From LaTeX to HTML and Back

Michel Goossens and Janne Saarela, CERN,
CN Division, CH-1211 Geneva 23, Switzerland

Abstract

Both LaTeX and HTML are languages that can express the structure of a document, and similarities between these two systems are shown. A detailed study is made of the `LaTeX2HTML` program, written by Nikos Drakos, that is today the most complete utility for translating LaTeX code into HTML, providing a quasi-automatic translation for most elements. A discussion of a few other tools for translating between HTML and LaTeX concludes the article.

1 Similarities between LaTeX and HTML

HTML and LaTeX are both generic markup systems, and a comparison between tags for structural elements in both cases is shown in Table 1. In most cases the differences are trivial, seeming to indicate that, at first approximation, translating between these two systems should not prove too difficult.

The translation programs described in this article use these similarities, but in order to exploit the richness of the LaTeX language as compared to HTML (especially HTML2, which has no support for tables or mathematics), an *ad hoc* approach has to be adopted. To handle correctly LaTeX commands that have no equivalent in HTML, such elements can either be transformed into bitmap or PostScript pictures (an approach taken by `LaTeX2HTML`), or the user can specify how the given element should be handled in the target language.

2 Converting LaTeX into HTML

Before discussing the `LaTeX2HTML` program, we want to mention a few other programs. First there is `l2x`[1], written by Henning Schulzrinne (Berlin, Germany), which translates LaTeX into various other formats. This program is written in C and calls a Tcl function (Ousterhout 1994) for each LaTeX command.

A converter `html.tcl` is available for translating LaTeX files into HTML, by writing, for instance:

```
l2x -p html.tcl article.tex
```

Presently, only a sub-set of all LaTeX commands are handled (no mathematical formulae, tables, verbatim texts, etc.), yet it is not too difficult to augment the code of the converter `html.tcl` by introducing new Tcl commands.

Schwarzkopf has developed `Hyperlatex`[2], a package written in the GNU `Emacs` Lisp language to translate documents marked up in (a subset of) LaTeX into HTML.

Another interesting tool is `tex2RTF`[3], a utility to convert from LaTeX to four other formats, including HTML. It does a relatively good job for a sub-set of LaTeX commands, but, as with the `tcl` approach of `l2x`, it cannot handle more complex structures, such as mathematical expressions and tables.

Finally, although not directly relevant to LaTeX, `texihtml`[4] translates `texinfo` sources[5] into HTML.

3 The LaTeX2HTML Converter—Generalities

`LaTeX2HTML` is a program written in the `perl` programming language[6] (Schwartz 1993; Till 1995; Wall and Schwartz 1991) by Nikos Drakos[7]. It transforms a LaTeX document into a series of HTML files linked in a way that reflects the structure of the original document.

3.1 What LaTeX2HTML is and What it is Not

`LaTeX2HTML` is a conversion tool that allows documents written in LaTeX to become part of the World Wide Web. In addition, it offers an easy migration path towards authoring complex hypermedia documents using familiar word-processing concepts.

`LaTeX2HTML` replicates the basic structure of a LaTeX document as a set of interconnected HTML files which can be explored using automatically generated navigation panels. The cross-references, citations, footnotes, the table of contents and the lists of figures and tables are also translated into hypertext links. Formatting information which has equivalent "tags" in HTML (lists, quotes, paragraph

[1] See the URL `http://info.cern.ch/hypertext/WWW/Tools/l2x.html`.

[2] The documentation is available at the URL `http://www.cs.ruu.nl/people/otfried/html/Hyperlatex/hyperlatex.html`. Otfried Schwarzkopf, who works at the University of Utrecht, can be reached via email at `otfried@cs.ruu.nl`.

[3] Written by Julian Smart (Edinburgh, Great Britain). For more information see the URL `http://www.aiai.ed.ac.uk/~jacs/tex2rtf.html`.

[4] Written in `perl` by Lionel Cons (CERN, Geneva). For more information see the URL `http://asis01.cern.ch/infohtml/texi2html.html`.

[5] `texinfo` is a TeX based markup language used for all gnu project related documentation.

[6] More information can be found in the UF/NA `perl` archive at the URL `http://www.cis.ufl.edu/perl/`.

[7] The documentation is at the URL `http://cbl.leeds.ac.uk/nikos/tex2html/doc/latex2html/latex2html.html`. One can also join the `LaTeX2HTML` mailing list by sending a message to `latex2html-request@mcs.anl.gov` with as only contents line: `subscribe`.

Description	HTML	LaTeX
Sectioning commands		
level 1	<H1>	\chapter or \section
level 2	<H2>	\section or \subsection
level 3	<H3>	\subsection or \subsubsection
level 4	<H4>	\subsubsection or \paragraph
level 5	<H5>	\paragraph or \subparagraph
level 6	<H5>	\subparagraph
new paragraph	<P>	\par
Lists		
numbered list	<OL>	\begin{enumerate}
unnumbered list	<UL>	\begin{itemize}
list element	<LI>	\item
description list	<DL>	\begin{description}
term	<DT>	\item
definition	<DD>	*text*
Highlighting text		
emphasis	<EM>text</EM>	\emph{text}
italic	<I>text</I>	\textit{text}
bold	<B>text</B>	\textbf{text}
fixed with	<TT>text</TT>	\texttt{text}

Table 1: Comparison of structural elements in HTML and LaTeX

breaks, type styles, etc.) is also converted appropriately. The remaining heavily formatted items such as mathematical equations, pictures or tables are converted to images placed automatically at the correct positions in the final HTML document.

`LaTeX2HTML` extends LaTeX by supporting arbitrary hypertext links and symbolic cross-references between evolving remote documents. It also allows the specification of *conditional text* and the inclusion of raw HTML commands. These hypermedia extensions to LaTeX are available as new commands and environments from within a LaTeX document.

3.2 Overview

The main characteristics of the `LaTeX2HTML` translator are summarized in this section.

- a document is broken into one or more components as specified by the user;
- optional, customizable navigation panels can be added to each generated page to link other parts of the document, or external documents;
- inline and displayed equations are handled as images;
- tables and figures, and all other arbitrary environments are passed on to LaTeX and included as images; these images are included inline or made available via hypertext links;
- figures or tables can be arbitrarily scaled and shown either as inlined images or "thumbnails";
- output can be generated to cope with the possibilities of various kind of browsers (for example, line browsers);
- definitions of new commands, environments, and theorems are given even when they are in external files;
- footnotes, tables of contents, lists of figures and tables, bibliographies, and the index are handled correctly;
- LaTeX cross-references and citations are transformed into hyperlinks, which work just as well inside a (sub)document as between several documents (located anywhere on the Internet);
- most LaTeX accented national characters are translated into their ISO-Latin-1 equivalent;
- hypertext links to arbitrary Internet services are recognized;
- programs running arbitrary scripts can be invoked (at the LaTeX level);
- a conditional text mechanism allows material to be included in the HTML or printed (dvi) versions only;
- similarly raw HTML material can be present in the LaTeX document (such as for specifying interactive forms);

- *common* LaTeX commands (i.e., those defined in the LaTeX Reference manual (Lamport 1994)) are handled gracefully;
- the user can define (in `perl`) functions to translate (un)known LaTeX commands in a customized way.

3.3 Using LaTeX2HTML

To use `LaTeX2HTML`, simply type

`latex2html` *options-list* ***file***`.tex`

By default a new directory "`file`" will be created to contain the generated HTML files, some log files and possibly some images.

The output from `LaTeX2HTML` can be customized using a number of command line options, as described below.

The command line options *options-list* allow one to change the default behavior of `LaTeX2HTML`. Alternatively, the corresponding `perl` variables in the initialization file `.latex2html-init` may be changed, in order to achieve the same result (see Section 3.5).

`-split` ***num*** The default is 8.
: Stop splitting sections into separate files at this depth. A value of 0 will put the document into a single HTML file.

`-link` ***num*** The default is 4.
: Stop revealing child nodes at each node at this depth. (A node is characterized by the sequence part-chapter-section-subsection-...). A value of 0 will show no links to child nodes, a value of 1 will show only the immediate descendents, etc. A value at least as big as that of the `-split` option will produce a table of contents for the tree structure, rooted at each given node.

`-external_images`
: Instead of including any generated images inside the document, leave them outside the document and provide hypertext links to them.

`-ascii_mode`
: Use only ASCII characters and do not include any images in the final output. In ASCII mode, the output of the translator can be used on character-based browsers that do not support inlined images (the `<IMG>` tag).

`-t` *top-page-title*
: Use the string *top-page-title* for the title of the document.

`-dir` *output-dir*
: Redirect the output to the *output-dir* directory.

`-no_subdir`
: Place the generated HTML files in the current directory. By default another file directory is created (or reused).

`-ps_images`
: Use links to external PostScript pictures rather than inlined `GIF` (Graphics Interchange Format) images.

`-address` *author-address*
: The address *author-address* will be used to sign each page.

`-no_navigation`
: Do not put navigation links in each page.

`-top_navigation`
: Put navigation links at the top of each page.

`-bottom_navigation`
: Put navigation links at the bottom of each page *as well* as at the top.

`-auto_navigation`
: Put navigation links at the top of each page. If the page exceeds `$WORDS_IN_PAGE` number of words (the default is 450) then put one at the bottom of the page also.

`-index_in_navigation`
: When an index exists, put a link to the index page in the navigation panel.

`-contents_in_navigation`
: When a table of contents exists, put a link to that table in the navigation panel.

`-next_page_in_navigation`
: Put a link to the next logical page in the navigation panel.

`-previous_page_in_navigation`
: Put a link to the previous logical page in the navigation panel.

`-info` *string*
: Generate a new section *About this document ...* containing information about the document being translated. The default is for generating such a section with information on the original document, the date, the user and the translator. If *string* is empty (or has the value 0), this section is not created. If *string* is non-empty, it will replace the default information in the contents of the *About this document ...* section.

`-dont_include` *file1 file2 ...*
: Do not include the specified file(s) *file1*, *file2*, etc. Such files can be package files that contain raw TeX commands that the translator cannot handle.

`-reuse`
: Images generated during a previous translation process should be reused as far as possible. This option disables the initial interactive session where

the user is asked whether to reuse the old directory, delete its contents or quit. Images which depend on context (for example, numbered tables or equations) cannot be reused and are always regenerated.

`-no_reuse`
: Do *not* reuse images generated during a previous translation. This enables the initial interactive session during which the user is asked whether to reuse the old directory, delete its contents or quit.

`-init_file` *file*
: Load the `perl` initialization script *file*. It will be loaded after `$HOME/.latex2html-init` (if it exists). It can be used to change default options.

`-no_images`
: Do not produce inlined images. If needed, the missing images can be generated "off-line" by restarting `LaTeX2HTML` with the `-images_only` option.

`-images_only`
: Try and convert any inlined images that were left over from previous runs of `LaTeX2HTML`. The advantage of using the latter two options is that the translation can be allowed to finish even when there are problems with image conversion. In addition, it may be possible to fix manually any image conversion problems and then run `LaTeX2HTML` again just to integrate the new images without having to translate the rest of the text.

`-show_section_numbers`
: Instruct `LaTeX2HTML` to show section numbers. By default, section numbers are not shown, in order to allow individual sections to be used as stand-alone documents.

`-h` Print the list of options.

3.4 Simple Use of LaTeX2HTML

To show the procedure for translating a LaTeX document into HTML, let us first look at a simple example, namely the file shown in Figure 1[8]. After running this file through LaTeX (twice, to resolve the cross-references) one obtains the output shown in Figure 2.

This same LaTeX source document is now run through `LaTeX2HTML` with the command

```
> latex2html -init_file french.pl babel.tex
```

[8] This one-page example is chosen because it is discussed in detail in Chapter 9 of Goossens et al. (1994) and at the same time shows how LaTeX2HTML handles non-English documents.

where the default options have been used apart from the fact that we want titles in French. That is why we use the option `-init_file` to load the file `french.pl`, which merely contains

```
$TITLES_LANGUAGE = "french";
 1;
```

as explained in Section 3.9.

The log messages generated by `LaTeX2HTML` are shown below.

```
This is LaTeX2HTML Version 95.1 (Fri Jan 20 1995)
    by Nikos Drakos,
Computer Based Learning Unit, University of Leeds.

OPENING /afs/cern.ch/usr/g/goossens/babel.tex

Loading /usr/local/lib/latex2html/styles/makeidx.perl
....
Reading ...
Processing macros ....+.....
Reading babel.aux ..............................
Translating ...0/8..............1/8....2/8....3/8
....4/8.............5/8............6/8...........
7/8.....8/8.....
Writing image file ...

This is TeX, Version 3.1415 (C version 6.1)
(images.tex
LaTeX2e <1994/12/01>

Generating postscript images using dvips ...
This is dvipsk 5.58e Copyright 1986, 1994
                            Radical Eye Software
' TeX output 1995.05.11:0844' -> 14024_image
(-> 14024_image001) <tex.pro><special.pro>
[1<colorcir.eps><tac2dim.eps>]
(-> 14024_image002) <tex.pro><special.pro>[2]
Writing 14024_image002.ppm
Writing img2.gif
Writing 14024_image001.ppm
Writing img1.gif

Doing section links .....
Doing table of contents .........
Doing the index ....
Done.
```

The results are shown in Figure 3. The main document is shown in the middle at the top. Numbered arrows indicate the secondary documents that are produced and to which point in the main document they are linked. The document also contains a table of contents that is not shown explicitly, since its contents are almost identical to that of the main document. Note the navigation buttons at the top of each "page". This navigation panel corresponds to the (default) option "`-top_navigation`". The navigation panel contains five push buttons:

Next to go to the *next* document,
    default option `-next_page_in_navigation`;
Up to go *up* one level;
Previous to move to the *previous* document,
    default option `-previous_page_in_navigation`;

```
\documentclass{article}
\usepackage{makeidx}
\usepackage[dvips]{graphicx}
\usepackage[french]{babel}
\makeindex
\begin{document}
\begin{center}\Large
  Exemple d'un article en fran\c{c}ais\\[2mm]\today
\end{center}
\tableofcontents
\listoffigures
\listoftables
\section{Une figure EPS}
\index{section}
Cette section montre comment inclure une figure PostScript\cite{bib-PS}
dans un document \LaTeX. La figure~\ref{Fpsfig} est ins\'er\'ee dans le
texte \`a l'aide de la commande \verb!\includegraphics{colorcir.eps}!.
\index{figure}\index{PostScript}
\begin{figure}
\centering
  \begin{tabular}{c@{\qquad}c}
    \includegraphics[width=3cm]{colorcir.eps} &
    \includegraphics[width=3cm]{tac2dim.eps}
  \end{tabular}
  \caption{Deux images EPS}\label{Fpsfig}
\end{figure}
\section{Exemple d'un tableau}
Le tableau~\ref{tab:exa} \`a la page \pageref{tab:exa}
montre l'utilisation de l'environnement \texttt{table}.
\begin{table}
\centering
  \begin{tabular}{cccccc}
    \Lcs{primo} \primo & \Lcs{secundo} \secundo & \Lcs{tertio} \tertio &
    \Lcs{quatro} \quatro & 2\Lcs{ieme}\ 2\ieme
  \end{tabular}
  \caption{Quelques commandes de l'option \texttt{french}
           de \texttt{babel}}\label{tab:exa}\index{tableau}
\end{table}
\begin{thebibliography}{99}
\index{r\'ef\'erences}
\bibitem{bib-PS}
Adobe Inc. \emph{PostScript, manuel de r\'ef\'erence (2i\`eme \'edition)}
Inter\'Editions (France), 1992
\end{thebibliography}
\printindex
\index{index}
\end{document}
```

Figure 1: Example of a LaTeX document

Exemple d'un article en français

7 décembre 1994

Table des matières

1 Une figure EPS **1**

2 Exemple d'un tableau **1**

Liste des figures

1 Deux images EPS 1

Liste des tableaux

1 Quelques commandes de l'option `french` de `babel` 1

1 Une figure EPS

Cette section montre comment inclure une figure PostScript[1] dans un document LaTeX. La figure 1 est insérée dans le texte à l'aide de la commande `\includegraphics{colorcir.eps}`.

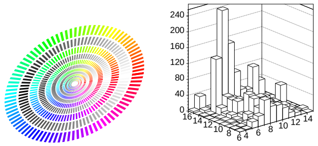


Figure 1: Deux images EPS

2 Exemple d'un tableau

Le tableau 1 à la page 1 montre l'utilisation de l'environnement `table`.

`\primo` 1°	`\secundo` 2°	`\tertio` 3°	`\quatro` 4°	2`\ieme` 2^e

Tableau 1: Quelques commandes de l'option `french` de `babel`

Références

[1] Adobe Inc. *PostScript, manuel de référence (2ième édition)* InterÉditions (France), 1992

Index

figure, 1
index, 1
PostScript, 1
références, 1
section, 1
tableau, 1

1

Figure 2: Output generated by LaTeX from document shown in Figure 1

Contents to jump directly to *Table of Contents*,
default option `-contents_in_navigation`;

Index to jump straight to the *Index*,
default option `-index_in_navigation`.

Each of the default values can be modified by redefining the corresponding `perl` variables in the initialization file `.latex2html.init`, as described in Section 3.5.4.

A detailed explanation of the meaning of the various numbers in Figure 3 is given below.

- ❶ the list of figures, containing a hyperlink pointing to document ❸ (containing the figure in question);
- ❷ the list of tables, containing a hyperlink pointing to document ❹ (containing the table in question);
- ❸ the first section, containing some text, a figure, and a hyperlink (`[1]`) pointing to an entry in the bibliography (document ❺);
- ❹ the second section, also containing some text and a table;
- ❺ the bibliographic references;
- ❻ the index, containing keywords that provide hyperlinks pointing to entry points in the various documents;
- ❼ an explanatory note detailing the procedure by which the document was translated into HTML. This text can be customized with the help of the option `-desc` (see Section 3.6.3).

3.5 Extending and Customizing the Translator

As the translator only partially covers the set of LaTeX commands and because new LaTeX commands can be defined arbitrarily using low level TeX commands, the translator should be flexible enough to allow end users to specify how they want particular commands to be translated.

3.5.1 Adding Support for Packages

`LaTeX2HTML` provides a mechanism to automatically load files containing code to translate specific packages. For instance, when in a LaTeX document, the command `\includegraphics{xxxx}` is found, a file `LATEX2HTMLDIR/styles/xxxx.perl` is looked for. If such a file exists, it will be loaded into the main script.

