XLaTeX, a DTD/Schema which is very close to LaTeX

Yannis Haralambous, John Plaice

XLaTeX, a DTD/Schema Which is Very Close to LaTeX

Yannis Haralambous
Département Informatique
École Nationale Supérieure des Télécommunications de Bretagne
CS 83818, 29238 Brest Cédex, France
Yannis.Haralambous@enst-bretagne.fr
http://omega.enstb.org/yannis

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

Abstract

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.

Résumé

L'idée centrale de XLaTeX 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 XLaTeX 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.

SGML and XML

The author (YH) first heard about SGML at the 1991 DANTE meeting in Vienna when a group of SGML evangelists presented it to the TeX community. Klaus Thull, sitting next to him, reacted by saying "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 advantages rather 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 document is a "good one" (in SGML jargon, a "valid one"), other than compiling it and seeing if it produces errors or warnings.

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

But SGML has other drawbacks: it uses escape characters which can be different in each document (a phenomenon similar to changing 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 documents hard to parse and hard to process.

A solution to these problems came with XML (see [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.

The principles of XLaTeX

When using LaTeX for more than 15 years, like 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 the first author, 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 used in the last 20 years). The braces are obtained by 2-keys combinations (alt + parentheses), and the backslash by a 3-key combination (alt + shift + slash).

The backslash is quite a strange symbol: in most languages it doesn't 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 ∧ (= "/" + "\") and ∨ (= "\" + "/"). As all legends, *se non è vero, è ben trovato*, since the author could not find any rational use of the backslash symbol before the arrival of programming language syntax (besides its use in set theory, which is quite limited).

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

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

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

```
\textcolor{red}{green}
```

Will this produce the word "green" in the color red, or the word "red" in the color green? The author of the `color` package has 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 `\today`) and others changing the status of data (like the % character, or the `comment` environment, or — in a more TeX-like fashion — the `\bdef` and `\edef` commands which convert a string into a command, or the `\token` command which converts a command into a string...).

Using software like *latex2html* one 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 can hardly be the case anymore. Nowadays documents are used in many different ways: they can be parsed, transformed, translated, re-assembled, etc.

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

But shall LaTeXists 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 XLaTeX 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 like `document`, `maketitle`, `center`, `quotation`, `itemize`, `enumerate`, `emph`, `footnote`, `chapter`, `section`, are used on a daily basis by all LaTeX users. They remain unchanged for XLaTeX. For example, the LaTeX code:

```
\begin{quotation}
Life shall go on\footnote{Said
```

```
\emph{he}.}.
\end{quotation}
```

becomes in XLaTeX:

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

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 the character <, and hence cannot include element tags.

Attributes are used for metadata, that is data which can be considered as being either markup or contents. For example, if `<section>` is the opening tag of a section title which will be automatically numbered, then one could 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 what an XLaTeX `tabular` environment/element looks:

```
<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/>`.

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:

```
<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 may not contain tags. There are two solutions, neither entirely satisfactory:

1. use an element instead of an attribute, for example:

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

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

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

How about TeX 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 XLaTeX one can switch to TeX code at any moment, via the `tex` processing instruction:

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

Elegant XLaTeX 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 XLaTeX 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 `math` (for math formulas), `displaymath` (for display mathematics), `verb` (for short verbatim, similar to the `\verb` command), `verbatim` (for long verbatim code), `special` (similar to `\special` command).

Using processing instructions has the advantage that one doesn't need to care about protecting the characters <, >, & (only the sequence ?> 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 `\special` command, although this is theoretically possible — thus, 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 XLaTeX 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;`). As an example, to obtain the output:

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

one can write either:

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

or:

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

The latter solution is "cleaner", but the former is more readable.

Other similar projects and history of XLaTeX In November 1998, Doug Lovell from IBM AlphaWorks released a package called *texml* to translate XML into TeX [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 a *TUGboat* article [6]. It has been retired from IBM and appeared on Sourceforge in 2004 [8] (developer: Oleg Paraschenko). In 1999, Stefan Krauß, from Stuttgart University, starts a similar project called *SESAMDoc* [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. For instance, instead of writing

```
<emph>bingo!</emph>
```

as we are, *SESAMDoc* would write:

```
<cmd name="emph">
   <param>bingo!</param>
</cmd>
```

(the TeXML code would be similar except for the spelling `parm` instead of `param`).

