, indexed, translated, re-assembled, etc.

For this to happen, a stable and simple markup system like XML is much more suitable than \LaTeX X.

But shall \LaTeX Xists learn an entirely new syntax? Of course not. Only the basic syntactic rules should change: “less than” and “greater than” instead of backslash and braces, “elements” and
“attributes” instead of “commands” and “environments.”

The XL\TeX proposal is the following: a set of XML elements and attributes (= a DTD or an XML schema), with tag names as close as possible to \LaTeX command and environment names, easily convertible to \LaTeX syntax.

Tag names such as document, maketitle, center, quotation, itemize, enumerate, emph, footnote, chapter, section, are used on a daily basis by all \LaTeX users. They remain unchanged for XL\TeX. For example, the \LaTeX code:

\begin{quotation}
Life shall go on\footnote{Said \emph{he}.}.
\end{quotation}

becomes in XL\TeX:

<quotation>
Life shall go on<footnote>Said <emph>he</emph>.</footnote>.
</quotation>

2.1 Elements or attributes?

An attribute has a tag name and contents. It belongs to an element (and is actually written inside the opening tag of the element). The order of attributes is not relevant, but there cannot be two attributes with the same name in the same element. The contents of an attribute cannot include character <, and hence cannot include element tags.

Attributes are used for metadata, whic is data considered to be either markup or contents. For example, if <section> is the opening tag of a section title to be automatically numbered, then one can imagine <section number="3"> as the opening tag of a section numbered “3.” Is this number “3” part of the contents of the document? The answer is not clear.

Attributes are very useful when we have variable markup. The typical example is the format of a tabular environment. This format is different for each table, but it is nevertheless pure markup (nobody would like \|c|c|p{2cm}| to appear in her document). Here is how an XL\TeX tabular environment/element looks like:

<tabular format="|c|c|p{2cm}|">
A<tab/>B<tab/>C<br/>
D<tab/>E<tab/>F<br/>
</tabular>

We notice two things: first of all, attributes have names, so every \LaTeX command argument becoming an attribute needs a name. “Format” seems to be the natural name for the format of a tabular environment. Secondly, the special character \ and the command \ have been replaced by elements: <tab/> and <br/>. The choice of the latter comes from plain \TeX where the command \textbackslash br plays a role similar to that of \ in \LaTeX.

Sometimes the choice between element and attribute is not clear. Let us take for example the optional argument of the \section command (the version of a title used in the table of contents). It seems natural to write, for example:

<section toc="Short version">Long version</section>

But what happens when we need further markup inside such a title? If the long title contained, for example, an <emph> element, then this element could not be used in the short one, since an attribute is not allowed to contain tags. There are two solutions to this problem, neither of which is entirely satisfactory:
1. use an element instead of an attribute, for example:

```xml
<section>
  <toc>Short <emph>version</emph></toc>
  Long <emph>version</emph>
</section>
```

2. use \LaTeX{} commands instead of XML markup in the attribute:

```xml
<section toc="Short \emph{version}">
  Long <emph>version</emph>
</section>
```

### 2.2 How about \LaTeX{} code?

One may argue that by “writing \LaTeX{} in XML,” one can only use pre-defined elements, and hence one loses all the flexibility of \TeX{} code. XML provides a very simple and natural mechanism to switch between syntaxes: processing instructions. In XML one can switch to \TeX{} code at any moment, via the \texttt{tex} processing instruction:

```xml
<footnote>This symbol was very <emph>scary</emph> and looked like an \texttt{\textbackslash xx\textbackslash char'124}?.</footnote>
```

Elegant XML code would, of course, rather try to avoid such processing instructions. As always in \LaTeX{}, \TeX{} code should be used only when unavoidable. But in XML such code is clearly marked and will be avoided by XML parsers. There is only one hitch: the string “?>” should never appear inside the \TeX{} code, since it is the processing instruction escape sequence.