This mechanism helps keep the core script smaller and modular and also makes it easier for others to contribute `perl` code to translate specific packages. The current distribution includes the files `german.perl`, `french.perl`, `makeidx.perl`, and for the hypertext extensions `html.perl`. Note, however, that writing such extensions requires an understanding of `perl` and of the way `LaTeX2HTML` is organized. Some more details will be given in Section C.

Presently, the user can ask that particular commands and their arguments be ignored or passed on to LaTeX for processing (the default behavior for unrecognized commands is for their arguments remains in the HTML text). Commands passed to LaTeX are converted to images that are either "inlined" in the main document or are accessible via hypertext links. Simple extensions using the commands below may be included in the system initialization file `LATEX2HTMLDIR/latextohtml.config`, or in the customization initialization file `.latex2html-init` in the user's home directory or in the directory where the files to be converted reside.

3.5.2 Directing the Translator to Ignore Commands

Commands that should be ignored may be specified in the `.latex2html-init` file as input to the `ignore_commands` subroutine. Each command which is to be ignored should be on a separate line followed by compulsory or optional argument markers separated by #'s, for example[9]:

```
<cmd_name>#{}# []# {}# [] ...
```

`{}`'s mark compulsory arguments and `[]`'s optional ones.

Some commands may have arguments which should be left as text, even though the command should be ignored (`\mbox`, `\center`, etc.). In these cases the arguments should be left unspecified.

Here is an example of how this mechanism may be used:

```
&ignore_commands( <<_IGNORED_CMDS_);
documentclass # [] # {}
linebreak# []
pagebreak# []
center
<add your commands here>
_IGNORED_CMDS_
```

3.5.3 Asking the Translator to Pass Commands to LaTeX

Commands that should be passed on to LaTeX for processing because there is no direct translation to HTML may be specified in the `.latex2html-init`

[9] It is possible to add arbitrary `perl` code between any of the argument markers that will be executed when the command is processed. For this, however, a basic understanding of how the translator works and, of course, `perl` is required.

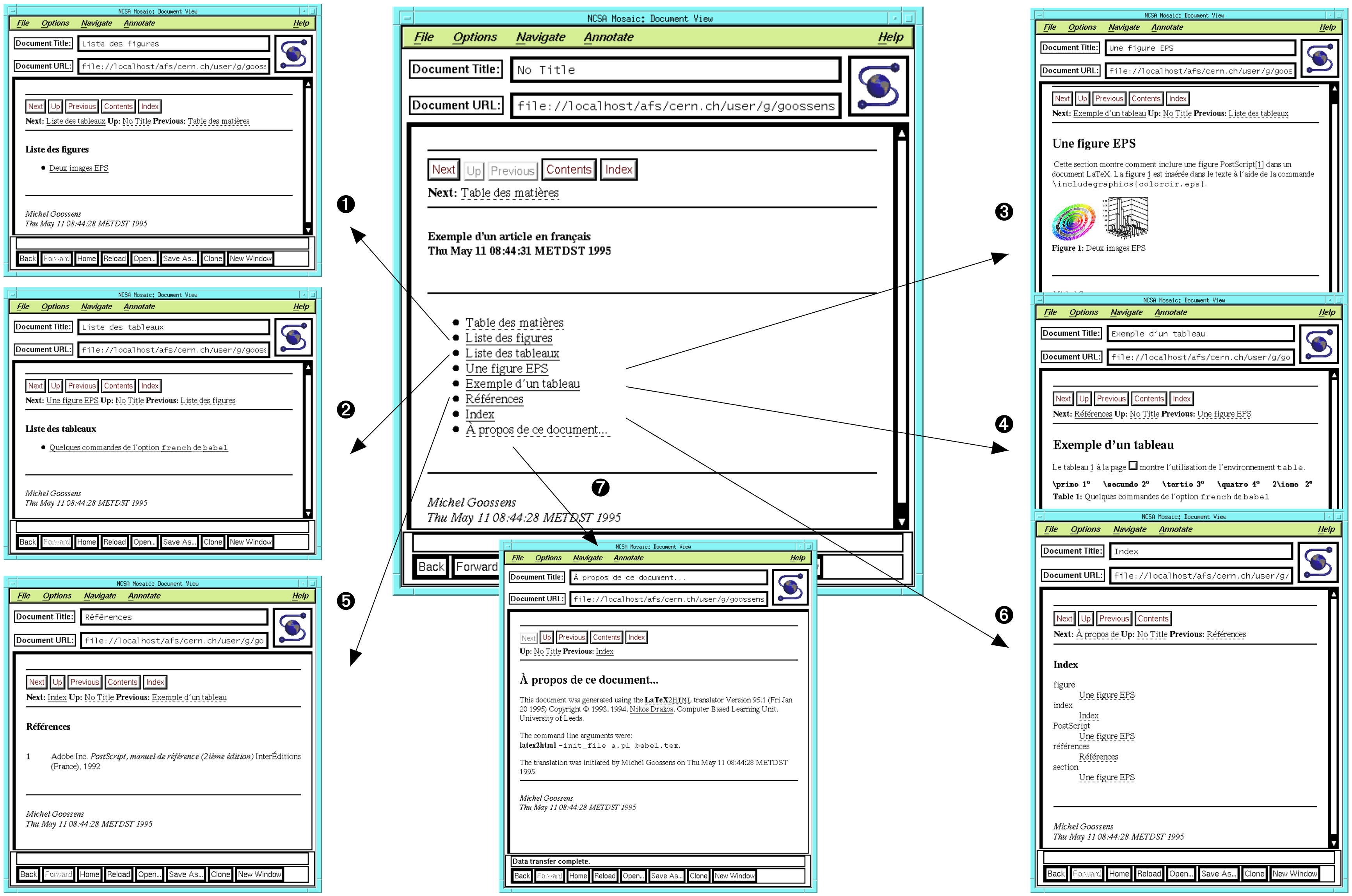


Figure 3: The HTML structure generated from the LaTeX source of Figure 1 as visualized with the Mosaic browser

file as input to the `process_commands_in_tex` subroutine. The format is the same as that for specifying commands to be ignored. Here is an example:

```
&process_commands_in_tex (<<_RAW_ARG_CMDS_);
fbox # {}
framebox # [] # [] # {}
<add your commands here>
_RAW_ARG_CMDS_
```

3.5.4 Customizing LaTeX2HTML

Besides honoring the options specified on the command line, `LaTeX2HTML` reads two standard files that can be used to customize its behavior. The first file, `latextohtml.config`, is a system-wide file (usually in the directory `/usr/local/lib/latex2html`). It contains the definitions for a complete installation, i.e., those common for all users, and specifies where certain external utility programs needed by `LaTeX2HTML` are to be found on the system (such as LaTeX, `dvips`, `gs`, `pmbplus`). Moreover, in this file important `perl` variables are initialized to their default values. At the end of the file one has the possibility of specifying those LaTeX commands or environments that should be ignored, and those that should be passed on to LaTeX to be transformed into images for inclusion in the HTML file.

The second file, `.latex2html-init`, allows the user to customize `LaTeX2HTML` on an individual level. `LaTeX2HTML` will normally look for this file in the user's home directory (variable `$HOME` on Unix). This file can contain the same information as the global configuration file `latextohtml.config` and is thus the ideal place to overwrite default values or to specify in the `perl` language how certain specific LaTeX commands should be handled. It should be noted that the `LaTeX2HTML` distribution `LaTeX2HTML` already contains a few files with definitions for translations of supplementary LaTeX commands introduced by certain extension packages, such as `german.perl`, `french.perl`, `html.perl` and `makeidx.perl`. To help the user, the distribution comes with an example file `dot.latex2html-init` that can serve as a model for writing one's own `.latex2html-init`.

3.5.5 Creating a Customization File `.latex2html-init`

Before discussing examples of commands that can be put in the `.latex2html-init` customization file, it should be emphasized once more that this file, as well as all other files that are part of the `LaTeX2HTML` system, contain only `perl` instructions, and that one should thus have at least a basic understanding of this language before trying to edit any of these files.

Figures 4 and 5 show the contents of the example initialization file `dot.latex2html-init`. Its first parts initialize most of the `perl` variables used by the `LaTeX2HTML` system by setting them equal to their default values (as defined in the system-wide initialization file `latex2html.config`. Values that need not to be changed can be deleted from the file. When studying the various system variables, note the correspondence between the `perl` variables and the options of the `latex2html` described in Section 3.3).

Examples

We want to leave most of the values at their defaults as shown in Figures 4 and 5. However, we specify the format of the address fields explicitly and make a few more modifications; in particular, we do not want images to be included inside the HTML documents. Thus we should write something like:

```
$ADDRESS = "<I>Michel Goossens<BR>" .
           "CN Division<BR>"        .
           "Tel. 3363<BR>"          .
           "\n$address_data[1]</I>";
$MAX_SPLIT_DEPTH = 2; # stop at subsection
$MAX_LINK_DEPTH = 1;  # child nodes to sections
$EXTERNAL_IMAGES = 1; # images outside document
# draw a nice rainbow-colored line (gif file)
# instead of the default simple line (<HR>)
$CHILDLINE = "<BR><IMG "  .
                      "SRC=rainbow_line.gif><BR>"
```

Normally, `LaTeX2HTML` will read all package and class files and interpret all the commands that are defined in those files. This can lead to problems, so it is wise to exclude some files. Also, one may want to define a translation into `perl` for the commands in one or more files, so they should also not be read. The list of files to be excluded, is specified as follows:

```
$DONT_INCLUDE = "memo:psfig:times:revtex:" .
                "aps:float:harvard:tabls";
```

Special symbols and inline equations are generally transformed into inlined (bitmap) images that are placed inside the HTML text on the same line when viewing the document with a browser. On the other hand, display environments, such as tables, figures, minipages, and multi-line equations are transformed into images that will also be shown on a line by themselves after starting a new paragraph. The scale factor for the two kinds of images (inline and displayed) can be specified by the following parameters:

```
$MATH_SCALE_FACTOR = 1.6;# inline images
$FIGURE_SCALE_FACTOR = 0;# display images
                     # = 0, original dimension
```

```
#LaTeX2HTML Version 95.1 : dot.latex2html-init
#
### Command Line Argument Defaults #######################################

$MAX_SPLIT_DEPTH = 8;          # Stop making separate files at this depth
$MAX_LINK_DEPTH = 4;             # Stop showing child nodes at this depth
$NOLATEX = 0;                    # 1 = do not pass unknown environments to Latex
$EXTERNAL_IMAGES = 0;            # 1 = leave the images outside the document
$ASCII_MODE = 0;                 # 1 = do not use any icons or internal images
$PS_IMAGES = 0;                  # 1 =  use links to external postscript
                                 #      images rather than inlined GIF's.
$TITLE = $default_title;         # The default is "No Title"
$DESTDIR = '.';                  # Put the result in this directory
$NO_SUBDIR = 0;                  # 0 = create (reuse) file subdirectory
                                 # 1 = put generated HTML files in current dir.
# Supply your own string if you don't like the default <Name> <Date>
$ADDRESS = "<I>$address_data[0] <BR>\n$address_data[1]</I>";
$NO_NAVIGATION = 0;          # 1 = no navigation panel at top of each page
$AUTO_NAVIGATION = 1;            # 1 = put navigation links at top of page
$WORDS_IN_PAGE = 300;            # if nb. words on page > $WORDS_IN_PAGE put
                                 # navigation panel at bottom of page.
$INDEX_IN_NAVIGATION = 1;        # put link to index page in navigation panel
$CONTENTS_IN_NAVIGATION = 1;     # put link to table of contents   " "  "
$NEXT_PAGE_IN_NAVIGATION = 1;    # put link to next logical page  " "  "
$PREVIOUS_PAGE_IN_NAVIGATION = 1;# put link to prev. "   "    " "  "
$INFO = 1;                       # 0 = do not make "About this document..." section
$REUSE = 1;                      # Reuse images generated during previous runs

# Do not try to translate these package files.
# Complex LaTeX packages may cause the translator to hang.
# If this occurs add the package's filename here.
$DONT_INCLUDE = "memo:psfig:pictex:revtex";

# When this is 1, the section numbers are shown. The section numbers should
# then match those that would have bee produced by LaTeX.
# The correct section numbers are obtained from the $FILE.aux file generated
# by LaTeX.
# Hiding the section numbers encourages use of particular sections
# as standalone documents. In this case the cross reference to a section
# is shown using the default symbol rather than the section number.
$SHOW_SECTION_NUMBERS = 0;

### Other global variables ###############################################
$CHILDLINE = "<BR> <HR>\n";

# This is the line width measured in pixels and it is used to right justify
# equations and equation arrays;
$LINE_WIDTH = 500;

# Affects ONLY the way accents are processed
$default_language = 'english';

# This number will determine the size of the equations, special characters,
# and anything which will be converted into an inlined image
# *except* "image generating environments" such as "figure", "table"
# or "minipage".
# Effective values are those greater than 0.
# Sensible values are between 0.1 - 4.
$MATH_SCALE_FACTOR = 1.6;

# This number will determine the size of
# image generating environments such as "figure", "table" or "minipage".
# Effective values are those greater than 0.
# Sensible values are between 0.1 - 4.
$FIGURE_SCALE_FACTOR = 0;
```

Figure 4: `dot.latex2html-init` file (part 1)

```
#  If this is set then intermediate files are left for later inspection.
#  This includes $$_images.tex and $$_images.log created during image
#  conversion.
#  Caution: Intermediate files can be *enormous*.
$DEBUG = 0;

# The value of this variable determines how many words to use in each
# title that is added to the navigation panel (see below)
#
$WORDS_IN_NAVIGATION_PANEL_TITLES = 4;

#  If both of the following two variables are set then the "Up" button
#  of the navigation panel in the first node/page of a converted document
#  will point to $EXTERNAL_UP_LINK. $EXTERNAL_UP_TITLE should be set
#  to some text which describes this external link.
$EXTERNAL_UP_LINK = "";
$EXTERNAL_UP_TITLE = "";

# If this is set then the resulting HTML will look marginally better if viewed
# with Netscape.
$NETSCAPE_HTML = 0;

# Valid paper sizes are "letter", "legal", "a4","a3","a2" and "a0"
# Paper sizes has no effect other than in the time it takes to create inlined
# images and in whether large images can be created at all ie
#  - larger paper sizes *MAY* help with large image problems
#  - smaller paper sizes are quicker to handle
$PAPERSIZE = "a4";

# Replace "english" with another language in order to tell LaTeX2HTML that you
# want some generated section titles (eg "Table of Contents" or "References")
# to appear in a different language. Currently only "english" and "french"
# is supported but it is very easy to add your own. See the example in the
# file "latex2html.config"
$TITLES_LANGUAGE = "english";

### Navigation Panel ##########################################################
#
# The navigation panel is constructed out of buttons and section titles.
# These can be configured in any combination with arbitrary text and
# HTML tags interspersed between them.
# The buttons available are:
# $PREVIOUS - points to the previous section
# $UP  - points up to the "parent" section
# $NEXT - points to the next section
# $NEXT_GROUP - points to the next "group" section
# $PREVIOUS_GROUP - points to the previous "group" section
# $CONTENTS - points to the contents page if there is one
# $INDEX - points to the index page if there is one
#
# If the corresponding section exists the button will contain an
# active link to that section. If the corresponding section does
# not exist the button will be inactive.
#
# Also for each of the $PREVIOUS $UP $NEXT $NEXT_GROUP and $PREVIOUS_GROUP
# buttons there are equivalent $PREVIOUS_TITLE, $UP_TITLE, etc variables
# which contain the titles of their corresponding sections.
# Each title is empty if there is no corresponding section.
#
# The subroutine below constructs the navigation panel in each page.
# Feel free to mix and match buttons, titles, your own text, your logos,
# and arbitrary HTML (the "." is the Perl concatenation operator).
sub navigation_panel {....}
1; # This must be the last line
```

Figure 5: `dot.latex2html-init` file (part 2). The navigation panel `perl` code is shown in Figure 9.

Finally, we specify—as described in Sections 3.5.2 and 3.5.3—a list of commands to be ignored and passed to LaTeX.

```
### Commands to ignore ##############

&ignore_commands( <<_IGNORED_CMDS_);
documentclass # [] # {}
usepackage # [] # {}
mbox
makebox# [] # []
_IGNORED_CMDS_

### Commands to pass on to LaTeX{} ##

&process_commands_in_tex (<<_RAW_ARG_CMDS_);
includegraphics # [] # [] # {}
rotatebox # {} # {}
_RAW_ARG_CMDS_

1; # This MUST be the last line
```

We notice that the mandatory argument of the `\mbox` and `\makebox` commands is not specified, so that it will end up in the text, while the optional arguments of the `\makebox` command will disappear. In the case of the framed box commands `\fbox` and `\framebox`, both mandatory and optional arguments are passed on to LaTeX.

It is important to note that the last line of the file *must* be the one shown in the example above.

3.6 Hypertext Extensions

These commands are defined in the html.sty package file that is part of the distribution. They are defined as LaTeX commands that are (mostly) ignored during the LaTeX run but are activated in the HTML version. To use them the `html` package must be included with a `\usepackage` command.

3.6.1 Hyperlinks in LaTeX

With the `\htmladdnormallink` and `\htmladdimg` commands one can build arbitrary hypertext references.

`\htmladdnormallink{`*text*`}{`⟨*URL*⟩`}`

When processed by LaTeX the URL part will be ignored, but `LaTeX2HTML` will transform it into an active hypertext link that can give access to sound, images, other documents, etc., for instance,

```
\htmladdnormallink{The $\Omega$ Project}
{http://www.ens.fr/omega/}
```

`\htmladdnormallinkfoot{`*text*`}{`⟨*URL*⟩`}`

This command takes the same two arguments and has the same effect when generating HTML as the command `\htmladdnormallink`, but when processed by LaTeX it shows the URL as a footnote.

`\htmladdimg{`⟨*URL*⟩`}`

In a similar way, the argument of the `\htmladdimg` command should be a URL pointing to an image. This URL is ignored in the LaTeX hard copy output.

3.6.2 Cross-References between Living Documents

In this case we want to use a mechanism for establishing cross-references between two or more documents via symbolic labels independent of the physical realisation of these documents. The documents involved may reside in remote locations and may be spread across one or more HTML files.

