HEVEA User Documentation

Version 1.07

Luc Maranget*

November 7, 2003

Abstract

HEVEA is a LaTeX to HTML translator. The input language is a fairly complete subset of LaTeX 2ε (old LaTeX style is also accepted) and the output language is HTML that is (hopefully) correct with respect to version 4.0 transitional.

Exotic symbols are translated into symbols pertaining to the symbol font of the HTML browser, using the now standard `FACE` attribute of the `FONT` tag. This allows the translation to HTML of quite a lot of the symbols used in LaTeX.

HEVEA understands LaTeX macro definitions. Simple user style files are understood with little or no modifications. Furthermore, HEVEA customization is done by writing LaTeX code.

HEVEA is written in Objective Caml, as many lexers. It is quite fast and flexible. Using HEVEA it is possible to translate large documents such as manuals, books, etc. very quickly. All documents are translated as one single HTML file. Then, the output file can be cut into smaller files, using the companion program HACHA.

HEVEA can also be instructed to output plain text or info files.

Information on HEVEA is available at `http://pauillac.inria.fr/~maranget/hevea/`.

*Inria Rocquencourt – BP 105, 78153 Le Chesnay Cedex. `Luc.Maranget@inria.fr`

Contents

Part A
Tutorial

1 How to get started

Assume that you have a file, "`a.tex`", written in LaTeX, using the *article*, *book* or *report* style. Then, translation is achieved by issuing the command:

```
# hevea a.tex
```

Probably, you will get some warnings. If HEVEA does not crash, just ignore them for the moment (Section 4 explains how to correct errors).

If everything goes fine, this will produce a new file, "`a.html`", which you can visualize through a HTML browser. If `a.tex` contains math symbols you need to instruct your browser to use symbol fonts (see section C.2).

If you wish to experiment HEVEA on small LaTeX source fragments, then launch HEVEA without arguments. HEVEA will read its standard input and print the translation on its standard output. For instance:

```
# hevea
$x \in {\cal E}$
^D
<I>x</I> <FONT FACE=symbol>Î</FONT> <FONT COLOR=red><I>E</I></FONT>
```

You can find some more elaborate examples[1] in the on-line documentation.

2 Style files

LaTeX style files are files that are not intended to produce output, but define document layout parameters, commands, environments, etc.

2.1 Standard base styles

The base style of a LaTeX document is the argument to the `\documentclass` command (`\documentstyle` in old style). Normally, the base style of a document defines the structure and appearance of the whole document.

HEVEA really knows about two LaTeX base styles, *article* and *book*. Additionally, the *report* base style is recognized and considered equivalent to *book* and the *seminar* base style for making slides is recognized and implemented by small additions on the *article* style.

Base style *style* is implemented by an HEVEA specific style file *style*`.hva`. More precisely, HEVEA interprets `\documentclass{`*style*`}` by attempting to load the file *style*`.hva` (see section C.1.1.1 on where HEVEA searches for files). Thus, at the moment, HEVEA distribution includes the files, `article.hva`, `book.hva`, etc.

2.2 Other base styles

Documents whose base style is not recognized by HEVEA can be processed when the unknown base style is a derivation of a recognized base style.

Let us assume that `mydoc.tex` uses an exotic base style such as *acmconf*. Then, typing `hevea mydoc.tex` will yield an error, since HEVEA cannot find the `acmconf.hva` file:

```
# hevea.opt mydoc.tex
mydoc.tex:1: Warning: Cannot find file: acmconf.hva
mydoc.tex:1: Error while reading LaTeX: No base style
Adios
```

[1] `http://pauillac.inria.fr/~maranget/hevea//examples/index.html`

This situation is avoided by invoking HEVEA with the known base style file `article.hva` as an extra argument:

```
# hevea article.hva mydoc.tex
```

The extra argument instructs HEVEA to load its `article.hva` style file before processing `mydoc.tex`. It will then ignore the document base style specified by `\documentclass` (or `\documentstyle`).

Observe that the fix above works because the *acmconf* and *article* base styles look the same to the document (i.e., they define the same macros). More generally, most base styles that are neither *article* nor *book* are in fact variations on either two of them. However, such styles usually provides extra macros. If users documents use these macros, then users should also instruct HEVEA about them (see section 4.1).

Finally, it is important to notice that renaming a base style file *style*`.cls` into *style*`.hva` will not work in general. As a matter of fact, base style files are TEX and not LATEX source and HEVEA will almost surely fail on TEX-ish input.

2.3 Other style files

A LATEX document usually loads additional style files, by using the commands `\input` or `\usepackage` or `\input`.

2.3.1 Files loaded with `\input`

Just like LATEX, HEVEA reacts to the construct `\input{`*file*`}` by loading the file *file*. (if I got it right, HEVEA even follows TEX crazy convention on "`.tex`" extensions).

As it is often the case, assume that the document `mydoc.tex` has a `\input{mymacros.tex}` instruction in its preamble, where `mymacros.tex` gathers custom definitions. Hopefully, only a few macros give rise to trouble: macros that performs fine typesetting or TEXish macros. Such macros need to be rewritten, using basic LATEX constructs (section 4 gives examples of macro-rewriting). The new definitions are best collected in a style file, `mymacros.hva` for instance. Then, `mydoc.tex` is to be translated by issuing the command:

```
# hevea mymacros.hva mydoc.tex
```

The file `mymacros.hva` is processed before `mydoc.tex` (and thus before `mymacros.tex`). As a consequence of HEVEA behavior with respect to definition and redefinition (see section B.8.1), the macro definitions in `mymacros.tex` override the ones in `mymacros.tex`, provided the document original definitions are performed by `\newcommand` (or `\newenvironment`).