Other processing instructions used are \texttt{math} (for math formulas), \texttt{displaymath} (for display mathematics), \texttt{verb} (for short verbatim, similar to the \texttt{\verb} command), \texttt{verbatim} (for long verbatim code), \texttt{special} (similar to the \texttt{\special} command).

Using processing instructions has the advantage that one does not need to care about protecting characters <, >, & (only the ?> must be avoided). But it also has a serious disadvantage: the data included in the processing instruction is not considered as contents of the document. In some cases this seems the right approach: in \LaTeX{} one would hardly put textual contents into a \texttt{\special} command, although this is theoretically possible—this means that using processing instructions for specials will most probably not “hide” any contents of the document.

This is less clear with, for example, verbatim code or math formulas (although in the latter case one could as well also use MathML as the proper way of writing mathematics with XML). For that reason XML also provides XML elements for math formulas, verbatim code and specials. When using these elements, one must always protect characters <, >, & by using the appropriate entities (&lt;, &gt;, &amp;). Here is an example: to obtain the code:

```xml
<textbf>this is cute</textbf>
```

one can write either:

```xml
<?verbatim
<textbf>this is cute</textbf>
?>
```

or:

```xml
<verbatim>
\&lt;textbf\&gt;this is cute\&lt;textbf\&gt;
</verbatim>
```

The latter solution is “cleaner,” but the former is more readable.
2.3 Other similar projects and history of XLTEX

In November 1998, Doug Lovell from IBM AlphaWorks released a package called texml to translate XML into \TeX{} \cite{5}. This package is described as a three-part solution that provides a path from XML into the \TeX{} formatting language. This project is described in the TUGboat article \cite{6}. It is retired from IBM and appears on Sourceforge in 2004 \cite{8} (developer: Oleg Paraschenko). In 1999, Stefan Krauß, from Stuttgart University, started a similar project called SESAMDoc \cite{4}.

Both TEXML and SESAMDoc use an approach quite different from ours. Instead of defining elements with names similar to those of \LaTeX{} commands and environments, they define elements from commands and environments, where the names appear in an attribute: instead of writing

\[
<\textbf{emph}>bingo!</\textbf{emph}>
\]

as we are, SESAMDoc would write:

\[
<\text{cmd name}="\textbf{emph}"
  <\text{param}>bingo!</\text{param}>
</\text{cmd}>
\]

(the \TeX{}ML code would be similar except for the spelling \texttt{parm} instead of \texttt{param}).

\TeX{}ML and SESAMDoc are less suitable for manual keying in and editing than XL\TeX{}. By systematically using \texttt{param/parm} elements, one loses the distinction between data and markup/metadata. In XL\TeX{} it is possible to write arbitrary commands and environments that way, but one can also use attributes for \LaTeX{} arguments, and mix the two approaches so that data gets into element contents and metadata/markup into attributes.

Compare the XL\TeX{} approach:

\[
<\text{com name}="\textcolor" arg1="red"
  <\text{arg2}>This text is typeset in red.</\text{arg2}>
</\text{com}>
\]

where argument 1 (whose value is metadata) is an attribute while argument 2 (whose value is textual content) is a sub-element, to the \TeX{}ML approach:

\[
<\text{cmd name}="\textcolor">
  <\text{arg1}>red</\text{arg1}>
  <\text{arg2}>This text is typeset in red.</\text{arg2}>
</\text{cmd}>
\]

where there is no qualitative distinction between “red” and “This text is typeset in red.”

The XL\TeX{} project was started in late 2002 as a Diploma Project for Paweł Grams, at that time student of ENST Bretagne. He presented his work at the 2003 GUST meeting in Bachotek \cite{2}.

In the following section we will describe the XL\TeX{} (version 1) syntax.

3 XL\TeX{} v. 0.9 syntax

3.1 Namespace

The namespace of XL\TeX{} v. 0.9 is:

http://omega.enstb.org/2003/XLaTeX
3.2 Conventions

Ages before the arrival of Unicode, Knuth introduced some easy ways to obtain characters not in ASCII: ‘‘ and ‘’ for the double quotes, ‘ and ‘ for simple quotes, -- and --- for n-dash and m-dash, ‘? and ‘! for Spanish inverted punctuation. The most frequently used of these is ‘’ which produces an “apostrophe” (a raised comma) although the character used in the document is an “ASCII apostrophe” (a small straight line).