The mechanism is an extension of the simple `\label`-`\ref` pairs that can be used only within a single document. A symbolic label defined with a `\label` command can be referred to using a `\ref` command. When processed by LaTeX, each `\ref` command is replaced by the section number at which the corresponding `\label` occurred. When processed by `LaTeX2HTML` each `\ref` is replaced by a hypertext link to the place where the corresponding `\label` occurred. The new commands, detailed below, show how it is possible to refer to symbolic labels defined with `\label` in other external documents. Such references to external symbolic labels are then translated into hyperlinks pointing to the external document.

Cross-references between documents are established with the commands:

`\externallabels`
`{`⟨*URL directory external document*⟩`}`
`{`⟨*external document labels.pl file*⟩`}`
`\externalref{`⟨*label in remote document*⟩`}`

The first argument to `\externallabels` should be a URL to the directory containing the external document. The second argument should be the full pathname to the `labels.pl` file belonging to the external document. The file `labels.pl` associates symbolic labels with physical files and is generated automatically for each translated document. For *remote* external documents it is necessary to copy the `labels.pl` file locally so that it can be read when processing a local document that uses it. The command `\externallabels` should be used once for each external document in order to import the external labels into the current document. The argument to `\externalref` can be any symbolic label defined in any of the external documents in the same way that the `\ref` command refers to labels defined internally.

After modifications in an external document, such as addition or deletion of sectioning levels, or a segmentation into different physical parts, a new file, `labels.pl`, will be generated. If in another document the `\externallabels` command contains the correct address to the `labels.pl` file, then cross-references will be realigned correctly. A warning will be given if `labels.pl` cannot be found.

3.6.3 Example of a Composite Document

In this section we show how to handle a set of composite documents taking advantage of the hypertext extensions described in Section 3.6.

We start with the LaTeX source document shown in Figure 1 and divide it, for demonstration purposes, into four sub-documents, shown in Figure 6, namely a main file (`ex20.tex`) and three secondary files (`ex21.tex`, `ex22.tex` and `ex2bib.tex`). We must first run all these files through LaTeX and then *in the correct order* through **LaTeX2HTML**. Indeed, as we use cross-references to refer to document elements in external documents (with the commands `\externalref` and `\externallabels`) we should first treat the secondary files `ex21.tex`, `ex22.tex`, and `ex2bib.tex`, before tackling the master file `ex20.tex`.

By default, **LaTeX2HTML** writes the files that it creates into a subdirectory with the same name as the original file, for example, after execution of the command:

```
> latex2html ex20.tex
```

one ends up with a directory `ex20` containing all files associated with the translation of the input file `ex20.tex`. Figure 7 shows the structure of the four sub-directories created.

To guide **LaTeX2HTML** in translating these documents we also used a customization file, `myinit.pl`, containing some redefinitions of `perl` constants.

```
# File myinit.pl
# Customization of latex2html
$ADDRESS = "<I>Michel Goossens<BR>" .
           "Division CN<BR>"        .
           "Tél. 3363<BR>"          .
           "\n$address_data[1]</I>";
$MAX_SPLIT_DEPTH = 0; # do not split document
$MAX_LINK_DEPTH = 0;  # no down links needed
$NO_NAVIGATION = 1;   # no navigation panel

1; # Mandatory last line
```

When executing **LaTeX2HTML** on the files we then issued the following command:

```
> latex2html -init_file myinit.pl \
>            -t "Image" \
```

```
Top directory (TeX source files)
================================

569  ex20.tex
721  ex21.tex
627  ex22.tex
322  ex2bib.tex

Subdirectories (generated HTML files)
=====================================

       ex20
       ----
 1187  ex20.html
  109  images.pl
   93  labels.pl

       ex21
       ----
 1755  T_18854_figure15.gif
12118  _18854_figure15.gif
  122  _18854_tex2html_wrap57.gif
 1345  ex21.html
  539  images.pl
  589  images.tex
  190  labels.pl

       ex22
       ----
  624  _15561_table12.gif
 1047  ex22.html
  512  images.pl
  687  images.tex
  191  labels.pl

       ex2bib
       ------
  844  ex2bib.html
  109  images.pl
  141  labels.pl
```

Note the presence of the files `labels.pl` that contain information associating the symbolic names used on the `\label` commands in the original LaTeX source documents with the physical documents. The other files are one or more HTML files that were created by the translation process. `GIF` images are generated for all environments that `LaTeX2HTML` cannot translate gracefully into HTML. In this case the relevant part of the LaTeX code is first copied into a file `images.tex` that is run through LaTeX, which places each such environment on a separate page, so that the `dvips` program can produce a PostScript picture for each that is then (in principle) translated into `GIF` by using the `Ghostscript` program (see Section A.1 for more information about all these programs)

Figure 7: Subdirectory structure after translation of the four documents shown in Figure 6

```
\documentclass{article}
\usepackage{html}
\usepackage[dvips]{graphicx}
\usepackage[french]{babel}
\begin{document}
\begin{center}
\Large Exemple d'un document compos\'e
\end{center}

\htmladdnormallink{Les Images}%
           {../ex21/ex21.html}
\externallabels{../ex21}%
     {../ex21/labels.pl}
R\'ef\'erence \`a une figure
externe~\externalref{Fpsfig}.

\htmladdnormallink{Les tableaux}%
             {../ex22/ex22.html}
\externallabels{../ex22}%
               {../ex22/labels.pl}
R\'ef\'erence \`a un tableau
externe~\externalref{tab-exa}.

\htmladdnormallink{La bibliographie}%
             {../ex2bib/ex2bib.html}
\end{document}
```

Master file (**ex2.tex**)

```
\documentclass{article}
\usepackage{html}
\usepackage[dvips]{graphicx}
\usepackage[french]{babel}
\makeindex
\begin{document}
\section{Une figure EPS}\label{sec-figure}
Cette section montre comment inclure une
\externallabels{../ex2bib}%
     {../ex2bib/labels.pl}%
figure PostScript\externalref{bibPS}
dans un document \LaTeX. La
\hyperref{figure}{figure }{}{Fpsfig}
est ins\'er\'ee dans le texte \`a l'aide
de la commande
\verb!\includegraphics{colorcir.eps}!.
\begin{figure}\centering
\htmlimage{thumbnail=0.2}
 \begin{tabular}{c@{\qquad}c}
  \includegraphics[width=6cm]{colorcir.eps} &
  \includegraphics[width=6cm]{tac2dim.eps}
 \end{tabular}
 \caption{Deux images EPS}\label{Fpsfig}
\end{figure}
\end{document}
```

File containing images (**ex21.tex**)

```
\documentclass{article}
\usepackage{html}
\usepackage[french]{babel}
\newcommand{\Lcs}[1]{%
    \texttt{\symbol{'134}#1}\enspace}
\begin{document}
\section{Exemple d'un tableau}
\label{sec-tableau}
Le \hyperref{tableau}{tableau }{}{tab-exa}
montre l'utilisation de l'environnement
\texttt{table}.
\begin{table}\centering
  \begin{tabular}{ccccc}
    \Lcs{primo} \primo     &
    \Lcs{secundo} \secundo &
    \Lcs{tertio} \tertio   &
    \Lcs{quatro} \quatro   &
    2\Lcs{ieme}\ 2\ieme
  \end{tabular}
  \caption{Quelques commandes de l'option
           \texttt{french} de
           \texttt{babel}}\label{tab-exa}
\end{table}
\end{document}
```

File containing the table (**ex22.tex**)

```
\documentclass{article}
\usepackage{html}
\usepackage[french]{babel}
\makeindex
\begin{document}
\begin{thebibliography}{99}
\bibitem{bib-PS}\label{bibPS}
Adobe Inc. \emph{PostScript, manuel de
r\'ef\'erence (2i\`eme \'edition)}
Inter\'Editions (France), 1992
\end{thebibliography}
\end{document}
```

File with the bibliography (**ex2bib.tex**)

Figure 6: Example of a composite document (LaTeX files)

```
>           -info "Test du 2/12/94" \
>           ex21.tex
```

Apart from loading our customization file `moninit.pl` (option `-init_file`), we also specify a title for the document (option `-t`), and add a description about the document (option `-info`). The result can be seen in the upper left corner of Figure 8.

Shown below are the informative messages generated by **LaTeX2HTML** when executing the above command. At first the file `html.perl` associated with the hypertext extensions described in Section 3.6 is loaded (thanks to the `\usepackage{html}` command as seen in the source in Figure 6). The auxiliary file `ex21.aux` is also read, thus reminding us that the documents should be treated by LaTeX before **LaTeX2HTML** is run. After reading the complete LaTeX input file, **LaTeX2HTML** generates the file `image.tex` which contains the LaTeX code corresponding to all environments for which no translation rules were available. After running LaTeX on `images.tex` the `dvi` file is transformed by the `dvips` program into PostScript. Then another program, `ghostview`, interprets this PostScript and transforms it into the `GIF` format (via an intermediate stage using the ppm format). It is these `GIF` images that are used by most browsers to show the images on screen. At the end, **LaTeX2HTML** reads the file(s) containing the labels of the external documents in order to resolve possible cross-references by including the necessary `<URL>` addresses.

```
This is LaTeX2HTML Version 0.6.4 (Tues Aug 30 1994)
          by Nikos Drakos,
Computer Based Learning Unit, University of Leeds.

OPENING /afs/cern.ch/usr/goossens/html/ex21.tex

Loading /usr/local/lib/latex2html/styles/html.perl...

Reading ...
Reading ex21.aux ..........................
Translating ...0/2.........1/2.......2/2......
Generating images using LaTeX ...
This is TeX, Version 3.1415 (C version 6.1)
(18854_images.tex
LaTeX2e <1994/06/01> patch level 3
Hyphenation patterns for english, loaded.

Generating postscript images using dvips ...
This is dvipsk 5.58c Copyright 1986, 1994
                          Radical Eye Software
' TeX output 1994.12.02:1830' -> 18854_image
(-> 18854_image001) <tex.pro><special.pro>[1]
(-> 18854_image002) <tex.pro>
<special.pro>[2<colorcir.eps><tac2dim.eps>]
GS>GS>Writing 18854_image001.ppm
GS>Writing _18854_tex2html_wrap57.gif
GS>GS>Writing 18854_image002.ppm
GS>Writing _18854_figure15.gif
GS>GS>Writing 18854_image002.ppm
GS>Writing T_18854_figure15.gif
Doing section links ....
Done.
```

The result of all our efforts is shown in Figure 8.

3.7 Including arbitrary HTML Markup

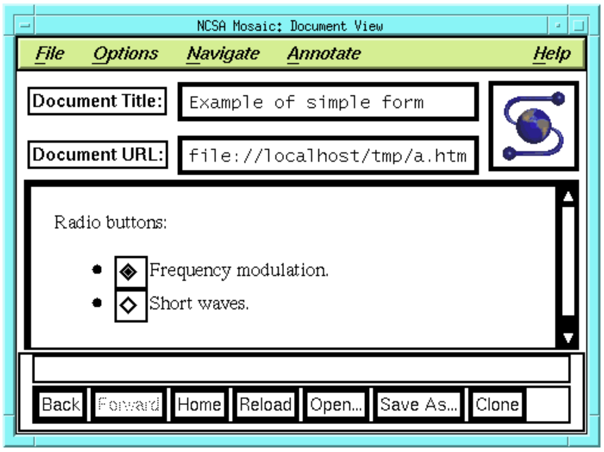
Raw HTML tags and text may be included using the `rawtext` environment. An interesting use of this feature is in the creation of interactive electronic forms. from within a LaTeX document. When producing the paper (DVI) version of a document the `rawhtml` environment is ignored.

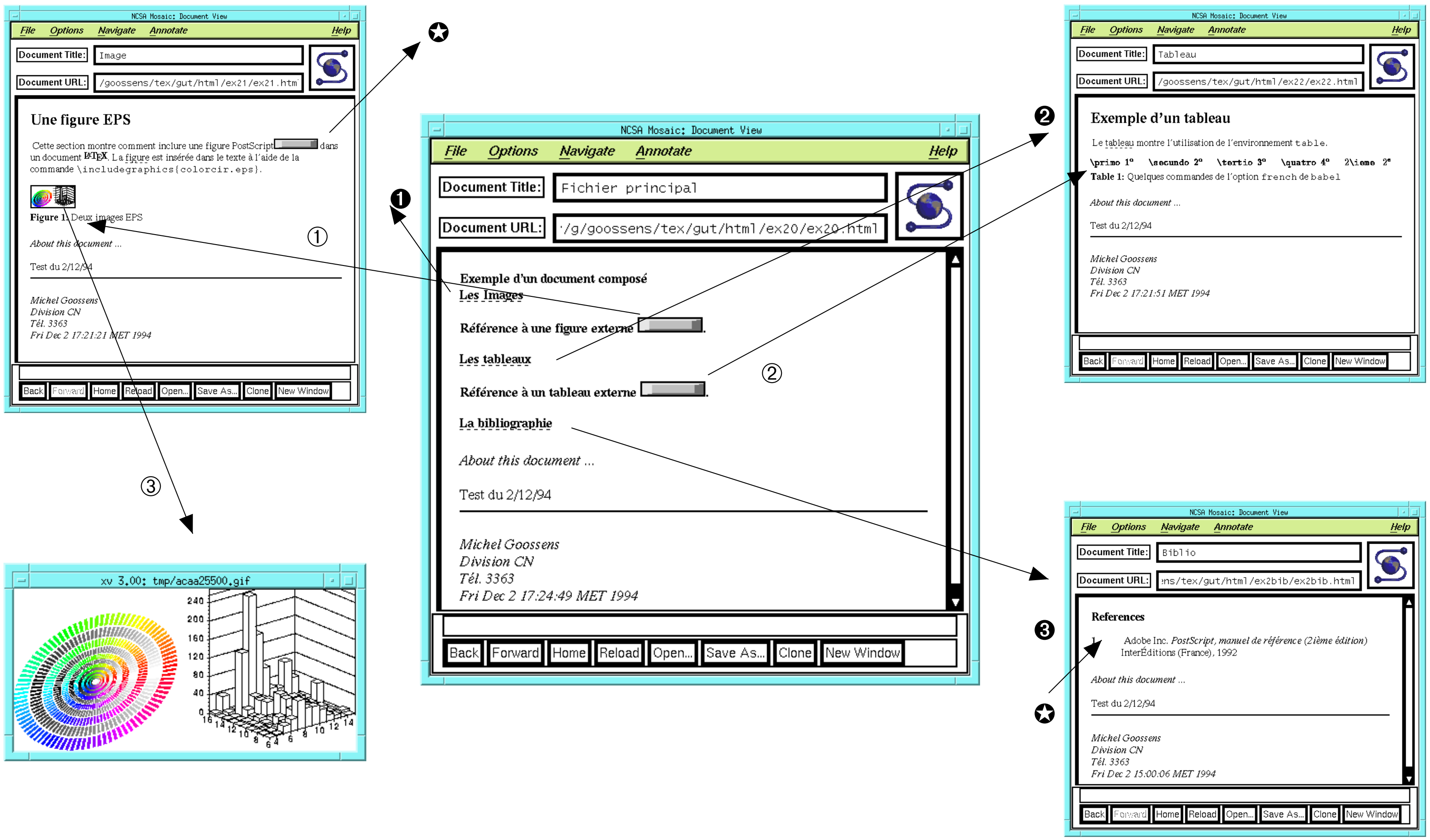
Here is an example:

```
\begin{rawhtml}
<HTML>
<HEAD>
<TITLE>Example of simple form</TITLE>
</HEAD>
<BODY>
<FORM
  ACTION="http://www.server.ch/form.cgi"
  METHOD="POST">
  Radio buttons:
<UL>
<LI> <INPUT TYPE="radio" NAME="mode"
       VALUE="FM"> Frequency modulation.
<LI> <INPUT TYPE="radio" NAME="mode"
       VALUE="SW" CHECHED> Short waves.
</UL>

</FORM>
</BODY>
</HTML>
\end{rawhtml}
```

The result of running this electronic form with Mosaic would yield something like:





As requested, there are no navigation pannels, the titles and the information part *About this document ...* have been customized as indicated in the file `myinit.pl`. The arrows carrying the numbers ❶, ❷, and ❸ correspond to hyperlinks pointing to an HTML document using the `\htmladdnormallink` command in the LaTeX source. The arrows numbered ① and ② are cross-references constructed with the commands `\externalref`, that make use of symbolic names specified as the argument of `\label` commands in the target documents. The arrow numbered ③ corresponds to a hyperlink connecting the thumbnail in the text with the real-size image available as a separate external `gif` file. Finally, the start and end points of the bibliographic reference link are indicated by the symbol ✪.

Figure 8: The HTML file structure obtained from the composite document and its sub-documents (Figure 6) as viewed by the `Mosaic` browser.

3.7.1 Conditional Text

Conditional text can be specified using the environments `latexonly` and `htmlonly`. These allow the writing of parts of a document intended only for electronic delivery or only for paper-based delivery.

This would be useful in, for example, adding a long description of a multi-media resource to the paper version of a document. Such a description would be redundant in the electronic version, as the user can have direct access to this resource.

Using LaTeX commands involving counters (for example, numbered figures or equations) in conditional texts may cause problems with the values of the counters in the electronic version.

3.7.2 Cross-References shown as "Hyperized" Text

In printed documents cross-references are shown by *numerical or symbolic indirection*, such as "see equation 3.1a" (numeric indirection), or "see chapter "Hypertext" (symbolic indirection). In a hypertext document, however, cross-references can be shown without any indirection by using highlighting of a relevant piece of text. This can contribute to making a document more readable by removing unnecessary visual information.

With `LaTeX2HTML` one can use the `\hyperref` command to specify how a cross-reference should appear in the printed and hypertext versions of a document.

`\hyperref{`*text-h*`}{`*text-d1*`}{`*text-d2*`}{`*label*`}`

The meaning of the four arguments is as follows:

text-h	text to be highlighted in the hypertext version;
text-d1	text to be shown in the printed version followed by a numeric reference;
text-d2	text following the reference text;
label	the label defining the cross-reference.

```
Example of the use of hyperref, with a
\hyperref
{reference to conditional text}
{reference to conditional text
  (see Section }
{ for more information)}
{sec:latexonly}
as an example.
```

Here is how it will be printed:

Example of the use of hyperref, with a reference to conditional text (see Section 3.7.1 for more information) as an example.