TeXML and *SESAMDoc* are less suitable for manual keyboarding and editing than XLaTeX. By systematically using `param`/`parm` elements, one loses the distinction between data and markup/metadata. In XLaTeX 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 XLaTeX approach:

```
<com name="textcolor" arg1="red">
<arg2>This text is typeset in red.</arg2>
</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 TeXML approach:

```
<cmd name="textcolor">
<arg1>red</arg1>
<arg2>This text is typeset in red.</arg2>
</cmd>
```

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

The XLaTeX project began 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 [2].

In the following section we describe the XLaTeX (version 1) syntax.

XLaTeX v. 0.9 syntax

Namespace The namespace of XLaTeX v. 0.9 is:

```
http://omega.enstb.org/2003/XLaTeX
```

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 single quotes, -- and --- for en-dash and em-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 ("typography").

One may argue whether these conventions should be left in XLaTeX 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 XLaTeX is based on Unicode, and this encoding contains all of these characters. The problem is not anymore to get the characters in 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 (that is: Unicode) a chance we have chosen not to activate these conventions by default. They can be activated, though, 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.

Encodings The default encoding of XML (and hence of XLaTeX) is Unicode UTF-8. This encoding can be

changed through the `encoding` pseudo-attribute of the XML declaration, but we do not advise the user to do so.

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 for us 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 XLaTeX 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):

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

or (more in XML style):

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

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 XLaTeX to LaTeX translation is not trivial and requires more than one parsing run. In the next subsection this will be even more clear.

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 XLaTeX:

```
<?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">
<emph>Ou est-ce dans la langue
de Molière ?</emph>
</selectlanguage>
</document>
```

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. Values of this attribute are combinations of standard 2-letter language names (ISO-639) and 2-letter country names (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 XLaTeX element can carry the `xml:lang` attribute and there is no need to declare the *babel* package with the appropriate language. The XLaTeX to LaTeX parser will find all occurences of the attribute and load the corresponding languages in the document preamble. Hence the example above could also be written as:

```
<?xml version="1.0"?>
<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>
```

Correspondence between values of the `xml:lang` attribute and *babel* language names is included in the XLaTeX configuration file `xlatex.conf`.

Sections, text styles, footnotes, lists, tables The LaTeX commands `\section` and the like become XLaTeX elements, containing section titles. Attribute `short` can contain a shorter version of the title for table of contents and/or headers (depending on the style file).

The LaTeX commands changing text style (`\emph`, `\textbf`, and the like) become XLaTeX elements. There is also a neutral element, used to carry attributes such as `xml:lang`; it is called `span` (as in HTML).

Footnotes are obtained with the `footnote` element. As in LaTeX there are also `footnotenumber` and `footnotetext` elements, but they should not be needed, as the XLaTeX to LaTeX translator should be able to replace a `<footnote>` element in an inappropriate LaTeX environment (such as a table) by paired `<footnotenumber>` and `<footnotetext>` elements. In other words, the XLaTeX 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 the 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 element `multicolumn`, which takes two attributes: `num` (the number of columns) and `format` (the format of the cell). The contents of the `multicolumn` element is 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:

One	Two	Three
Four	Five	
Six	Seven	Eight

```
\begin{tabular}{|c|c|c|}\hline
One&Two&Three\\\hline
Four&\multicolumn{2}{c|}{Five}\\\cline{2-3}
Six&Seven&Eight\\\hline
\end{tabular}
```

```
<tabular format="|c|c|c|"><hline/>
One<tab/>Two<tab/>Three<br/><hline/>
Four<tab/><multicolumn num="2"
  format="c|">Five</multicolumn><br/>
  <cline num="2-3"/>
Six<tab/>Seven<tab/>Eight<br/><hline/>
</tabular>
```

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

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

```
<section>Lyubov Bruk &amp; Mark Taimanov
<label id="bruk-taimanov"/></section>
```

is the "LaTeX way", and:

```
<section id="bruk-taimanov">Lyubov Bruk
&amp; Mark Taimanov</section>
```

the "XML way".

Mathematics, verbatim As already mentioned, mathematics and verbatim code can be included in two ways: either by XML processing instructions or by XLaTeX elements `math`, `displaymath`, `verb` and `verbatim`:

```
To calculate <math>\sqrt{2}</math>
use function <verb>sqrt(2)</verb>.
```

or:

```
To calculate <?math \sqrt{2}?>
use function <?verb sqrt(2)?>.
```

to obtain:

To calculate $\sqrt{2}$ use function `sqrt(2)`.

In the case of XML elements, the characters `<`, `>` and `&` must be entered as `&lt;`, `&gt;`, `&amp;`. In the case of processing instructions the only constraint is that the string `?>` must not be included in the contents.

Let us emphasize that the XLaTeX to LaTeX translator does not produce `\verb` commands and `verbatim` environments from `verb` and `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.

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

Here is an example: suppose that a user has defined a LaTeX command called `toto` with one optional argument and two mandatory ones. She wants to use it as follows:

```
\toto[red]{2}{Some text.}
```

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

```
<?tex \toto[red]{2}{Some text.}?>
```

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

```
<com name="toto"
   arg1="2"
   arg2="Some text."
   optarg="red"/>
```

or by `com` and `arg*` elements:

```
<com name="toto"><arg1>2</arg1>
   <arg2>Some text.</arg2>
   <optarg>red</optarg>
</com>
```

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

```
<com name="toto" optarg="red"
   arg1="2"><arg2>Some text.</arg2>
</com>
```

To obtain an environment instead of a command, one uses the `env` element. The contents of the element is 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, as desired — while this is hardly possible (or at least much more difficult) inside a processing instruction.

Graphics, figures, multiple columns, files The following elements produce *graphics* package commands:

```
<includegraphics src="toto.eps"
   bbox="50 70 327 655"/>
<scalebox amount="0.1"/>
<rotatebox amount="30"/>
<resizebox x="0.5" y="!"/>
```

Instead of using elements for scaling, rotating and resizing, one can also use attributes of the `includegraphics` element:

```
<includegraphics src="toto.eps"
   bbox="50 70 327 655"
   scale="0.1"
   rotate="30"
   resizex="0.5" resizey="!"/>
```

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 *graphics* package.

Floating figures and tables are obtained by elements `figure` and `table` having a single argument `pos`. If an `H` is included in the value of `pos`, the *float* package is automatically loaded.

Captions are obtained by the `caption` element.

Multiple columns are obtained by the `multicols` element, which takes one attribute: `pos`. The *multicol* package is automatically loaded. One can also use elements `twocolumns` and `onecolumn` as in standard LaTeX.

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

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

Bibliography, index Index entries are obtained by the `index` element, which can be used in three different ways:

1. empty, and with an `id` attribute:
   `<index id="horse"/>horses` is equivalent to:
   `\index{horse}horses`
2. non-empty without attribute:
   `<index>horse</index>` is equivalent to:
   `\index{horse}horse`
3. non-empty with an `id` attribute:
   `<index id="horse">horses</index>`
   is equivalent to:
   `\index{horse}horses`

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

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

```
<bibliography
   style="plain"
   width="666"
   src="mybibliography"/>
```

The `bibliography` element can contain `bibitem` subelements. In that case it is converted into a `thebibliography` environment. `bibitem` elements contain `key` and `label` attributes.

Availability, further developments

XLaTeX 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 XLaTeX to LaTeX translator.

The first real-world document written in XLaTeX was the first author's book *Fontes et codages*, published by O'Reilly France in April 2004.

The up-to-date Web page of XLaTeX is `http://omega.enstb.org/xlatex`. A prototype XLaTeX to LaTeX translator (written in Perl) can be found on this page. All XLaTeX development is publically available (GNU copyleft license).

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

References

[1] Anelyse Boukhors, Alexandre Kaszycki, Jérôme Laplace, Sandrine Munerot, and Laurent Poublan. *XML, la synthèse*. Eyrolles, 2003.

[2] Paweł Grams. XML jako wejście Omegi. In *XI Ogólnopolska Konferencja Polskiej Grupy Użytkowników Systemu TeX*, 2003.

[3] Yannis Haralambous. *Fontes & codages*. O'Reilly France, 2004.

[4] Stefan Krauß. SESAMDoc, LaTeX-DTD für die Druckausgabe. `http://www.iste.uni-stuttgart.de/se/people/krauss/sesamdoc`, 1999.

[5] Douglas Lovell. IBM AlphaWorks TeXML. `http://www.alphaworks.ibm.com/aw.nsf/techreqs/texml`, 1998.

[6] Douglas Lovell. TeXML: Typesetting XML with TeX. *TUGboat*, 20(3):176–183, September 1999.

[7] W. Scott Means and Elliotte Rusty Harold. *XML in a Nutshell, manuel de référence*. O'Reilly France, 2e edition, December 2002. `http://www.oreilly.fr/catalogue/xml_nutshell_2.html`.

[8] Oleg Paraschenko. TeXML: an XML vocabulary for TeX. `http://sourceforge.net/projects/getfo/`, 2004.