Packages like babel and Omega Translation Processes have introduced new conventions: for example, in French one leaves a blank space in front of double punctuation, this space is converted into a non-breakable space (in the case of colon) or into a thin space (in all other cases).

Such conventions were invented almost a century ago, when the typewriter began to be used. Going from a full blank space to a thin space is the same as going from the typewriter’s world (“dactylography”) to the printer’s one (“typography”).

One may argue whether these conventions should be left in X\LaTeX or not. They are part of \TeX and our fingers are used to them, especially if we consider ourselves as being dactylographers and \TeX as the typographer-in-the-box. On the other hand, XML and hence X\LaTeX is based on Unicode, and this encoding contains all of these characters. The problem is no longer to get the characters into the document, but to configure our keyboard to produce them easily. And the final argument is that even in \TeX these conventions were deactivated in some contexts, for example in verbatim environment or when using a typewriter font.

To give the future, i.e., Unicode, a chance we have chosen not to activate these conventions by default. They can still be activated using attribute tex-conventions which can take values on and off. The value of this attribute is inherited by children of a node, like XSL-FO properties.

3.3 Encodings

The default encoding of XML (and hence of X\LaTeX) is Unicode UTF-8. This encoding can be changed through the encoding pseudo-attribute of the XML declaration, but we wouldn’t advise the user to do so.

3.4 Global document structure

XML documents have a tree structure: they need a single top node. \LaTeX documents have two parts: the preamble (which has no top node) and the document body (which has the top node document). To obtain a structure similar to \LaTeX, we have to introduce an additional node above document. It is only natural to call this node xlatex.

On the other hand, every \LaTeX document has one, and only one, \documentclass command. There are two ways to translate this into XML: either as an element under xlatex, or as an attribute of document (in the latter case, the value of this attribute is the name of the document class, and a second attribute options includes the eventual class options).

In a \LaTeX preamble one finds a lot of code but mostly \usepackage commands. These can be included in an XML document as \usepackage elements (with self-explanatory name and options attributes). These elements can be used either under xlatex and before document or directly under document.

Hence a typical \LaTeX document, like:

\documentclass[11pt]{article}
\usepackage{francais}{babel}
\usepackage[dvips]{graphics}
\begin{document}
...
\end{document}

can become (for \LaTeX purists):

\documentclass[11pt]{article}
\usepackage{francais}{babel}
\usepackage[dvips]{graphics}
\begin{document}
...
\end{document}
or (more in XML style):

```xml
<?xml version="1.0"?>
<xlatex version="0.9">
  <document class="article" options="11pt">
    <usepackage name="babel" options="francais"/>
    <usepackage name="graphics" options="dvips"/>
    ... 
  </document>
</xlatex>
```

In the latter case, the `\usepackage` instructions will be placed at the beginning of the preamble before any code included under `xlatex` and before `document`.

It becomes obvious from the example above that X\LaTeX to \LaTeX translation is not trivial and requires more than one parsing run. In the next subsection this will be even more the case.

### 3.5 Languages

Using the `babel` package, languages are first declared (as options of the `\usepackage` command) and then activated through the `\selectlanguage` command. This approach is made possible in X\LaTeX:

```xml
<?xml version="1.0"?>
<xlatex version="0.9">
  <usepackage name="babel" options="francais,english"/>
  <document class="article" options="11pt">
    Are we writing in Shakespeare's language?
    <selectlanguage name="french">
      Ou est-ce dans la langue de Molière ?
    </selectlanguage>
  </document>
</xlatex>
```

But there is also a different approach, more XML-oriented! In XML, there is a standard way to specify the language used in an element: the `xml:lang` attribute. A value of this attribute is a combination of a 2-letter language code (ISO 639) and a 2-letter country code (ISO 3166), separated by a dash.