In the hypertext version what would appear is:

Example of the use of hyperref, with a <u>reference to conditional text</u> as an example.

A simpler version of the above command but having the same effect for the HTML version:

`\htmlref{`*text*`}{`*label*`}`

In the HTML version the text will be "hyperized" pointing to the label, while in the printed version the text will be shown as it is and the label ignored.

3.7.3 Customizing the Navigation Panel

The navigation panel is the strip containing "buttons" and text that appears at the top and possibly at the bottom of each generated page and that provides hypertext links to other sections of a document. Some of the options and variables that control whether and where it should appear have already been mentioned in Section 3.3.

A simple mechanism for appending customized buttons to the navigation panel is provided by the command, `\htmladdtonavigation`. This takes one argument, which `LaTeX2HTML` appends to the navigation panel:

```
\htmladdtonavigation
 {\htmladdnormallink
  {\htmladdimg{http://server/mybutton.gif}}
  {http://server/link}}
```

For example, the above will add an active button `mybutton.gif` pointing to the specified location.

It is also possible to completely specify what is to appear in the navigation panel and its order of appearance. As each section is processed, `LaTeX2HTML` assigns relevant information to a number of global variables. These variables are used by the subroutine `navigation_panel` where the navigation panel is constructed as a string consisting of these variables and some formatting information.

This subroutine can be redefined in the system or user configuration files `HOME/.latex2html-init` and `LATEX2HTMLDIR/latex2html.config`.

The list below contains the names of control panel variables that relate to navigation icons and explains where they point to.

`CONTENTS`	contents page (if it exists);
`INDEX`	index page (if it exists).
`NEXT`	next section;
`PREVIOUS`	previous section;
`UP`	"parent" section;
`NEXT_GROUP`	next "group" section;
`PREVIOUS_GROUP`	previous "group" section.

The list below contains the names of textual links that point to the title information associated with the control panel variables described above.

`NEXT_TITLE`	next section;
`PREVIOUS_TITLE`	previous section;
`UP_TITLE`	"parent" section;
`NEXT_GROUP_TITLE`	next "group" section;
`PREVIOUS_GROUP_TITLE`	previous "group" section.

If the corresponding section exists, each iconic button will contain an active link to that section. If the corresponding section does not exist, the button will be inactive. If the section corresponding to a textual link does not exist then the link will be empty.

The number of words in each textual link is controlled by the `WORDS_IN_NAVIGATION_PANEL_TITLES` variable, which may also be changed in the configuration files. Figure 9 shows an example of a navigation panel.

3.8 Image Conversion

`LaTeX2HTML` converts equations, special accents, external PostScript files, and LaTeX environments it cannot directly translate into inlined images. It is possible to control the final appearance of such images, both for inline and display-type constructs.

The size of all "inline" images depends on a configuration variable, `MATH_SCALE_FACTOR`, which specifies how much to enlarge or reduce them in relation to their original size in the printed version of the document, i.e., scale factors of 0.5 or 2.0 make all images half or twice as large as the original. A value of 0.0 means that no scaling factor has to be applied.

On the other hand, display-type images (such as those generated by the environments `table`, `figure`, `equation`, or `minipage`) are controlled by the variable `FIGURE_SCALE_FACTOR`. The default value for both of these variables is 1.6.

Moreover, several parameters can affect the conversion of a single "figure" with the `\htmlimage` command:

`\htmlimage{`*options*`}`

This command can be used inside every environment that is normally translated into a display-type image. To be effective the `\htmlimage` command (and its options) must be placed inside the environment on which it has to operate. The argument *options* specifies how the image in question will be handled; it contains a comma-separated list of keyword-value pairs.

`scale=`*scale-factor*
: the scale factor for the final image;

`external`
: the image does not have to be included in the document, but a hyperlink whose URL points to it has to be inserted to access it;

`thumbnail=`*reduction-factor*
: a small inlined image will be generated and placed in the caption; its size depends on the specification *reduction-factor* that downsizes the image by that amount. Note that this option implies `external`.

`map=`*image-map-URL*
: turns the inlined image into an active image map[10].

An example is the following LaTeX code:

```
\begin{figure}
  \htmlimage{thumbnail=0.25}
  \includegraphics{myfig.eps}
  \caption{description of my figure}
  \label{fig-myfig}
\end{figure}
```

`\htmlimage` can also be used to locally cancel out the effect of the `FIGURE_SCALE_FACTOR` configuration variable. For instance, if one does not want to resize a given image, then the command `htmlimage{scale=0}` will do the trick.

3.9 Internationalization

A special variable, `TITLES_LANGUAGE`, in the initialization or configuration files determines the language in which some section titles will appear. For example, by setting it to

```
$TITLES_LANGUAGE = "french";
```

`LaTeX2HTML` will produce "Table des matières" instead of "Table of Contents".

Currently, "french" and "english" are the only languages supported. It is not difficult, however, to add support for another language in the initialization file `latex2html.config`. As an example, below is shown the entry for French titles:

```
sub french_titles {
 $toc_title = "Table des matières";
 $lof_title = "Liste des figures";
 $lot_title = "Liste des tableaux";
 $idx_title = "Index";
 $bib_title = "Références";
 $info_title =
      "À propos de ce document...";
```

[10] More information on active image maps is at the URL `http://wintermute.ncsa.uiuc.edu:8080/map-tutorial/image-maps.html`.

```
sub navigation_panel {

    #  Start with a horizontal rule (3-d dividing line)
    "<HR> ".

    # Now add few buttons with a space between them
    "$NEXT $UP $PREVIOUS $CONTENTS $INDEX $CUSTOM_BUTTONS" .

    "<BR>\n" . # Line break

    # If ``next'' section exists, add its title to the navigation panel
    ($NEXT_TITLE ? "<B> Next:</B> $NEXT_TITLE\n" : undef) .

    # Similarly with the ``up'' title ...
    ($UP_TITLE ? "<B>Up:</B> $UP_TITLE\n" : undef) .

    # ... and the ``previous'' title
    ($PREVIOUS_TITLE ? "<B> Previous:</B> $PREVIOUS_TITLE\n" : undef) .

    #  Horizontal rule (3-d dividing line) and new paragraph
    "<HR> <P>\n"
}
```

Figure 9: Example definition of a navigation panel. (Note that "." is the perl string concatenation operator and "#" signifies a comment).

}

In order to provide full support for another language you may also want to replace the navigation buttons which come with **LaTeX2HTML** (which are by default in English) with your own. As long as the new buttons have the same filenames as the old ones there should not be a problem.

3.10 Known Problems

Users of **LaTeX2HTML** should take note of the following shortcomings of the translator.

- *Unrecognized commands and environments.*
  Unrecognized *commands* are ignored and any arguments are left in the text. Unrecognized *environments* are passed to LaTeX and the result is included in the document as one or more inlined images. Users can take care of this by providing information to **LaTeX2HTML** on how to handle such cases, either by deciding to ignore them (see Section 3.5.2 on page 6), or by defining a `perl` procedure (see Appendix C on page 31).
- *Cross-references.*
  References in environments that are passed to LaTeX for processing (such as `\cite` or `\ref` commands), are not processed correctly. On the other hand, `\label` commands are handled satisfactorily.
- *Order-sensitive commands.*
  Commands affecting global parameters during the translation that are sensitive to the order in which they are processed may cuase problems. In particular, counter manipulation with commands such as `\newcounter`, `\setcounter`, `\stepcounter` should be watched.
- *Index.*
  LaTeX2HTML generates its own index by saving the arguments of the `\index` command. The contents of the `\theindex` environment are ignored.
- *New definitions.*
  New definitions (with the `\def`, `\newcommand`, `\newenvironment`, `\newtheorem` commands) will not work as expected if they are defined more than once. Indeed, only the last definition will be used throughout the document.
- *Scope of declarations and environments.*
  LaTeX2HTML processes sections as independent units. Thus, when the scope of a declaration or environment crosses section boundaries, the output may not be as expected.

4 HTML3 Extensions to LaTeX2HTML

4.1 The math2html Program

The simple notation for even complex mathematics and the diversity of the symbols and characters sets available makes LaTeX the typesetting system of choice in many of the scientific fields. Tens of thousands of articles, theses, and reports have been written in LaTeX and most publishing houses that deal with scientific papers use LaTeX for handling, storing and archiving their documents. Therefore it is to be expected that all these parties wish to protect their investment and prefer not to have to recode their mathematics formulae for hypertext purposes only.

The `LaTeX2HTML` translator solves the problem of presenting mathematics in HTML by converting each mathematical sentence into a bitmap image. Although simple and straighforward, this approach seems a little unreasonable in general, since in many cases an article of a few pages can generate many hundreds of bitmap images, which have to be stored with the document, kept up to date, and transmitted with the document over the Internet, thus wasting an enormous amount of bandwidth. Therefore, a clear need for a translator from LaTeX mathematics into HTML3's primitive mathematics was considered an important goal. Thanks to the increased displaying capabilities of HTML3 complyable browsers, most inline mathematics and a fair proportion of display equations can be translated into HTML3 source code and hence transmitted in textual format together with the rest of the document, doing away with well over 90% of the images that are created in the HTML2 case where only bitmap images are generated. In addition, mathematics text can be searched for keywords as the rest of the document, thus increasing the value of the HTML document.

The `math2html` program has been interfaced to the `LaTeX2HTML` program via a new option `-html3`. When this option is specified, `LaTeX2HTML` will first pass the LaTeX input source code through the `math2html` translator. In this case, native HTML3 code will be generated for mathematics and tables when `math2html` can handle the input. In case `math2html` cannot parse the given LaTeX input, it gives an error message and `LaTeX2HTML` creates an image as usual.

At CERN we have translated thousands of pages of manuals and hundreds of physics articles. We found that `math2html` successfully translates on average 95% of all mathematics present in the input files, thus reducing by a substantial amount the number of generated bitmap images.

4.1.1 A few Examples

The HTML3 extensions translate quite a large fraction of not-too-complex LaTeX math constructs (for as far as they can be handled by the HTML3 DTD, of course).

A first explicit example is the code representing the differential cross-section of δ-ray production. The original LaTeX code and its result as typeset by LaTeX are shown in parts (a) and (b) of Figure 10, while the result of the translation by `math2html` of the LaTeX source in (a) into HTML3 is shown in (c), yielding the output with the `arena` browser shown in (d). Part of the tree constructed by `math2html` when parsing this LaTeX input is shown in Figure 17 on page 40.

Multi-line mathematical constructs, such as arrays (`array` and `eqnarray` environments), are also handled without too many problems, and the present limits of the translation are due more to shortcomings of the (only) HTML3 browser `arena` (which is, after all, merely a beta-test version), than to intrinsic limitations in the approach. In Figure 11 we show the LaTeX source and result as seen with `arena` of two multi-line environments.

4.1.2 Writing Convertible LaTeX

By following the rules below, one can expect the `LaTeX2HTML` translator enhanced with `math2html` to produce good output in terms of a low number of bitmap images.

- Do not write the base of a superscript or a subscript outside the mathematics markup, i.e., `a$^2$` is not converted correctly but creates a bitmap image. The correct way is to write it `$a^2$` or `$\mathrm{a}^$` depending, on whether or not one wants the letter "a" in math italic or in a roman font. When you leave the base outside of the math markup (the $ signs) the text between the mathematics delimiters is passed to the `math2html` translator and the latter does not know where to place the mathematics start (`<math>`) tag.
- Do not write nested `array`/`tabular` environments. The `math2html` translator cannot create an HTML3 counterpart for that markup since the HTML3 table model does not allow nested tables. Keeping the tables simple (not nested, for example) will improve their reusability.

4.2 Tables to HTML3 Conversion

Hennecke recently developed some code for treating LaTeX's `tabular` environment with `LaTeX2HTML` by translating it into HTML3-compliant tables. His patches[11] allow `LaTeX2HTML` to translate most LaTeX tables reasonably well. There are a few things it cannot do, but mainly because HTML tables are

[11] Available from the URL `ftp://ftp.crc.ricoh.com/pub/www/l2h/tables.tar.gz`. The author Marcus E. Hennecke can be reached by email at `marcush@crc.ricoh.com`.

(a) LaTeX source that has to be translated:

```
\frac{d\sigma}{d\epsilon}=\frac{2\pi Z r_0^2m}{\beta^2(E-m)}\left[\frac{(\gamma-1)^2}
{\gamma^2}+\frac{1}{\epsilon}\left(\frac{1}{\epsilon}-\frac{2\gamma-1}{\gamma^2}\right)
+\frac{1}{1-\epsilon}\left(\frac{1}{1-\epsilon}\frac{2\gamma-1}{\gamma^2}\right)\right]
```

(b) Result of the above source as typeset with LaTeX:

$$\frac{d\sigma}{d\epsilon}=\frac{2\pi Z r_0^2m}{\beta^2(E-m)}\left[\frac{(\gamma-1)^2}{\gamma^2}+\frac{1}{\epsilon}\left(\frac{1}{\epsilon}-\frac{2\gamma-1}{\gamma^2}\right)+\frac{1}{1-\epsilon}\left(\frac{1}{1-\epsilon}\frac{2\gamma-1}{\gamma^2}\right)\right]$$

(c) Result of the translation of the code in (a) into HTML3:

```
<math><box>d&sigma;<over>d&epsi;</box>=<box>2&pi;Zr<sub>0</sub><sup>2</sup>m<over>&beta;
<sup>2</sup>(E-m)</box>[<box>(&gamma;-1)<sup>2</sup><over>&gamma;<sup>2</sup></box>+
<box>1<over>&epsi;</box>(<box>1<over>&epsi;</box>-<box>2&gamma;-1<over>&gamma;<sup>2
</sup></box>)+<box>1<over>1-&epsi;</box>(<box>1<over>1-&epsi;</box>
<box>2&gamma;-1<over>&gamma;<sup>2</sup></box>)]</math>
```

(d) Result of viewing of the HTML3 code of (c) with the **arena** browser:

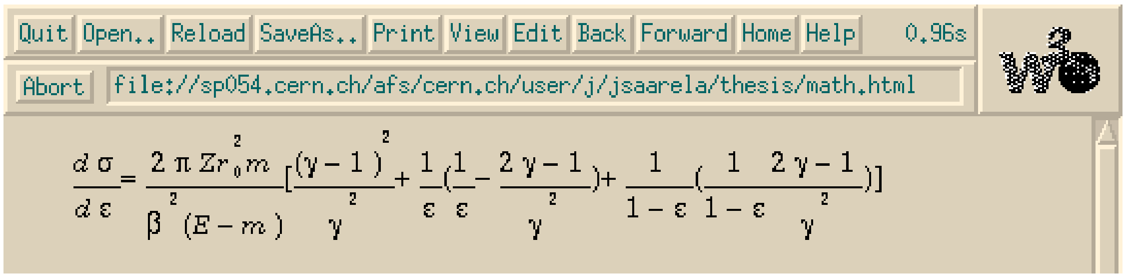


Figure 10: Example of transforming LaTeX code to HTML3 with `math2html`

```
\begin{eqnarray}
a & = & \sin \alpha_2      \\
b & = & \cos \omega_3      \\
\Gamma & = & \Phi + \Theta\\
\end{eqnarray}

\[
 \begin{array}{cccccc}
 a_{11}                    \\
 a_{21} & a_{22}           \\
 a_{31} & a_{32} & a_{33}  \\
 a_{41} & a_{42} & a_{43} &
   a_{44}                   \\
 a_{51} & a_{52} & a_{53} &
   a_{54} & a_{55}         \\
 a_{61} & a_{62} & a_{63} &
   a_{64} & a_{65} & a_{66}\\
 \end{array}
\]
```

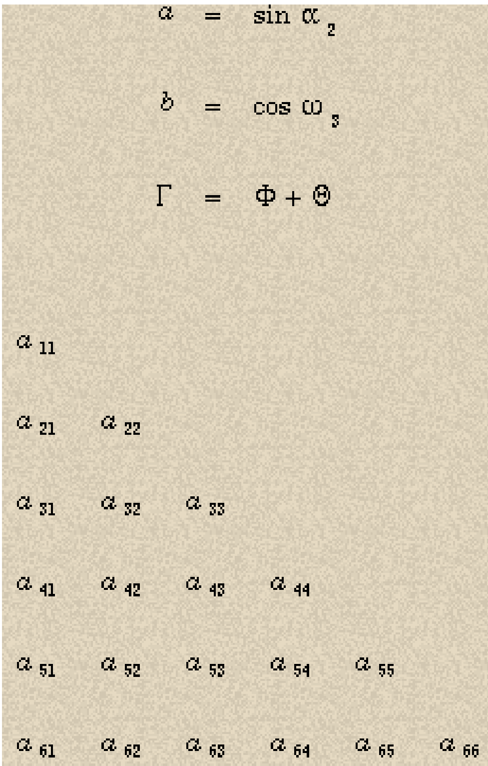


Figure 11: How `math2html` translates LaTeX multi-line mathematics into HTML3

not quite as powerful as LaTeX tables. Most importantly, HTML tables are quite limited when it comes to borders, since they are not nearly as flexible in specifying borders as LaTeX tables. In his implementation, when a LaTeX table has a border anywhere, the resulting HTML table will have borders around all cells. LaTeX commands inside cells are treated as they should and declarations are limited in scope to the cell in which they appear (just as in LaTeX itself).

His additions can be placed in the `LaTeX2HTML` `perl` code itself, or in the customization files. In any case to leave open the possibility of generating tables with and without this feature turned on, a new command line option `-html_level` can be used to specify the level of HTML to be produced.

4.2.1 Examples

First, we look at a simple table with different alignments:

```
\begin{tabular}{|l|c|r|}  \hline
first column   &
second column  &
third column              \\\hline
111 111  & 22 22 22 &
3 3 3 3                   \\\hline
\end{tabular}
```

The result is seen at the top of Figure 12.