One can consider—although this is not stated in the XML specifications—that the value of this attribute is inherited by nodes underneath the element carrying it.

Every X\LaTeX element can carry the `xml:lang` attribute and there is no need to declare the `babel` package with the appropriate language. The X\LaTeX to \LaTeX parser will find all occurrences of the attribute and load the corresponding languages in the document preamble. Hence the example above could also be written as:
<xlatex version="0.9">
<document class="article"
  options="11pt" xml:lang="en">
Are we writing in Shakespeare's language?
<emph xml:lang="fr">Ou est-ce dans la langue de Molière ?</emph>
</document>
</xlatex>

Correspondence between values of the xml:lang attribute and babel language names is included in the XL\TeX's configuration file xlatex.conf.

### 3.6 Sections, text styles, footnotes, lists, tables

\LaTeX commands \section and the like become XL\TeX elements, containing section titles. Attribute short can contain a shorter version of the title for tables of contents and/or headers (depending on the style file).

Commands changing text style (\emph, \textbf, and the like) become XL\TeX elements. There is also a neutral element, used to carry attributes such as xml:lang: it is called span (like in HTML).

Footnotes are obtained through the footnote element. As in \LaTeX there are also \footnotenumber and \footnotetext commands, but the XL\TeX to \LaTeX translator should be able to replace a \footnote element by a pair of \footnotenumber and \footnotetext elements, whenever the footnote occurs in a not appropriate \LaTeX environment (such as a table). In other words, the XL\TeX to \LaTeX translator should be able to rectify some of \LaTeX's deficiencies (or at least to act as if these deficiencies were not there).

Lists are obtained through itemize, enumerate and description elements. Each list item is contained in an item element. This element can carry a mark attribute, containing the list item mark. If this mark contains a closing bracket, then it will be automatically converted into a \char"5D command (this is a well-known \LaTeX problem coming from the fact that the mark is an optional argument of the \item command and hence is delimited by brackets instead of braces).

Tables are obtained through the tabular element, which takes two attributes: format and pos. When the format attribute contains values such as m or b then the array package is automatically loaded. When it contains value X then the tabularX environment is automatically loaded. Cells are separated by the tab element, and ends of line are given by the \br element. Horizontal lines are included by the hline element.

Multicolumn cells are obtained by the multicolumn element, which takes two attributes: num (the number of columns) and format (the format of the cell). The contents of the multicolumn element are the contents of the cell. Partial horizontal lines are included by the cline element which carries a num attribute.

Here is an example of a table with all the features described:

<table>
<thead>
<tr>
<th>One</th>
<th>Two</th>
<th>Three</th>
</tr>
</thead>
<tbody>
<tr>
<td>Four</td>
<td>\multicolumn{2}{</td>
<td>c</td>
</tr>
<tr>
<td>\cline{2-3} Five</td>
<td>Seven</td>
<td>Eight</td>
</tr>
</tbody>
</table>

\begin{tabular}{|c|c|c|}
\hline 
One&Two&Three\\
\hline 
Four&\multicolumn{2}{|c|}{Five}\\
\hline 
Six&Seven&Eight\\
\hline 
\end{tabular}
3.7 Cross references

There are two approaches to cross references: the \LaTeX way and the XML way. The former is to use \texttt{label}, \texttt{ref} and \texttt{pageref} elements, carrying \texttt{id} attributes. The latter is to use \texttt{id} attributes instead of \texttt{label} elements. These attributes can be carried by any XL\LaTeX element.

Here is an example of these two approaches: a reference to a section title.

\begin{verbatim}
\section{Lyubov Bruk & Mark Taimanov}
\label{bruk-taimanov}
\end{verbatim}

is the “\LaTeX way,” and:

\begin{verbatim}
\section{id=bruk-taimanov}{Lyubov Bruk & Mark Taimanov}
\end{verbatim}

the “XML way.”

3.8 Mathematics, verbatim

As already mentioned, mathematics and verbatim code can be included in two ways: either by processing instructions or by XL\LaTeX elements \texttt{math}, \texttt{displaymath}, \texttt{verb} and \texttt{verbatim}:

To calculate \begin{verbatim}$\sqrt{2}$\end{verbatim} use function \begin{verbatim}\verb|sqrt(2)|\end{verbatim}.

or:

To calculate \begin{verbatim}$\sqrt{2}$\end{verbatim} use function \begin{verbatim}\verb|sqrt(2)|\end{verbatim}.

\begin{verbatim}
\verb|To calculate $\sqrt{2}$ use function \verb|sqrt(2)|.|
\end{verbatim}

In the case of XML elements, characters \texttt{<}, \texttt{>} and \texttt{&} should be entered as \texttt{&lt;}, \texttt{&gt;} and \texttt{&amp;}.

In the case of processing instructions the only constraint is that the string \texttt{?>} should be included in the contents.

Let us underline the fact that the XL\LaTeX to \LaTeX translator does not produce \texttt{\verb} commands and \texttt{verbatim} environments out of \texttt{verb} and \texttt{verbatim} elements or PIs. Instead it simply changes the font into a typewriter one and translates characters. In other words, we obtain the verbatim effect in a “manual way.” This has the enormous advantage that verbatim code can be used everywhere, including in footnotes, tables, section titles and other places where it is prohibited in normal \LaTeX.

3.9 Arbitrary commands and environments

It may happen that a user wants to use a given \LaTeX command that is not included in the XL\LaTeX DTD (or schema). In that case one can either use the \texttt{tex} processing instruction, which switches immediately into \TeX mode, or elements \texttt{com}, \texttt{env}, \texttt{arg1}, \ldots, \texttt{arg9} and \texttt{optarg}.

Here is an example: suppose that a user has defined a \LaTeX command called \texttt{toto} with one optional argument and two mandatory ones. She wants to use it as follows:
\texttt{\textcolor{red}{toto}}\{2\}{Some text.}

There are three ways to obtain this code. By a processing instruction:

\begin{verbatim}
<?tex \texttt{\textcolor{red}{toto}}\{2\}{Some text.}?>
\end{verbatim}

in which case an XML parser would not be able to parse the data properly, or by the \texttt{com} element and \texttt{arg*} attributes:

\begin{verbatim}
<com name="toto"
  arg1="2"
  arg2="Some text."
  optarg="red"/>
\end{verbatim}

or by \texttt{com} and \texttt{arg*} elements:

\begin{verbatim}
<com name="toto">
  <arg1>2</arg1>
  <arg2>Some text.</arg2>
  <optarg>red</optarg>
</com>
\end{verbatim}

The two approaches can be mixed so that the author of the document has full control on what is to be considered as markup/metadata, and what is data (textual content). In the case of our example, 2 and \texttt{red} are probably metadata while \texttt{Some text.} is obviously text. Hence, it would be more elegant to write:

\begin{verbatim}
<com name="toto" optarg="red"
  arg1="2"><arg2>Some text.</arg2>
</com>
\end{verbatim}

To obtain an environment instead of a command, one uses the \texttt{env} element. The contents of the element are the contents of the environment.

The advantage of using elements instead of merely switching to \TeX{} mode via a PI is that XML parsers or processors (like SAX/DOM or XSLT) can transform these elements into other XML elements, at wish—while this is hardly possible (or at least much more difficult) inside a processing instruction.

3.10 Graphics, figures, multiple columns, files

The following elements produce \texttt{graphics} package commands:

\begin{verbatim}
<includegraphics src="toto.eps"
  bbox="50 70 327 655"/>
<scalebox amount="0.1"/>
<rotatebox amount="30"/>
<resizebox x="0.5" y="1"/>
\end{verbatim}

Instead of using elements for scaling, rotating and resizing, one can also use attributes carried by the \texttt{includegraphics} element:

\begin{verbatim}
<includegraphics src="toto.eps"
  bbox="50 70 327 655"
  scale="0.1"
  rotate="30"
  resizex="0.5" resizey="1"/>
\end{verbatim}
In that case, operations are done in the following order: first rotating, then resizing, and finally scaling.