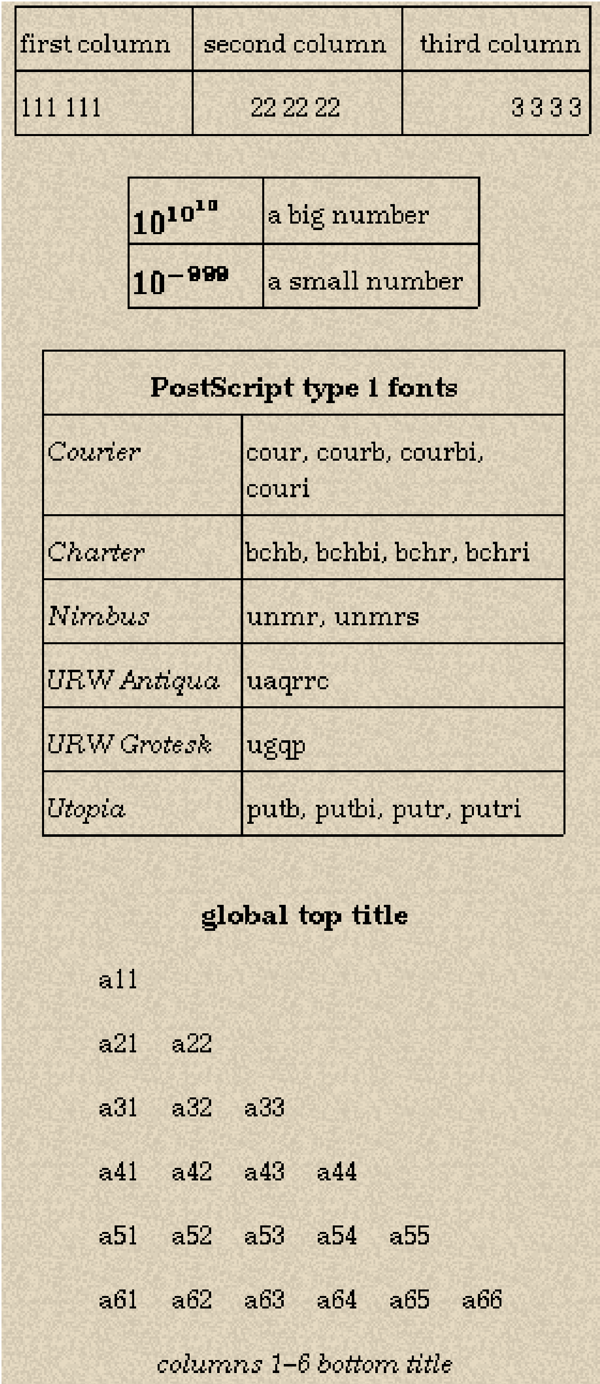
Math can also be handled (in this case it will be translated into images). With a little bit if "handwork" it could be translated into native HTML3:

```
\begin{tabular}{|ll|}           \hline
$10^{10^{10}}$& a big number  \\\hline
$10^{-999}$   & a small number\\\hline
\end{tabular}
```

The result is seen in the second table from the top in Figure 12.

Modifications to text inside cells remain limited to that cell (as it should). In the present version only *one* `\multicolumn` command is recognized (when more than one such command is encountered inside a row, only the first one is taken into account):

```
\begin{tabular}{|ll|}
\multicolumn{2}{c}{\bf PostScript
                   type 1 fonts}  \\
\em Courier     &
  cour, courb, courbi, couri      \\
\em Charter     &
  bchb, bchbi, bchr, bchri        \\
\em Nimbus      &
  unmr, unmrs                     \\
\em URW Antiqua &
  uaqrrc                          \\
```



first column	second column	third column
111 111	22 22 22	3 3 3 3

$10^{10^{10}}$	a big number
10^{-999}	a small number

PostScript type 1 fonts	
Courier	cour, courb, courbi, couri
Charter	bchb, bchbi, bchr, bchri
Nimbus	unmr, unmrs
URW Antiqua	uaqrrc
URW Grotesk	ugqp
Utopia	putb, putbi, putr, putri

global top title

a11					
a21	a22				
a31	a32	a33			
a41	a42	a43	a44		
a51	a52	a53	a54	a55	
a61	a62	a63	a64	a65	a66

columns 1–6 bottom title

Figure 12: Four examples of `tabular` environments translated automatically to HTML3 as viewed with the `arena` browser

```
\em URW Grotesk &
  ugqp                              \\
\em Utopia      &
  putb, putbi, putr, putri
\end{tabular}
```

The result is seen in the third table from the top of Figure 12. Note that, even though only vertical rules were specified in the `tabular`'s preamble, rules are drawn everywhere. This is because the `BORDER` attribute of the `<TABLE>` tag in HTML3 has only one value, i.e., borders are present or absent everywhere.

Our final example has again a few `\multicolumn` commands, but also shows that non-specified cells are treated gracefully (this can be compared to the example in Section 4.1.1, where a similar table was built as an array inside math mode):

```
\begin{tabular}{cccccc}
\multicolumn{6}{c}{\bf global top title}\\
a11                          \\
a21 & a22                    \\
a31 & a32 & a33              \\
a41 & a42 & a43 & a44        \\
a51 & a52 & a53 & a54 & a55    \\
a61 & a62 & a63 & a64 & a65 & a66    \\
\multicolumn{6}{c}%
           {\em columns 1-6 bottom title}
\end{tabular}
```

The result is seen as the bottom table of Figure 12. As no vertical nor horizontal rules were specified in the input, the resulting table has no borders.

5 Caml based LaTeX to HTML Translation

Xavier Leroy (INRIA, France) developed a LaTeX to HTML translator based on the Caml language[12].

What was needed was to translate a 200-page technical document (the reference manual and user's documentation for their implementation of the `Caml Light` programming language). This manual was written in LaTeX and contained some rather non-standard environments and macros written directly in TeX. Parts of the document were automatically generated: syntactic definitions (typeset from BNF descriptions) and descriptions of library functions (extracted from commented source code).

5.1 Why not just use LaTeX2HTML?

When `LaTeX2HTML` finds a LaTeX construct that it does not know how to translate into HTML, it simply turns it into a bitmap. This approach was considered inappropriate by Leroy *et al.*, since

- information transformed into a bitmap is not searchable;
- bitmaps cannot easily be integrated into Macintosh or Windows online documentation systems;
- bitmaps are generally hard to read, since their resolution usually does not match that of the HTML viewer;
- as bitmaps can be quite large, transmission times increase and network bandwidth suffers.

In order to minimize the generation of bitmaps and to allow the production of a better quality HTML source, the information in the LaTeX source was tagged by LaTeX macros to explicitly show its semantics meaning. Special care was taken to avoid inline mathematics constructs, since they result in bitmap images, for example, `\var{x}` was preferred to its typographic equivalent `$x$` (denoting a metavariable), and `\nth{v}{n}` was used to mean the n-th component of `v`, rather than writing `$v_n$`. The same technique was also used to eliminate "low-level" typesetting constructs and environments such as `center` and `tabular`.

When typesetting the document with LaTeX these new commands were simple translated into the needed typographic representation, but during the translation into HTML they were explicitly recognized and individually translated into a form that corresponds with the possibilities of HTML. For instance, `\nth{v}{n}` would become something like `<i>v(n)</i>`, showing `<i>v(n)</i>`.

The programs that automatically generated BNF or program fraction for inclusion in the LaTeX source were adapted so that its contents could now also be included without problems in the HTML source by "hiding" the generated material inside a `rawhtml` environment.

Finally, the few places where more complex mathematical constructs were needed were hand-translated into a form acceptable to HTML and stored inside a `rawhtml` environment, leaving the original mathematics expressions inside a `latexonly` environment. Thus both the LaTeX and HTML views of the document were optimized. Although in principle such an approach can lead to synchronization problems between the LaTeX and HTML parts of the information, it was found that, due to the care that was taken in using the generic markup approach outlined above, only about 0.2% of the source had to be manually translated.

Although Leroy and his collaborators originally planned to use `LaTeX2HTML` for translating their document into HTML, they found that some commands

[12] More information can be found at the URL `http://pauillac.inria.fr/~xleroy/w4g.html`.

(especially those using verbatim-like constructs, most notably the `alltt` environment) cannot be defined in `perl` in an easy way using the interface of init files described earlier. Therefore modifications have to be made inside the body of the `LaTeX2HTML` program itself, and this is very complicated since the inner workings of `LaTeX2HTML` are undocumented and scarcely commented, so that the `perl` code is not always clear to follow. Also, the memory requirements of `LaTeX2HTML` (especially the pre-1995 versions, when the tests were done) can be huge, exhausting all the memory available on the machine and causing the program to crash (this should no longer be a problem with the current version of `LaTeX2HTML` if the LaTeX source in divided into a set of smaller files). They therefore decided to write their own LaTeX-to-HTML translator for the extended subset of LaTeX commands they used.

This translator works in two main stages:

- The translator first reads the whole LaTeX document and outputs one large HTML document. It is written in `Caml Light` and uses the lexical analyzer generator `camllex` (the `Caml` equivalent of `lex` for C) heavily. Note that `Caml` is a modern, type-safe high-level programming language with good memory management, so that the translator has negligible memory requirements, runs quickly, is easy to extend, and took little time to develop.
- The output of the translator is then split into smaller parts (for instance at the `<H1>` or `<H2>` heading levels), and these parts are linked together using "Next" and "Previous" buttons. This linking is performed by two simple `perl` scripts.

In order to get a feeling of the result of the translation, one can look at a randomly chosen page from the manual that was converted. Figure 13 shows the result of LaTeX (viewed with `xdvi`) and Figure 14 is the result of the HTML conversion, as shown by `Mosaic`[13]. Appendix E takes a closer look at how the `Caml` system translates LaTeX commands into HTML.

5.2 Discussion

Based on their experience with writing and using their translator Leroy and collaborators draw the conclusions summarized in the next sections.

[13] The complete manual—HTML and dvi files—are at the URLs `http://pauillac.inria.fr/~xleroy/man-caml/` and `ftp://ftp.inria.fr/lang/caml-light/Release7beta/cl7refman.dvi.gz`, respectively.

5.2.1 About the HTML Language

Despite its apparent simplicity, the HTML language is almost rich enough to format TeX-intensive technical documentation. The sole features that were badly missed were tables, subscripts, and superscripts. This is much less true today, since HTML3 already contains an interesting table model, and allows for super and subscripts. Moreover, the latest versions of `Mosaic` and `Netscape` support these functions.

5.2.2 About HTML Viewers

The quality of the typesetting performed by popular HTML viewers (such as `Mosaic` and `Netscape`) is very often insufficient. It seems especially difficult to ensure font consistency throughout a document.

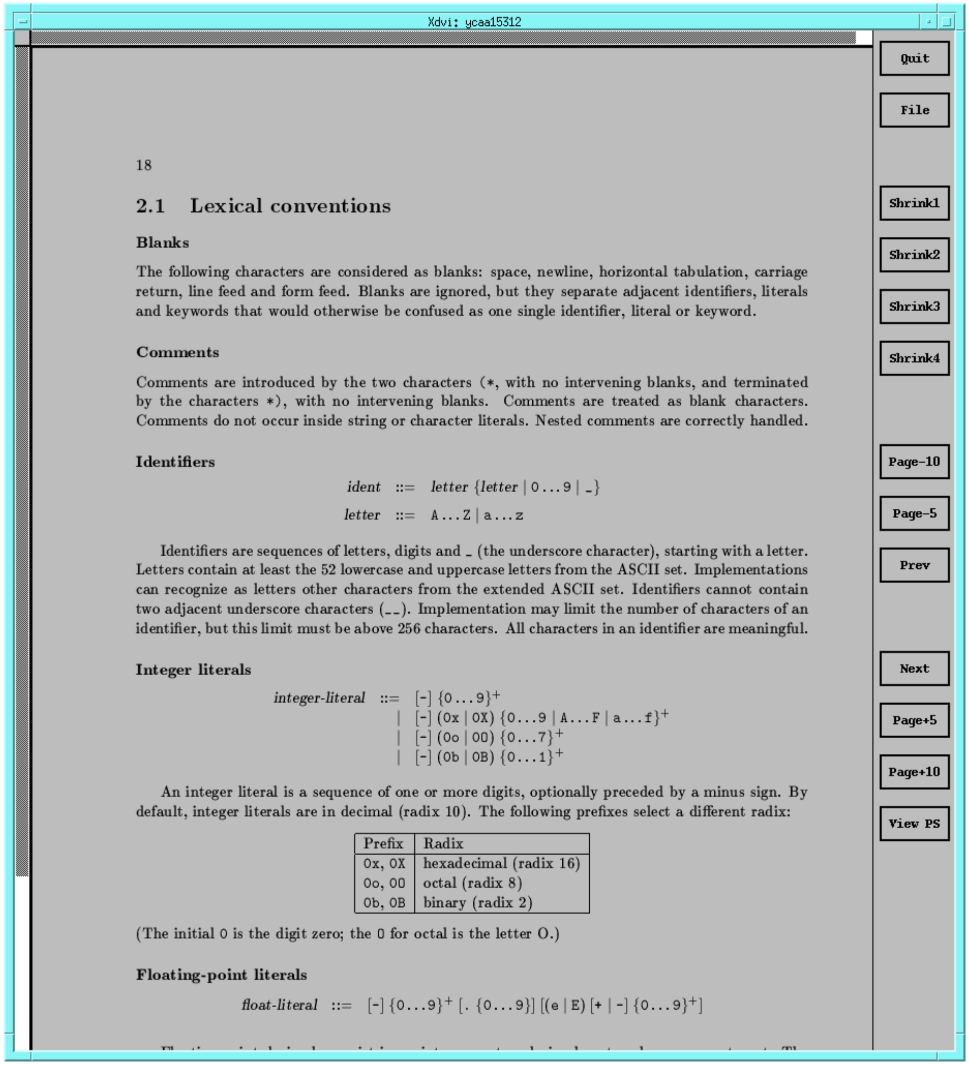
The difficulty in finding good translators and adequate viewers probably has to do with the immaturity of the field. Leroy et. al. are convinced that the widespread use of `perl` for programming translation tools is partly responsible for this situation. They state that `perl` is inherently not suited to the parsing and transformation of structured languages, such as LaTeX and HTML, and go on to say that languages with high-level parsing capabilities, real data structures and clean semantics are much more suited for these tasks.

They also ask the question: what is the best markup language for preparing documentation so that it can be nicely printed but also easily transformed into HTML for publishing on the Web? They accept that, LaTeX presently being the *de facto* standard markup language used in computing and other science fields, it will be difficult in the short term to propose a solution other than to invest more effort in developing cleverer and more comprehensive LaTeX-to-HTML translators.

6 Converting HTML to LaTeX

Although utilities for obtaining PostScript representations from HTML files are readily available, either using HTML browsers, such as `Mosaic`, that offer PostScript as one of their output formats, or directly (for example, `htps`[14]) the visual layout of these documents is often appalling, and the structuring of the information has been made almost completely invisible. Often one would like to obtain a nicely typeset document that presents the information marked up in HTML in a structured way, with all document elements clearly identifiable. A translation into LaTeX allows one to combine the power of the TeX typesetting engine while at the same time exploiting the

[14] More information is at the URL `http://info.cern.ch/hypertext/WWW/Tools/htps.html`.



18

2.1 Lexical conventions

Blanks

The following characters are considered as blanks: space, newline, horizontal tabulation, carriage return, line feed and form feed. Blanks are ignored, but they separate adjacent identifiers, literals and keywords that would otherwise be confused as one single identifier, literal or keyword.

Comments

Comments are introduced by the two characters `(*`, with no intervening blanks, and terminated by the characters `*)`, with no intervening blanks. Comments are treated as blank characters. Comments do not occur inside string or character literals. Nested comments are correctly handled.

Identifiers

ident ::= *letter* {*letter* | `0...9` | `_`}

letter ::= `A...Z` | `a...z`

Identifiers are sequences of letters, digits and `_` (the underscore character), starting with a letter. Letters contain at least the 52 lowercase and uppercase letters from the ASCII set. Implementations can recognize as letters other characters from the extended ASCII set. Identifiers cannot contain two adjacent underscore characters (`__`). Implementation may limit the number of characters of an identifier, but this limit must be above 256 characters. All characters in an identifier are meaningful.

Integer literals

integer-literal ::= [`-`] {`0...9`}$^+$
| [`-`] (`0x` | `0X`) {`0...9` | `A...F` | `a...f`}$^+$
| [`-`] (`0o` | `0O`) {`0...7`}$^+$
| [`-`] (`0b` | `0B`) {`0...1`}$^+$

An integer literal is a sequence of one or more digits, optionally preceded by a minus sign. By default, integer literals are in decimal (radix 10). The following prefixes select a different radix:

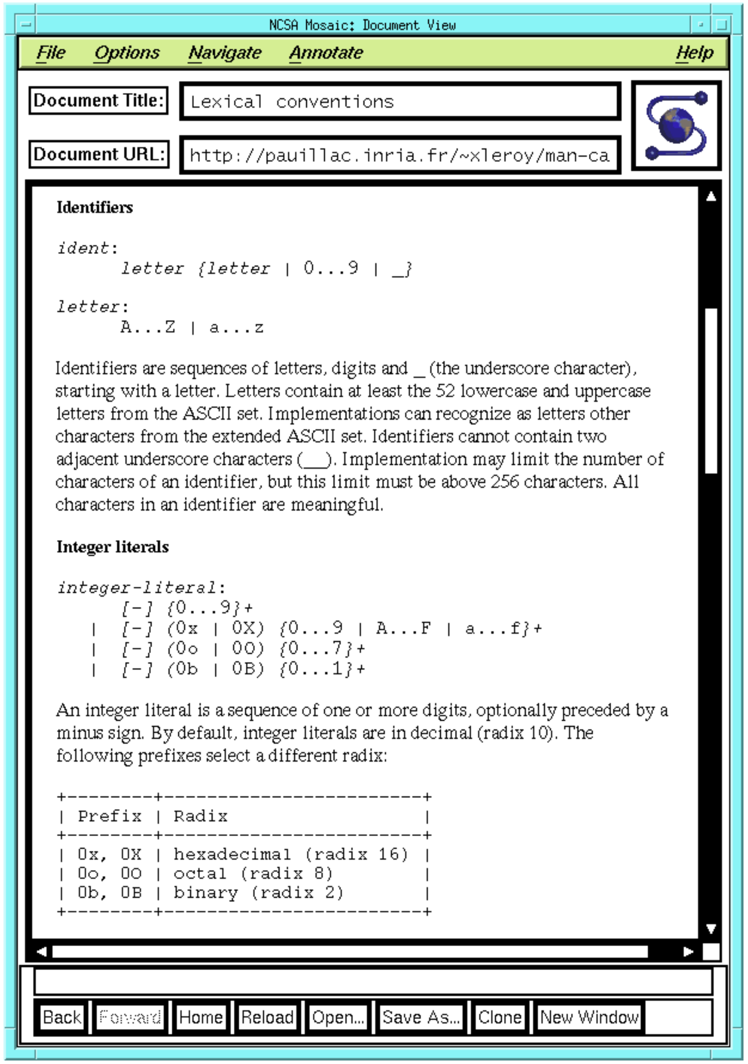
Prefix	Radix
`0x`, `0X`	hexadecimal (radix 16)
`0o`, `0O`	octal (radix 8)
`0b`, `0B`	binary (radix 2)

(The initial `0` is the digit zero; the `O` for octal is the letter `O`.)