Using either one of these elements automatically loads the \texttt{graphics} package.

Floating figures and tables are obtained by elements \texttt{figure} and \texttt{table} having a single argument \texttt{pos}. By using an \texttt{H} in the value of \texttt{pos}, the \texttt{float} package is automatically loaded.

Captions are obtained by the \texttt{caption} element.

Multiple columns are obtained by the \texttt{multicols} element which takes one attribute: \texttt{pos}. The \texttt{multicol} package is automatically loaded. One can also use elements \texttt{twocolumns} and \texttt{onecolumn} as in standard \TeX.

To include files one can use \texttt{input}, \texttt{include} and \texttt{includeonly} elements (with \texttt{src} attribute, containing the file name).

3.11 Miscellanea

The \TeX, \LaTeX, \XeLaTeX, \METAFONT, etc. logos are obtained through elements \texttt{<TeX/>}, \texttt{<LaTeX/>}, \texttt{<XLaTeX/>}, \texttt{<MF/>}, and so on. The \texttt{\today} command is obtained by the \texttt{today} element.

3.12 Bibliography, index

Index entries are obtained by the \texttt{index} element, which can be used in three different ways:

1. empty, and with an \texttt{id} attribute:

\begin{verbatim}
<index id="horse"/>horses is equivalent to:
\index{horse}horses
\end{verbatim}

2. non-empty without attribute:

\begin{verbatim}
<index>horse</index> is equivalent to:
\index{horse}horse
\end{verbatim}

3. non-empty with an \texttt{id} attribute:

\begin{verbatim}
<index id="horse">horses</index> is equivalent to:
\index{horse}horses
\end{verbatim}

The \texttt{printindex} command produces the index. The \texttt{makeidx} package is automatically loaded and the \texttt{\makeindex} command automatically inserted. The \texttt{index} and \texttt{printindex} elements can carry another attribute: \texttt{name}. In that case several indexes are built, identified by their “names.” The \texttt{multind} package is automatically loaded.

Bibliographical references are obtained through the \texttt{cite} element, which takes two attributes: \texttt{key} and \texttt{opt}. There is also a \texttt{nocite} element with \texttt{key} attribute. To obtain the list of bibliographical references one can use elements \texttt{bibliographystyle} (with attribute \texttt{src}) and \texttt{bibliography} (with attribute \texttt{src}). Instead of \texttt{bibliographystyle} element can also use attribute \texttt{style} carried by element \texttt{bibliography}:

\begin{verbatim}
<bibliography
template="plain"
width="666"
src="mybibliography"/>
\end{verbatim}

The \texttt{bibliography} element can contain \texttt{bibitem} sub-elements. In that case it is converted into a \texttt{thebibliography} environment. \texttt{bibitem} elements contain \texttt{key} and \texttt{label} attributes.
4 Availability, further developments

\[\Xi\LaTeX\] is not yet stable since we would like the \TeX{} community to provide us with feedback and thorough testing, before version 1.0 is released. Open questions remain, such as the additional packages that should be provided and automatically loaded as well as the exact features of the \[\Xi\LaTeX\] to \LaTeX{} translator.

The first real-word document written in \XiLaTeX{} was the author YH’s book \textit{Fontes et codages} published by O’Reilly France in April 2004.

The up-to-date Web page of \XiLaTeX{} is \url{http://omega.enstb.org/xlatex}. A prototype \XiLaTeX{} to \LaTeX{} translator (written in Perl) can be found on this page. All \XiLaTeX{} development belongs to the public domain (GNU copyleft license).

Once \XiLaTeX{} is stable we are planning to write an Omega input model (so that \XiLaTeX{} files can be read directly by Omega) as well as an \XiLaTeX{} mode for Emacs, XSLT code for converting \XiLaTeX{} into XHTML or XSL-FO, etc.

References