Floating-point literals

float-literal ::= [`-`] {`0...9`}$^+$ [`.` {`0...9`}] [(`e` | `E`) [`+` | `-`] {`0...9`}$^+$]

Figure 13: Example page of `Caml` manual (LaTeX viewed with `xdvi`)



Identifiers

```
ident:
      letter {letter | 0...9 | _}

letter:
      A...Z | a...z
```

Identifiers are sequences of letters, digits and _ (the underscore character), starting with a letter. Letters contain at least the 52 lowercase and uppercase letters from the ASCII set. Implementations can recognize as letters other characters from the extended ASCII set. Identifiers cannot contain two adjacent underscore characters (__). Implementation may limit the number of characters of an identifier, but this limit must be above 256 characters. All characters in an identifier are meaningful.

Integer literals

```
integer-literal:
      [-] {0...9}+
   |  [-] (0x | 0X) {0...9 | A...F | a...f}+
   |  [-] (0o | 0O) {0...7}+
   |  [-] (0b | 0B) {0...1}+
```

An integer literal is a sequence of one or more digits, optionally preceded by a minus sign. By default, integer literals are in decimal (radix 10). The following prefixes select a different radix:

```
+--------+-----------------------+
| Prefix | Radix                 |
+--------+-----------------------+
| 0x, 0X | hexadecimal (radix 16)|
| 0o, 0O | octal (radix 8)       |
| 0b, 0B | binary (radix 2)      |
+--------+-----------------------+
```

Figure 14: Example page of `Caml` manual (HTML converted with `Caml` based translator viewed with `Mosaic`)

structural similarities between HTML and LaTeX as explained in Section 1 and Table 1.

A first program **HTML2LaTeX** translates a large fraction of the HTML commands into LaTeX, while `SGML2TeX` takes a more general approach and allows the transformation of an arbitrary SGML source into LaTeX.

6.1 HTML2LaTeX, an HTML-to-LaTeX Converter

HTML2LaTeX is a C-program written by Nathan Torkington (New Zealand). Basically, the HTML parser of the NCSA `Mosaic` HTML browser is used for the translation. The calling sequence of the program is:

`html2latex` [*options*] [*filenames*]

For each input file specified, **HTML2LaTeX** transforms the HTML markup in the source into the equivalent LaTeX markup. When no filenames are specified, **HTML2LaTeX** will display a short description of how to use the program. If *filenames* is equal to `-`, then the input text is read on standard input `stdin`. For each input file an output file with the same name, but with the extension `.tex` instead of `.html` is generated.

6.1.1 Options

HTML2LaTeX has a number of options that modify its way of operation. The more important are:

- `-n` number the sections;
- `-p` start a new page after the titlepage (if present) or the table of contents (if present);
- `-c` generate a table of contents;
- `-s` write output information on `stdout`;
- `-t` *Title* generate a titlepage containing the title *Title*;
- `-a` *Author* generate a titlepage containing the author(s) *Author*;
- `-h` *Start-Text* introduce the text *Start-Text* immediately following the command `\begin{document}`;
- `-h` *End-Text* introduce the text *End-Text* immediately preceding the command `\end{document}`;
- `-o` *options* specify the options *options* on the `\documentclass` command.

6.1.2 Examples

Let us consider the following command:

```
html2latex -n - < file.html | more
```

In this case the file `file.html` is transformed into LaTeX and the result is shown on the screen. The option `-n` makes sure no section numbers are generated.

A more complex example is shown below:

```
html2latex -t 'HTML for Pedestrians' \
           -a 'First Last' -p          \
           -c -o'[12pt,twoside]{article}'\
           my-article.html
```

In this case file `my-article.html` will be read, and the output written to the file `my-article.tex`. A titlepage (using the text "HTML for Pedestrians" as title and "First Last" as author) will be output on a separate page (option `-p`). A Table of contents (option `-c`) followed by a new page (option `-p` again) will also be generated. Sections will be numbered (default behavior). The LaTeX document will be typeset at 12 point using the document option `twoside`, to allow *two-sided* printing.

6.1.3 Limitations

The present version of **HTML2LaTeX** recognizes the following HTML tags: `<TITLE>`, `<H1>` to `<H6>`, for lists `<OL>`,`<UL>`, `<DT>`, `<DD>` and `<LI>`, plus the presentation tags `<B>`, `<I>`, `<U>`, `<EM>`, `<CODE>`, `<SAMP>`, `<STRONG>`, `<KBD>`, `<VAR>`, `<DFN>`, `<CITE>`, and `<LISTING>`. Of the entities only `&`, `<` and `>` are handled correctly. The content fields of the tags `<ADDRESS>`, `<DIR>` and `<MENU>` are not handled correctly. Moreover, the **COMPACT** attribute of the `<DL>` tag is not honored and the text of the `<TITLE>` tag is ignored. Even worse, the body of the `<PRE>` elements are completely ignored.

Note that the complete HTML file is read into memory; this can lead to problems when handling large files on machines with limited memory capabilities.

6.2 SGML2TeX, a General-Purpose SGML to LaTeX Converter

SGML2TeX[15] is a program written by Peter Flynn (Cork, Ireland) that translates SGML tags into TeX instructions. At present the system is only implemented in PCL[16] running runs under MS-DOS on a PC but the author has plans to rewrite it in a more portable programming language.

SGML2TeX does not verify the SGML source for correctness but accepts all SGML documents marked up using the reference concrete syntax. It is up to the user to define a LaTeX equivalent for each of the

[15] For more information see the URL `http://info.cern.ch/hypertext/WWW/Tools/SGML2TeX.html`.

[16] PCL stands for Personal Computer Language, an interpreted language for DOS on the *86 chips. It is a very fast prototyping tool, not a production language since it cannot link executable images.

SGML document elements, their attributes, and the entities used in the source. A configuration file that contains a set of such predefined correspondences for certain elements, attributes, or entities, can be read by `SGML2TeX`, thus substantially alleviating the task of the user, who will only have to provide the missing definitions. By default, i.e., in the absence of an explicit translation, SGML elements are translated in a form acceptable to LaTeX by adopting the following conventions:

- start tags get the prefix `\start` and end tags the prefix `\finish` followed by the tagname in upper case, followed by a pair of braces (`{}`). This pair of braces corresponds to a *do-nothing* definition for each of the tags thus handled;
- SGML entities of the form `&ent;` are translated into `\ent{}` and written into the output file;
- attributes are handled in the same way, but their value is specified between curly braces like a LaTeX argument.

Acknowledgments

We sincerely thank Nelson Beebe (Utah University, `beebe@math.utah.edu`) for e-mail discussions and for his detailed comments of the compuscript. His many suggestions improvements have without doubt substantially increased the readability and quality of the article. We also want to acknowledge Steven Kennedy (CERN) for proofreading the article.

References

Goossens, Michel and Eric van Herwijnen. The elementary Particle Entity Notation (PEN) scheme. *TUGboat*, 13(1), pages 201–207, July 1992.

Goossens, Michel, Frank Mittelbach and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, 1994.

Lamport, Leslie. *LaTeX, User's Guide and Reference Manual (2nd Edition)*. Addison-Wesley, Reading, 1994.

Ousterhout, John K. *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, 1994.

Rumbaugh et al. *Object-Oriented Modeling and Design*. Prentice Hall, Inc., Englewood Cliffs, N.J., 1991.

Schwartz, Randal L. *Learning Pearl*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1993.

Joachim Schrod. Towards interactivity for TeX. *TUGboat*, 15(3), pages 309-317, September 1994.

Till, David. *Reach Yourself* PERL *in 21 Days. Sams* PUBLISHING, Indianapolis, 1995.

Wall, Larry and Randal L. Schwartz. *Programming Pearl*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1991.

A complete and up-to-date list of titles of books on HTML and `perl` is maintained by Nelson Beebe (Utah University, `beebe@math.utah.edu`) and can be found in his BibTeX databases `sgml.bib` and `unix.bib`, respectively, in the directory with URL address `ftp://ftp.math.utah.edu/pub/tex/bib/`
.

Appendices

Appendices A and B present a few practical details that we found particularly relevant when installing or troubleshooting `LaTeX2HTML`[17]. Appendix C then provides some more information about the internal workings of the `LaTeX2HTML` program and how it can be extended by writing `perl` procedures. Finally, Appendix D contains technical information about the `math2html` extension to `LaTeX2HTML`, while Appendix E takes a closer look at Leroy's Caml-based LaTeX-to-HTML translator.

A `LaTeX2HTML`—Installation

A.1 Requirements to run `LaTeX2HTML`

`LaTeX2HTML` uses several publicly available tools that can be readily found on most computer platforms, namely:

- LaTeX (of course).
- `perl` (version 4 from patch level 36 onward, or, even better, version 5).
- `DBM` or `NDBM`, the Unix DataBase Management system.
- `dvips` (version 5.516 or later) or `dvipsk`.
- `gs` (Ghostscript version 2.6.1 or later).
- The `pbmplus` or better still the `netpbm` libraries; some of the filters in those libraries are used during the postscript to `GIF` conversion.
- For making transparent inlined images one needs `giftrans.c`[18] by A. Ley together with `pbmplus`. Alternatively, `netpbm` will do the trick.

To reduce the memory requirements of the translation, `LaTeX2HTML` spawns off separate Unix processes to deal with each of the `input`'ed or `include`'d files. As each process terminates, all the space that it used is reclaimed. Asynchronous communication between processes takes place using the Unix DataBase Management system (`DBM` or `NDBM`) which should be present. To take advantage of these changes it is necessary to split the source text into multiple files that can be assembled using LaTeX's `\input` or `\include` commands.

When `gs` or the `pbmplus` (`netpbm`) library are not available, one can still generate HTML output, but without images (using the `-no_images` option). Also, do not forget to include the html package with the `\usepackage` command if you want to include any of the hypertext extension commands described in Section 3.6.

A.2 Installing `LaTeX2HTML`

Those intending to install `LaTeX2HTML` on their system should read the manual in detail. Below we describe only the main steps.

- *Specify where **perl** is on the system.*
  In the files `latex2html`, `texexpand`, `pstogif`, and `install-test` modify the first line saying where `perl` is on your system.
- *Specify where the external utilities are on the system.*
  In the file `latex2html.config` give the correct pathnames for some directories (the `latex2html` directory and the `pbmplus` or `netpbm` library) and some executables (`latex`, `dvips`, `gs`).
  Note that `LaTeX2HTML` can be run even if one does not have some of these utilities.

One can also include the following supplementary customization:

- *Setting up different initialization files.*
  One can customize on a "per user" basis the initialization file. To this effect one should copy the file `dot.latex2html-init` into the home directory of any user who wants it, modify it according to the user's preferences and rename it to `.latex2html-init`.

  At runtime both `latex2html.config` and `$HOME/.latex2html-init` files will be loaded, but the latter will take precedence. Moreover, one can also set up a "per directory" initialization file by copying

[17] These sections are adapted from the LaTeX2HTML manual that is available at the URL `http://cbl.leeds.ac.uk/nikos/tex2html/doc/latex2html/latex2html.html`.

[18] `ftp://ftp.rz.uni-karlsruhe.de/pub/net/www/tools/giftrans.c`.

a version of the initialization file `.latex2html-init` into each directory where it should be effective. In this case an initialization file `/X/Y/Z/.latex2html-init` takes precedence over all other initialization files if `/X/Y/Z` is the "current directory" when `LaTeX2HTML` is invoked.

- *Make local copies of the* ***LaTeX2HTML*** *icons.*
  The `icons` subdirectory should be copied to a place in the local WWW tree where it can be served by the local server. Therefore, in the file `latex2html.config` file the value of the variable `$ICONSERVER` should be changed accordingly.

B LaTeX2HTML—Troubleshooting

This section gives a few hints about how to solve problems with `LaTeX2HTML`. As a general rule, if one gets really lost, one can obtain a lot of information from the `perl` system by setting the environment variable `DEBUG` to 1. In particular it will point out missing files or utilities. Below we present some often occurring problems and propose a way how to deal with them.

LaTeX2HTML just stops without further warnings.

Check the package files that are included, since they might contain raw TeX commands, which cannot be handled. In this case start `LaTeX2HTML` with the option `-dont_include` followed by the name of the package file in question. Alternatively, one can add the name of the package file to the variable `DONT_INCLUDE` in the `HOME/.latex2html-init` file, or create one in the current directory containing the following lines:

```
$DONT_INCLUDE = "$DONT_INCLUDE:<name-of-package-file>";
1;  # This must be the last line
```

Similarly, when the LaTeX source file itself contains raw TeX command (`\let` is a common example!) `LaTeX2HTML` might also stop. Such commands should therefore be introduced inside a `latexonly` environment.

LaTeX2HTML gives an Out of memory message and crashes.

Divide the LaTeX source file into several files that can be input using `\include` commands. One can also try the `-no_images` option.

The "tilde" (~) does not show.

The easiest solution is to use the command `\~{}`. Alternatively it is possible to write something like:

```
\htmladdnormallink{mylink}
\begin{rawhtml}
  {http://host/~me/path/file.html}
\end{rawhtml}
```

Macro definitions do not work correctly.

As already mentioned, plain TeX definitions are be converted. But there can be problems even when using LaTeX definitions (with the `\newcommand` and `\newenvironment` commands) if such definitions make use of *sectioning* or *verbatim* commands, since these are handled in a special way by `LaTeX2HTML` and cannot be used in macro definitions.

LaTeX2HTML behaves differently when running on the same file.

When noticing strange side-effects due to files remaining from previous runs of `LaTeX2HTML` one can use the option `-no_reuse` and choose `(d)` when prompted. This deletes intermediate files generated during previous runs. One can also delete those files oneself by removing the complete subdirectory created by `LaTeX2HTML` for storing the translated files. Note that in this case the image reuse mechanism is disabled.

```
> latex2html -no_reuse myfile.tex
This is LaTeX2HTML Version 95.1 (Fri Jan 20 1995) by Nikos Drakos,
Computer Based Learning Unit, University of Leeds.
OPENING /afs/cern.ch/user/goossens/myfile.tex
Cannot create directory ./myfile: File exists
(r) Reuse the images in the old directory OR
(d) *** DELETE *** ./myfile AND ITS CONTENTS OR
(q) Quit ?
:d
```

Cannot convert PostScript images included in the LaTeX file.

It is likely that the macros used for including PostScript files (for example, `\epsffile` or `\includegraphics`) are not understood by `LaTeX2HTML`. To avoid this problem enclose them in an environment which will be passed to LaTeX anyway, for instance:

```
\begin{figure}
  \epsffile{<PostScript file name>}
\end{figure}
```

Another reason why this might happen is that the shell environment variable `TEXINPUTS` is undefined. This is not always fatal but if you have problems you can use full pathnames for included postscript files (even when the PostScript files are in the same directory as the LaTeX source file). Therefore it is important to check the setting of the `TEXINPUTS` environment variable and make sure that it ends in a colon ":", for example, "`.:/somedir:`".

Some of the inlined images are in the wrong places.

This occurs when any one of the inlined images is more than a (paper) page long. This is sometimes the case with very large tables or large PostScript images. In this case, one should specify a larger paper size (such as "a3", "a2", or even "a0") instead of the default ("a4") using the `LaTeX2HTML` variable `PAPERSIZE` in the file `latex2html.config`.

The labels of figures, tables or equations are wrong.

This can happen if inside figures, tables, equations or counters are used inside conditional text, i.e., inside a `latexonly` or a `htmlonly` environment.

With `Ghostscript 3.X` inline images are no longer generated for equations, etc.

One can run the installation script `install-test` again, or else change the way `gs` is invoked in the file `pstogif`, using something like:

```
open (GS, "|$GS -q -sDEVICE=ppmraw -sOutputFile=$base.ppm $base.ps");
```

Cannot get it to generate inlined images.

Try a small test file for example,

```
% image-test.tex
\documentclass{article}
\begin{document}
Some text followed by \fbox{some more text in a box}.
\end{document}
```

One should get something like the following:

```
> latex2html image-test.tex
This is LaTeX2HTML Version 95.1
        (Fri Jan 20 1995) by Nikos Drakos,
Computer Based Learning Unit, University of Leeds.

OPENING /afs/cern.ch/usr/goossens/image-test.tex

Reading ...
Processing macros ...
Translating ...0/1.....1/1.....
Writing image file ...
This is TeX, Version 3.1415 (C version 6.1)
(images.tex
LaTeX2e <1994/12/01>
Generating postscript images using dvips ...
This is dvipsk 5.58e Copyright 1986, 1994 Radical Eye Software
' TeX output 1995.05.08:1958' -> 6666_image
(-> 6666_image001) <tex.pro>[1]
Writing 6666_image001.ppm
```

```
Writing img1.gif
Doing section links .....
Done.
```

Problems encountered during the conversion from PostScript to **GIF** can be located by doing the translation manually, as shown below for a generation using `gs 3.33`.

```
> latex image-test
This is TeX, Version 3.1415 (C version 6.1)
(image-test.tex
LaTeX2e <1994/12/01>
(/usr/local/lib/texmf/tex/latex/base/article.cls
Document Class: article 1994/12/09 v1.2x Standard LaTeX document class
(/usr/local/lib/texmf/tex/latex/base/size10.clo))
No file image-test.aux.
[1] (image-test.aux) )
Output written on image-test.dvi (1 page, 348 bytes).
Transcript written on image-test.log.
> dvips -o image-test.ps image-test.dvi
This is dvipsk 5.58e Copyright 1986, 1994 Radical Eye Software
' TeX output 1995.05.08:2006' -> image-test.ps
<tex.pro>. [1]
cblelca% gs -dNODISPLAY pstoppm.ps
> gs -dNODISPLAY pstoppm.ps
Aladdin Ghostscript 3.33 (4/10/1995)
Copyright (C) 1995 Aladdin Enterprises, Menlo Park, CA.  All rights reserved.
This software comes with NO WARRANTY: see the file PUBLIC for details.
Usage: (file) ppmNrun
   converts file.ps to file.ppm (single page),
     or file.1ppm, file.2ppm, ... (multi page).
   N is # of bits per pixel (1, 8, or 24).
Examples: (golfer) ppm1run   ..or..   (escher) ppm8run
Optional commands you can give first:
   horiz_DPI vert_DPI ppmsetdensity
   horiz_inches vert_inches ppmsetpagesize
   (dirname/) ppmsetprefix
   page_num ppmsetfirstpagenumber
GS>(image-test) ppm1run
Writing image-test.ppm
GS>quit
> pnmcrop image-test.ppm >image-test.crop.ppm
pnmcrop: cropping 74 rows off the top
pnmcrop: cropping 139 rows off the bottom
pnmcrop: cropping 149 cols off the left
pnmcrop: cropping 249 cols off the right
> ppmtogif image-test.crop.ppm >image-test.gif
ppmtogif: computing colormap...
ppmtogif: 2 colors found
```

Still no inlined images are obtained.

When there have been no problems with the above file `image-test.tex` but some images have still not been successfully converted in some of the files then one should look in the directory with the generated HTML files for the two files `images.tex` and `images.log`. In particular, one should check whether there is something unusual in these files. One can copy the source `images.tex` into the directory of the original LaTeX file, run LaTeX on `images.tex` and check for any errors in the log file `images.log`. If errors are found then one should fix `images.tex`, put it back into the subdirectory with the HTML files, and run **LaTeX2HTML** on the original document using the option `-images_only`.

If one gets into trouble, then one should rerun **LaTeX2HTML** with the options `-no_reuse` and `-no_images`, for example,

```
> latex2html -no_reuse -no_images image-test.tex
This is LaTeX2HTML Version 95.1 (Fri Jan 20 1995) by Nikos Drakos,
Computer Based Learning Unit, University of Leeds.

OPENING /afs/cern.ch/user/goossens/image-test.tex
Cannot create directory ./image-test: File exists
(r) Reuse the images in the old directory OR
```

```
(d) *** DELETE *** ./image-test AND ITS CONTENTS OR
(q) Quit ?
:d

Reading ...
Processing macros ...
Translating ...0/1.....1/1.....
Writing image file ...

This is TeX, Version 3.1415 (C version 6.1)
(images.tex
LaTeX2e <1994/12/01>

Doing section links .....

*********** WARNINGS ***********

If you are having problems displaying the correct images with Mosaic,
try selecting "Flush Image Cache" from "Options" in the menu-bar and
then reload the HTML file.

Done.
```

Now one should look into the file `images.tex` (as described above) and correct possible problems. Once everything seems alright, **LaTeX2HTML** should be run again with the option **-images_only**.

Some problems in displaying the correct inlined images may be due to the image-caching mechanisms of the browser. With some browsers, a simple "Reload Current Document" will be enough to refresh the images, but with others (including `Mosaic`) one may need to refresh the cache explicitly. With `Mosaic` one should select "**Flush Image Cache**" in the `Options` menu, then reload the HTML file.

C For perl Hackers Only—Inside LaTeX2HTML

The basic principle of **LaTeX2HTML** is that it reads a LaTeX source code document, converts the parts it recognizes into HTML and passes unknown parts to LaTeX, which, in turn, creates pictures out of them. These pictures are then placed inside the final hypertext document.

As discussed in Section 3.3, the program is started by specifying the LaTeX source code document together with a set of parameters. The result is a number of HTML documents and images as **GIF** or PostScript files. An overall flow-diagram is shown in Figure 15

Unknown environments, tables, or pictures are also passed on to LaTeX and transformed into **GIF** or PostScript images, and kept inline or outside the hypertext documents.

C.1 The Translation Process

Below are shown the various phases that a document goes through when translated from LaTeX into HTML. Let us first consider the original LaTeX source document:

```
\documentclass{article}
\begin{document}
\section{test}

This is a list of two items:
\begin{itemize}
  \item{First item}
  \item{Second item}
\end{itemize}

\begin{verbatim}
This section includes some special characters such as $, <, >, _.
\end{verbatim}

\end{document}
```

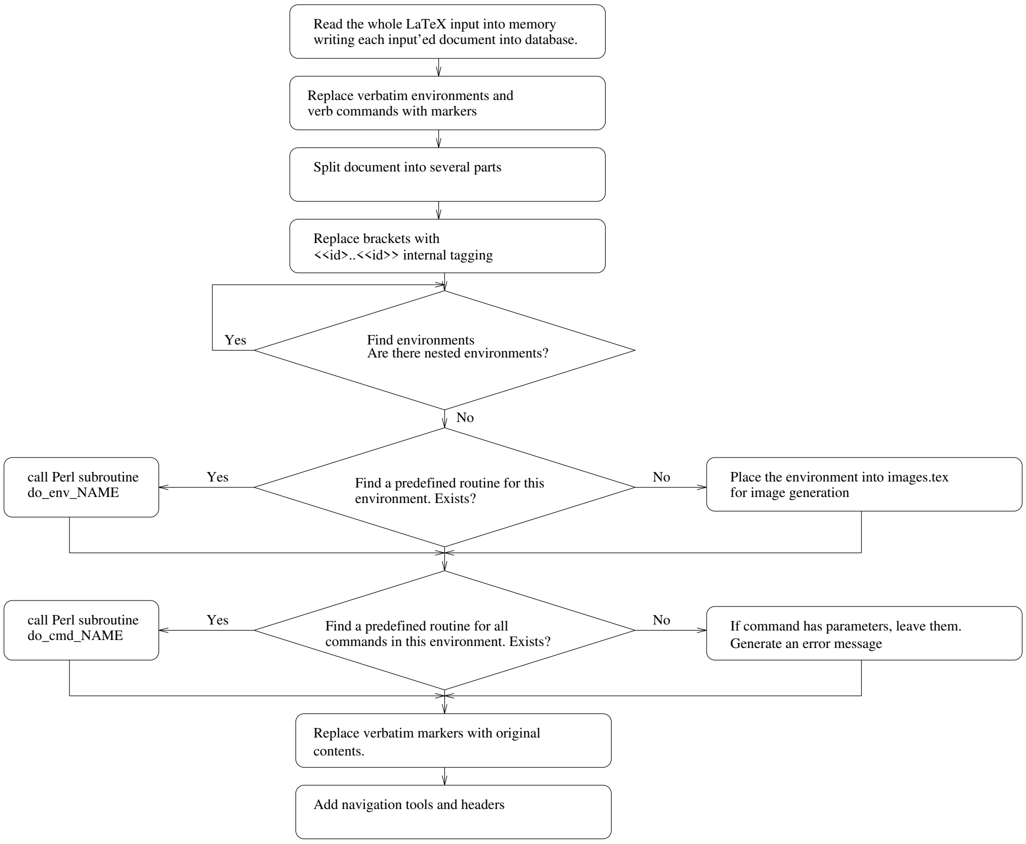


Figure 15: Flow diagram of the `LaTeX2HTML` system

This LaTeX source is first preprocessed by removing parts which have a special meaning in LaTeX, such as the **`verbatim`** and **`\verb`** constructs. In this example the verbatim part is stored in a separate file for later reference and a marker is placed inside the document together with a unique identification number "**`<id>`**" that will later be used to find the original text.

```
\documentclass{article}
\begin{document}
\section{test}

This is a list of two items:

\begin{itemize}
  \item{First item}
  \item{Second item}
\end{itemize}

<tex2html_verbatim_mark>verbatim1

\end{document}
```

At the end of preprocessing in the `mark_string` procedure, all the bracketed areas are replaced by **`<<id>><<id>>`** tags where "**`id`**" is identical at both ends of the originally bracketed text.

```
\documentclass<<1>>article<<1>>
\begin<<2>>document<<2>>
\section<<3>>test<<3>>

This is a list of two items:
```

```
\begin<<4>>itemize<<4>>
  \item<<5>>First item<<5>>
  \item<<6>>Second item<<6>>
\end<<7>>itemize<<7>>

<tex2html_verbatim_mark>verbatim1

\end<<8>>document<<8>>
```

Next, the document is split into sections. The LaTeX commands **\chapter**, **\section**, **\subsection**, etc. work as search-patterns used to split the document into items in an **perl** array. In our example, the conversion is configured to create a single document (i.e., no splitting).

For each section, the conversion rules are applied. These rules are implemented as procedures that have names like **do_env_X** or **do_cmd_X**, depending on whether one is dealing with a LaTeX environment or command, where **X** stands for either the environment or command name. For instance, our example document includes an **itemize** environment, and **LaTeX2HTML** will thus call the **perl** procedure **do_env_itemize**, that will receive as its parameter the contents of the environment, and will then parse that information.

Similarly a procedure **do_cmd_chapter** exists for converting a chapter command, and so on for the other sectioning commands. The resulting document after applying these conversion rules looks as follows.

```
<H1><A NAME=SECTION00100000000000000000> test</A></H1>
This is a list of two items:
<UL>
  <LI><#5#>First item<#5#>
  <LI><#6#>Second item<#6#>
</UL>
<tex2html_verbatim_mark>verbatim1
```

After this each document is enhanced with headers and navigation tools.

```
<!DOCTYPE HTML PUBLIC "-//W3O//DTD W3 HTML 2.0//EN">
<!Converted with LaTeX2HTML 95.1 (Fri Jan 20 1995) by Nikos
Drakos (nikos@cbl.leeds.ac.uk), CBLU, University of Leeds >
<HEAD>
<TITLE> test</TITLE>
</HEAD>
<BODY>
<meta name="description" value=" test">
<meta name="keywords" value="example">
<meta name="resource-type" value="document">
<meta name="distribution" value="global">
<BR>
<HR>
<A NAME=tex2html13 HREF="node2.html">
   <tex2html_next_page_visible_mark></A>
<A NAME=tex2html11 HREF="example.html">
   <tex2html_up_visible_mark></A>
<A NAME=tex2html5 HREF="example.html">
   <tex2html_previous_page_visible_mark></A>
<BR>
<B> Next:</B> <A NAME=tex2html14 HREF="node2.html">About this document</A>
<B>Up:</B>    <A NAME=tex2html12 HREF="example.html">No Title</A>
<B> Previous:</B><A NAME=tex2html6 HREF="example.html">No Title</A>
<BR>
<HR>
<P>
<H1><A NAME=SECTION00100000000000000000> test</A></H1>
This is a list of two items:
<UL><LI><#5#>First item<#5#>
    <LI><#6#>Second item<#6#>
```

```
</UL>
<tex2html_verbatim_mark>verbatim1
<BR>
<HR>
```

Finally, the markers are replaced with the contents to which they point. Extraneous tags are removed and the address of the author is appended to the file.

```
<!DOCTYPE HTML PUBLIC "-//W3O//DTD W3 HTML 2.0//EN">
<!Converted with LaTeX2HTML 95.1 (Fri Jan 20 1995) by Nikos
Drakos (nikos@cbl.leeds.ac.uk), CBLU, University of Leeds >
<HEAD>
<TITLE> test</TITLE>
</HEAD>
<BODY>
<meta name="description" value=" test">
<meta name="keywords" value="example">
<meta name="resource-type" value="document">
<meta name="distribution" value="global">
<P>
<BR>
<HR>
<A NAME=tex2html13 HREF="node2.html">
  <IMG ALIGN=BOTTOM ALT="next"
    SRC="http://asdwww.cern.ch/icons/next_motif.gif"></A>
<A NAME=tex2html11 HREF="example.html">
  <IMG ALIGN=BOTTOM ALT="up"
    SRC="http://asdwww.cern.ch/icons/up_motif.gif"></A>
<A NAME=tex2html5 HREF="example.html">
  <IMG ALIGN=BOTTOM ALT="previous"
    SRC="http://asdwww.cern.ch/icons/previous_motif.gif"></A>
<BR>
<B> Next:</B> <A NAME=tex2html14 HREF="node2.html">About this document</A>
<B>Up:</B>    <A NAME=tex2html12 HREF="example.html">No Title</A>
<B> Previous:</B> <A NAME=tex2html6 HREF="example.html">No Title</A>
<BR>
<HR>
<P>
<H1><A NAME=SECTION00100000000000000000> test</A></H1>
<P>
This is a list of two items:
<UL><LI>First item
  <LI>Second item
</UL>
<P>
<PRE>This section includes some special
characters such as $, &lt;, &gt;, _.
</PRE>
<P>
<BR> <HR>
```

C.2 Enhancing the Translator

From the previous section it is evident that the way to handle user commands and environments is to add **perl** code into the system or personal configuration files, as also discussed in Section 3.5. One can include as well a file with new definitions on the command line using the **-init_file** option.

To give a taste of how commands and environments are handled by **LaTeX2HTML**, we provide a few simple examples that nevertheless clearly show the powerful techniques used to generate HTML documents that preserve the information present in the original LaTeX document.

Let us first consider a LaTeX command (\Ucom) used to tag commands that have to be typed by the user on the keyboard. A possible definition using the HTML tag <KBD> for keyboard input is:

```
sub do_cmd_Ucom {
    local($_) = @_;
    s/$next_pair_pr_rx//o;
    join('',qq+<KBD>$&</KBD>+,$_);
}
```

The perl variable $next_pair_pr_rx contains the substitution pattern that extracts the string of characters surrounded by the following pair of delimiters. The string of characters and the delimiters are eliminated and the string is then copied between the HTML <KBD> and </KBD> appended to the output stream.

Similarly, one can translate the argument of a \URL command (containing a Universal Resource Locator) into an HTML anchor, as shown below:

```
sub do_cmd_URL {
    local($_) = @_;
    s/$next_pair_pr_rx//o;
    join('',"<a href=\"$&\">$&</a>",$_);
}
```

This procedure creates a link to the specified URL by returning an anchor with the URL as its target and an anchor description along with the rest of the as yet unprocessed document.

Our next example shows an enumerated list EnumZW of a special type whose "numbers" are icons available on a WWW server. The name of the icon depends on the value of the perl variable count, which is incremented for each \item command used inside the EnumZW environment. Everything takes place inside an HTML description list <DL>.

```
sub do_env_EnumZW {
    local($_) = @_;
    local($count) = 0;
    s|\\item|do {++$count; qq!<DT><IMG ALIGN=TOP ALT=""
    SRC="http://somewhere/icons/circled$count.xbm"><DD>!}|eog;
  "<DL COMPACT>$_</DL>";
    }
```

Two or more arguments can also be handled graciously, as shown by the following two commands, which have two and three arguments, respectively, and are typeset by LaTeX as follows:

\Command{*arg1*}

\Command[*arg1*]{*arg2*}

The translation in perl is straighforward, since one must merely extract the relevant arguments from the input stream, one after the other.

```
sub do_cmd_BDefCm { # \BDefCm{Command}{arg1}
    local($_) = @_;
    s/$next_pair_pr_rx//o; $command = $&;
    s/$next_pair_pr_rx//o; $mandatory1 = $&;
    join('',"<strong>\\$command\{$mandatory1\}<\/strong>", $_);
}

sub do_cmd_BDefCom { # \BDefCom{Command}{arg1}{arg2}
    local($_) = @_;
    s/$next_pair_pr_rx//o; $command = $&;
    s/$next_pair_pr_rx//o; $optional1 = $&;
    s/$next_pair_pr_rx//o; $mandatory1 = $&;
    join('',"<strong>\\$command\[$optional1\]\{$mandatory1\}<\/strong>", $_);
}
```

Explaining all this **perl** code would lead us a little too far, but it should be fairly clear by now that before trying to develop new code for **LaTeX2HTML** it is a good idea to study in detail the way Nikos Drakos coded his program, not only in order to write **perl** code compatible with his conventions, but also as a source of inspiration for one's own extensions. Below we show definitions for frequently-occurring regular expressions in the **LaTeX2HTML** **perl** code.

```
$delimiters = '\'\\s[\\]\\\\<>(=).,#;:~\/!-';
$delimiter_rx = "([$delimiters])";

# $1 : br_id
# $2 : <environment>
$begin_env_rx = "[\\\\]begin\\s*$O(\\d+)$C\\s*([^$delimiters]+)\\s*$O\\1$C\\s*";

$match_br_rx = "\\s*$O\\d+$C\\s*";

$optional_arg_rx = "^\\s*\\[([^]]+)\\]"; # Cannot handle nested []s!

# Matches a pair of matching brackets
# $1 : br_id
# $2 : contents
$next_pair_rx = "^[\\s%]*$O(\\d+)$C([\\s\\S]*)$O\\1$C";
$any_next_pair_rx = "$O(\\d+)$C([\\s\\S]*)$O\\1$C";
$any_next_pair_rx4 = "$O(\\d+)$C([\\s\\S]*)$O\\4$C";
$any_next_pair_rx5 = "$O(\\d+)$C([\\s\\S]*)$O\\5$C";

# $1 : br_id
$begin_cmd_rx = "$O(\\d+)$C";

# $1 : largest argument number
$tex_def_arg_rx = "^[#0-9]*#([0-9])$O";

# $1 : declaration or command or newline (\\)
$cmd_delims = q|-#,.~/\'`^"=|; # Commands which are also delimiters!
# The tex2html_dummy is an awful hack ....
$single_cmd_rx = "\\\\([$cmd_delims]|[^$delimiters]+|\\\\|(tex2html_dummy))";

# $1 : description in a list environment
$item_description_rx =
"\\\\item\\s*[[]\\s*((($any_next_pair_rx4)|([[][^]]*[]])|[^]])*)[]]";

$fontchange_rx = 'rm|em|bf|it|sl|sf|tt';

# Matches the \caption command
# $1 : br_id
# $2 : contents
$caption_rx = "\\\\caption\\s*([[]\\s*((($any_next_pair_rx5)|([[][^]]*[]])|[^]])*)[]])?$O(\\d+)$C([\\s\\S]*)$O\\8$C";

# Matches the \htmlimage command
# $1 : br_id
# $2 : contents
$htmlimage_rx = "\\\\htmlimage\\s*$O(\\d+)$C([\\s\\S]*)$O\\1$C";

# Matches a pair of matching brackets
# USING PROCESSED DELIMITERS;
# (the delimiters are processed during command translation)
# $1 : br_id
# $2 : contents
$next_pair_pr_rx = "^[\\s%]*$OP(\\d+)$CP([\\s\\S]*)$OP\\1$CP";
$any_next_pair_pr_rx = "$OP(\\d+)$CP([\\s\\S]*)$OP\\1$CP";

# This will be used to recognise escaped special characters as such
# and not as commands
$latex_specials_rx = '[\$]|&|%|#|{|}|_';

# This is used in sub revert_to_raw_tex before handing text to be processed by latex.
$html_specials_inv_rx = join("|", keys %html_specials_inv);

# This is also used in sub revert_to_raw_tex
$iso_latin1_character_rx = '(&#\d+;)';

# Matches a \begin or \end {tex2html_wrap}. Also used be revert_to_raw_tex
$tex2html_wrap_rx = '[\\\\](begin|end)\s*{\s*tex2html_wrap[_a-z]*\s*}';

$meta_cmd_rx = '[\\\\](renewcommand|renewenvironment|newcommand|newenvironment|newtheorem|def)';

# Matches counter commands - these are caught ealry and are appended to the
# file that is passed to latex.
$counters_rx ="[\\\\](newcounter|addtocounter|setcounter|refstepcounter|stepcounter|".
```

```
              "arabic|roman|Roman|alph|Alph|fnsymbol)$delimiter_rx";

# Matches a label command and its argument
$labels_rx = "[\\\\]label\\s*$O(\\d+)$C([\\s\\S]*)$O\\1$C";

# Matches environments that should not be touched during the translation
$verbatim_env_rx = "\\s*{(verbatim|rawhtml|LVerbatim)[*]?}";

# Matches icon markers
$icon_mark_rx = "<tex2html_(" . join("|", keys %icons) . ")>";

# Frequently used regular expressions with arguments
sub make_end_env_rx {
    local($env) = @_;
    $env = &escape_rx_chars($env);
    "[\\\\]end\\s*$O(\\d+)$C\\s*$env\\s*$O\\1$C";
}
sub make_begin_end_env_rx {
    local($env) = @_;
    $env = &escape_rx_chars($env);
    "[\\\\](begin|end)\\s*$O(\\d+)$C\\s*$env\\s*$O\\2$C(\\s*\$)?";
}
sub make_end_cmd_rx {
    local($br_id) = @_;
    "$O$br_id$C";
}
sub make_new_cmd_rx {
    "[\\\\](". join("|", keys %new_command) . ")"
if each %new_command;
}
sub make_new_env_rx {
    local($where) = @_;
    $where = &escape_rx_chars($where);
   "[\\\\]$where\\s*$O(\\d+)$C\\s*(".
       join("|", keys %new_environment) .
   ")\\s*$O\\1$C\\s*"
       if each %new_environment;
}
sub make_sections_rx {
    local($section_alts) = &get_current_sections;
    # $section_alts includes the *-forms of sectioning commands
    $sections_no_delim_rx = "\\\\($section_alts)";
    $sections_rx = "\\\\($section_alts)$delimiter_rx"
}
sub make_order_sensitive_rx {
    local(@theorem_alts, $theorem_alts);
    @theorem_alts = ($preamble =~ /\\newtheorem\s*{([^\s}]+)}/og);
    $theorem_alts = join('|',@theorem_alts);
    $order_sensitive_rx =
        "(equation|eqnarray|caption|ref|counter|\\\\the|\\\\stepcounter" .
        "|\\\\arabic|\\\\roman|\\\\Roman|\\\\alph|\\\\Alph|\\\\fnsymbol)";
    $order_sensitive_rx =~ s/\)/|$theorem_alts|/ if $theorem_alts;
}
sub make_language_rx {
    local($language_alts) = join("|", keys %language_translations);
    $setlanguage_rx = "\\\\setlanguage{\\\\($language_alts)}";
    $language_rx = "\\\\($language_alts)TeX";
}
sub make_raw_arg_cmd_rx {
    # $1 : commands to be processed in latex (with arguments untouched)
    $raw_arg_cmd_rx = "\\\\(" . &get_raw_arg_cmds . ")([$delimiters]+|\\\\|#|\$)";
}
# Creates an anchor for its argument and saves the information in the array %index;
# In the index the word will use the beginning of the title of
# the current section (instead of the usual pagenumber).
# The argument to the \index command is IGNORED (as in latex)
sub make_index_entry {
    local($br_id,$str) = @_;
    # If TITLE is not yet available (i.e the \index command is in the title of the
    # current section), use $ref_before.
    $TITLE = $ref_before unless $TITLE;
    # Save the reference
    $str = "$str###" . ++$global{'max_id'}; # Make unique
    $index{$str} .= &make_half_href("$CURRENT_FILE#$br_id");
    "<A NAME=$br_id>$anchor_invisible_mark<\/A>";
}
```

D Technical Details of the `math2html` Program

D.1 Different Approaches

Various people have approached the problem of translating LaTeX into SGML or HTML using different programming paradigms. Joachim Schrod of the Technical University of Darmstadt, Germany has written a lisp parser for TeX code which can also be used for conversions (Schrod 1994)[19]. As already discussed in Section 5, Xavier Leroy used `Caml` to achieve the same goal, while `LaTeX2HTML` uses `perl` (other approaches based on `sgmls` also use that language).

Common to all approaches, whether using a procedural or a functional language, is the basic implementation. A lexer is used to recognize tokens from the input, a parser to create an internal representation and the conversion process produces the wanted output.

The major difference between functional and procedural languages is the way a language such as TeX can be parsed. Since the TeX language can at any point in the input define new rules for delimiters and symbols, the program parsing this input should also be able to cope with these dynamic features. Functional programming languages can do this by their nature, easily introducing new rules to the parser at runtime. This is what the parser written by Joachim Schrod can do. In comparison this cannot easily be done with a fixed grammar inside a parser.

Xavier Leroy's translator resembles a `bison`[20] input file. It sees groups of tokens and reduces the stacked input by given BNF-like rules. When it reduces the tokens it produces HTML output for LaTeX counterparts.

D.2 Implementation of the Translator

The `math2html` program, written in C++, takes LaTeX mathematics input, parses it and converts it into HTML3 mathematics (if possible). The program consists of the following components:

- `flex`, a fast lexical analyzer generator;
- `bison`, a parser generator;
- C++ code.

The parsing of LaTeX source code is, however, non-trivial, since its grammar has been developed step-by-step to cope with all LaTeX syntactical notations. The basic mathematical notation is presented here in detail.

`\[...\]`	Display mathematics.
`txt1 $...$ txt2`	Inline mathematics.
`{abc}`	Characters `a`, `b` and `c` are grouped into one.
`\abc`	Characters `a`, `b` and `c` are a control sequence.
`a^b`	Superscripts (`b` can be a group of characters).
`a_b`	Subscripts (`b` can be a group of characters). Superscripts and subscripts can be nested.

The lexical analyser recognizes LaTeX primitives by generating tokens for the parser. A control sequence, plain text, superscript, subscript, begingroup, endgroup, fraction, array, column separators and end of row are examples of typical tokens. These tokens correspond to classes. These classes are depicted in Figure 16 with the object modeling technique (OMT) (Rumbaugh et al. 1991).

The class library presents the supported structures of LaTeX mathematics as sums, integrals, fractions, plain input, sequences and groups. These are currently the only primitives which can be reasonably converted into HTML3 mathematics. A few examples of basic primitives that can be treated by `math2html` are shown below:

```
Sum:      \sum_{i=1}^{n}i          Integral: \int_0^1f(x)dx
Fraction: \fraction{1}{n}          Sequence: \infty
Group:    {|x+1|}^2
Table:    \begin{table}{lr}        Eqntable: \begin{eqnarray}
            a & b \\ c & d                     y&=&x^2\\z&<=& x^3
          \end{table}                        \end{eqnarray}
```

[19] The system is available at URL `ftp://ftp.th-darmstadt.de/pub/tex/src/etls/`.

[20] Bison is a parser generator in the style of `yacc`.

The parser analyzes the tokens using an ad-hoc BNF grammar generated specifically to parse LaTeX code. When reducing the input according to the grammar rules, the parser generates instances of C++ classes (see Figure 16), which correspond to these LaTeX primitives. Once the whole input has been parsed, the internal representation is linked together so that all these instances can be reached from one top-level list.

The conversion is implemented by calling a conversion method to each instance in the list. Each primitive knows how to convert itself and also propagates the conversion to all its children nodes.

An instance of the runtime organization of the parsing tree corresponding to the example of Figure 10 is shown in Figure 17 on the next page.

D.3 Mapping of Control Sequences

Since the wide variety of different control sequences is quite impossible to hardcode into the program, an external configuration file is read every time the program starts. The mapping between control sequences and HTML3 counterparts is read into a hash table and in this way the user can configure the program to cope with special control sequences not natively supported by the converter. An example of this is the Particle Entity Notation scheme (Goossens and van Herwijnen 1992), a set of standard control sequences for representing elementary particles. This naming scheme consists of about 240 control sequences and their presentation counterparts. The configuration file maps each control sequence into its HTML3 counterpart using the following format:

```
\Pgppm     &pi;<sup>&plusmn;</sup>          \Pgpz      &pi;<sup>0</sup>
\Pgh       &eta;                            \Pgr       &rho;(770)
\Pgo       &omega;(783)                     \Pghpr     &eta;'(958)
\Pfz       <t>f</t><sub>0</sub>(975)
```

D.4 Program Heuristics

The program uses a few heuristics in order to be able to parse LaTeX code successfully. If these coding rules are not used, parsing may fail.

Optional parameters specified between square brackets (`[]`) after a control sequence are not parsed with respect to the control sequence. Therefore, there should be no space left between the control sequence and the opening bracketwhere optional parameters are used. Space should be left if the brackets are used as delimiters. An example is the difference between the following two control sequences:

```
\root[3]{\pi}                          \left [ \pi+2 ]
```

It is also worth noticing that all control sequences not supported primitively in `math2html`, apart from integrals, fractions, roots, sums and a few others, are dropped out during the conversion, for example, no text is produced in the HTML3 version. The only way to convert them is to create specific code or map it in the configuration file.

D.5 Interfacing with other Programs

This application was built to make it easy for other applications to call it. The program can either be compiled into a single executable program with a command line interface or into a library that can be linked with any other applications.

The modular approach has the advantage of being both simple and straightforward. The object-oriented implementation makes the linearisation of the internal representation almost effortless and eases the future addition of new HTML3 primitives by the user. The program is quite flexible and, as pointed out above, can be used in different contexts: embedded or stand-alone.

D.6 Drawbacks of the presented Solution

The end-user may find extending the program too difficult, especially if one has no experience with `flex`, `bison`, or C++. The configuration file that comes with the program provides an easy way to do simple mappings, but if one wants to add more functionality, one must understand the organization of the program.

As trickier tables and equations need to be converted, the program will need extension for analyzing the internal tree structure and to add, modify or delete specific nodes.

If the LaTeX input code uses low-level TeX commands the program will not be able to handle the input.

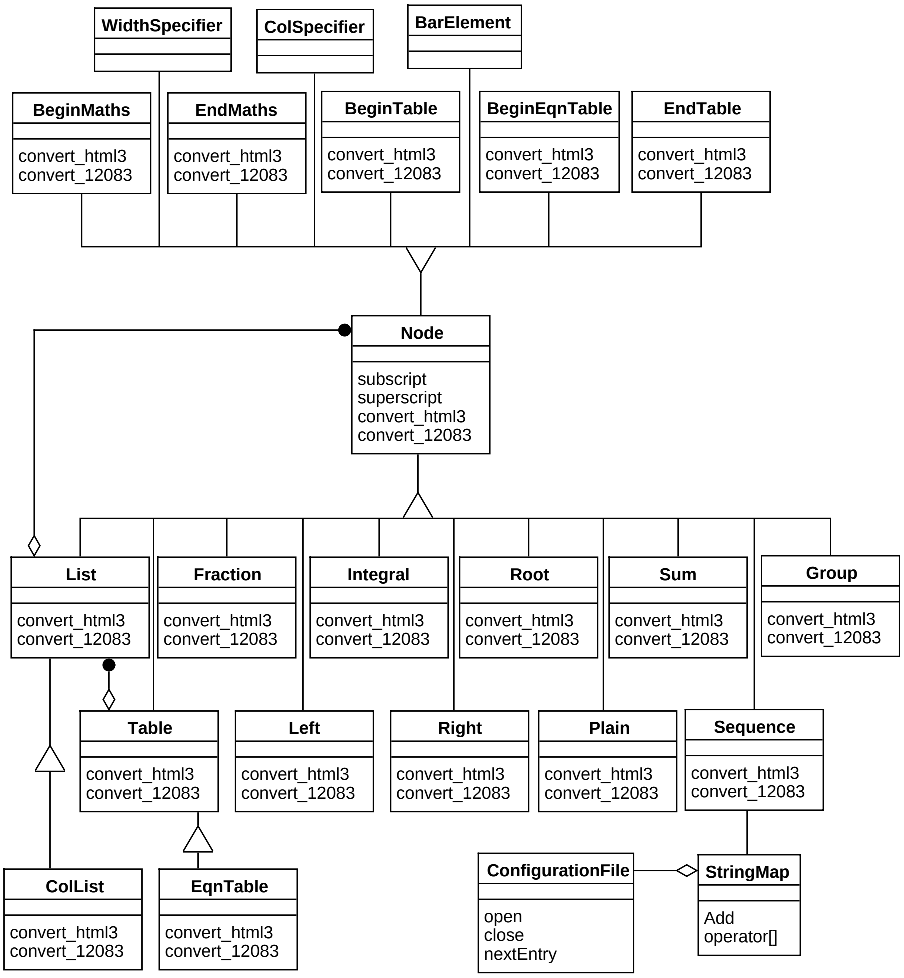


Figure 16: OMT model of the mathematics conversion program

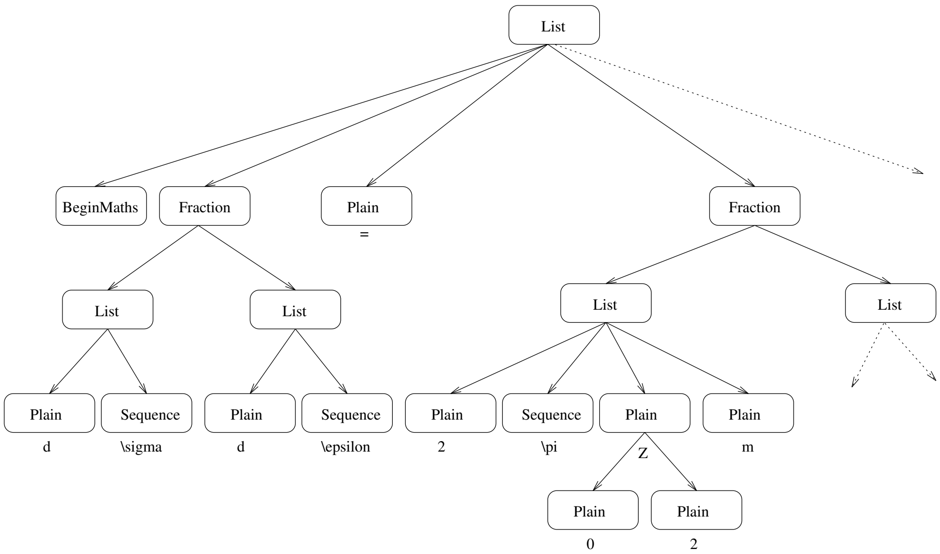


Figure 17: Example of a runtime parsing tree

E Using the `Caml` System for Translating LaTeX to HTML

The program works by expressing the LaTeX grammar in a YACC-like format and parsing the LaTeX input lines rule by rule, converting all recognized patterns into HTML. An example of `Caml Light` grammar rules for LaTeX to HTML conversion is given below.

```
(* Font changes *)

  | "{\\it" | "{\\em"
               { print_string "<i>"; upto ‘}‘ main lexbuf;
                 print_string "</i>"; main lexbuf }

  | "{\\bf"    { print_string "<b>"; upto ‘}‘ main lexbuf;
                 print_string "</b>"; main lexbuf }

  | "{\\tt"    { print_string "<tt>"; upto ‘}‘ main lexbuf;
                 print_string "</tt>"; main lexbuf }

  | ‘"‘        { print_string "<tt>"; indoublequote lexbuf;
                 print_string "</tt>"; main lexbuf }

(* Verb, verbatim *)

  | "\\verb" _  { verb_delim := get_lexeme_char lexbuf 5;
                 print_string "<tt>"; inverb lexbuf;
                 print_string "</tt>"; main lexbuf }

  | "\\begin{verbatim}"
               { print_string "<pre>"; inverbatim lexbuf;
                 print_string "</pre>"; main lexbuf }
```

Unlike `LaTeX2HTML` the program does not pass mathematics on to the TeX engine in order to create bitmap images for unparsable input, but produces plain text only. As the LaTeX control sequences recognized by the program are read from a separate file, the addition of new commands and their HTML counterparts is relatively easy. An example of such mappings is the following:

```
def "\\chapter"             [Print "<H1>"; Print_arg; Print "</H1>\n"];
def "\\chapter*"            [Print "<H1>"; Print_arg; Print "</H1>\n"];

def "\\begin{itemize}"      [Print "<p><ul>"];
def "\\end{itemize}"        [Print "</ul>"];

def "\\begin{enumerate}"    [Print "<p><ol>"];
def "\\end{enumerate}"      [Print "</ol>"];

def "\\begin{description}"  [Print "<p><dl>"];
def "\\end{description}"    [Print "</dl>"];

def "\\begin{center}"       [Print "<blockquote>"];
def "\\end{center}"         [Print "</blockquote>"];
```

The use of this program requires the compilation of the `Caml Light` distribution, available for a variety of platforms. The language is compiled with an intermediate step in the C language. The executable program suffers from some overhead, mainly affecting execution time.

Because the program does not deal with mathematics and tables, it can only be used for a restricted set of documents. To be useful for the general user it will have to be extended to convert mathematics and tables either into bitmaps or into HTML3.