Another situation is when HEVEA fails to process a whole style file. Usually, this means that HEVEA crashes on that style file. The basic idea is then to write a `mymacros.hva` style file that contains alternative definitions for all the commands defined in `mymacros.sty`. Then, HEVEA should be instructed to load `mymacros.hva` and not to load `mymacros.tex`. This is done by invoking `hevea` as follows:

```
# hevea mymacros.hva -e mymacros.tex mydoc.tex
```

Of course, `mymacros.hva` must now contain replacements for all the useful macros of `mymacro.tex`.

2.3.2 Files loaded with `\usepackage`

As far as I know, LATEX reacts to the construct `\usepackage{`*name*`}` by loading the file *name*`.sty`. HEVEA reacts in a similar, but different, manner, by loading the file *name*`.hva`.

HEVEA distributions already includes quite a few "`.hva`" implementations of famous packages (see section B.17). When a given package (say `zorglub`) is not implemented, the situation may not be as bad as it may seem first. Hopefully, you are only using a few commands from package `zorglub`, and you feel confident enough to implement them yourself. Then, it suffices to put your definitions in file `zorglub.hva` and HEVEA will react to `\usepackage{zorglub}` by loading `zorglub.hva`.

See section B.5.2 for the full story on `\usepackage`.

3 A note on style

3.1 Spacing, Paragraphs

Spacing in the HTML document reflects the original source spacing. More precisely, any sequence of spaces is outputed as one space, whereas a single newline is replicated in the output. However one blank line (i.e., two newlines in a row) or more introduce a paragraph break. Whether the tabulation character is a space or not is random, so avoid tabs in your source document.

Paragraphs are rendered by a blank line and there is no paragraph indentation. HEVEA is a bit simplistic in breaking paragraphs and extra paragraph breaks may be present in the final HTML documents. This can usually be corrected by modifying the source, without altering LaTeX output. For instance, some blank line before or after a comment or macro definition can be deleted.

Space after macros with no argument is skipped (as in LaTeX) — however this is not true in math mode, as explained in section 3.2.1 below. Consider the following example:

```
\newcommand{\open}{(}
\newcommand{\close}{)}
\open text opened by ``\verb+\open+''
and closed by ``\verb+\close+''\close.
```

We get:

(text opened by “`\open`” and closed by “`\close`”).

In the output above, the space after `\open` does not find its way to the output.

More generally, HEVEA tries to emulate LaTeX behavior in all situations, but discrepancies probably exist. Thus, users are invited to make explicit what they want. This is good practice anyway, because LaTeX is mysterious here. Consider the following example, where the `\tryspace` macro is first applied and then expansed by hand:

```
\newcommand{\bfsymbol}{\textbf{symbol}}
\newcommand{\tryspace}[1]{#1 XXX}

Some space: \tryspace{\bfsymbol}\\
No space: \bfsymbol XXX
```

Spacing is a bit chaotic here, the space after **symbol** remains when `#1` is substituted for it by LaTeX (or HEVEA).

Some space	: **symbol** XXX
No space	: **symbol**XXX

Note that, if a space before “XXX” is wanted, then one should probably write:

```
\newcommand{\tryspace}[1]{#1{} XXX}
```

3.2 Math mode

HEVEA math mode is not very far from normal text mode, except that all letters are shown in italics and that space after macros is echoed.

However, typesetting math formulas in HTML rises two difficulties. First, formulas contain symbols, such as Greek letters; second, even simple formulas do not follow the simple basic typesetting model of HTML.

3.2.1 Spacing in math mode

By contrast with LaTeX, spaces from the input are significant in math mode, this feature allows users to instruct HEVEA on how to put space in their formulas. For instance, `\alpha\rightarrow\beta` is typeset without spaces between symbols, whereas `\alpha \rightarrow \beta` produces these spaces. Note that LaTeX ignores spaces in math mode, so that users can freely adjust HEVEA output without changing anything to LaTeX output.

3.2.2 Symbols

Outputting symbols is performed using an HTML extension: the now standard `FACE` attribute to the `FONT` element instruct the browser to switch to a symbol font. HEVEA assumes this choice for the symbol font to be as shown by figure 1.

A browser correctly displays HEVEA symbols when figure 1 resembles the HTML page located at `symbol.html`[2] in HEVEA on-line documentation directory. Some browsers do not know about symbol fonts by default and need to be configured (see section C.2).

For authors that do not want to generate symbols that cannot be shown by any browser, HEVEA offers a degraded mode that outputs text in place of symbols. HEVEA operates in this mode when given the `-nosymb` flag. Replacement text is in English, unless HEVEA is also given the `-francais` flag. In that case replacement text is in French. For instance. the "∈" symbol is replace by "in" (or by "appartient à" if French mode is selected). This is far from being satisfactory, but degraded mode may be appropriate for documents than contain few symbols.

3.2.3 Displays

Apart from containing symbols, formulas specify strong typesetting constraints: sub-elements must be combined together following patterns that departs from normal text typesetting. For instance, fractions numerators and denominators must be placed one above the other. HEVEA handles such constraints in display mode only.

The main two operating modes of HEVEA are *text* mode and *display* mode. Text mode is the mode for typesetting normal text, when in this mode, text items are echoed one following the other and paragraph breaks are just blank lines, both in input and output. The so called *displayed-paragraph environments* of LaTeX (such as `center` or `quote`) are rendered by HTML block-level elements (such as `DIV` or `BLOCKQUOTE`). Rendering is correct becauses both LaTeX displayed environments and HTML block-level elements start a new line. Conversly, since opening a HTML block-level elements means starting a new line, any text that sould appear inside a paragraph must be translated using only HTML text-level elements. HEVEA chooses to translate in-text formulas that way.

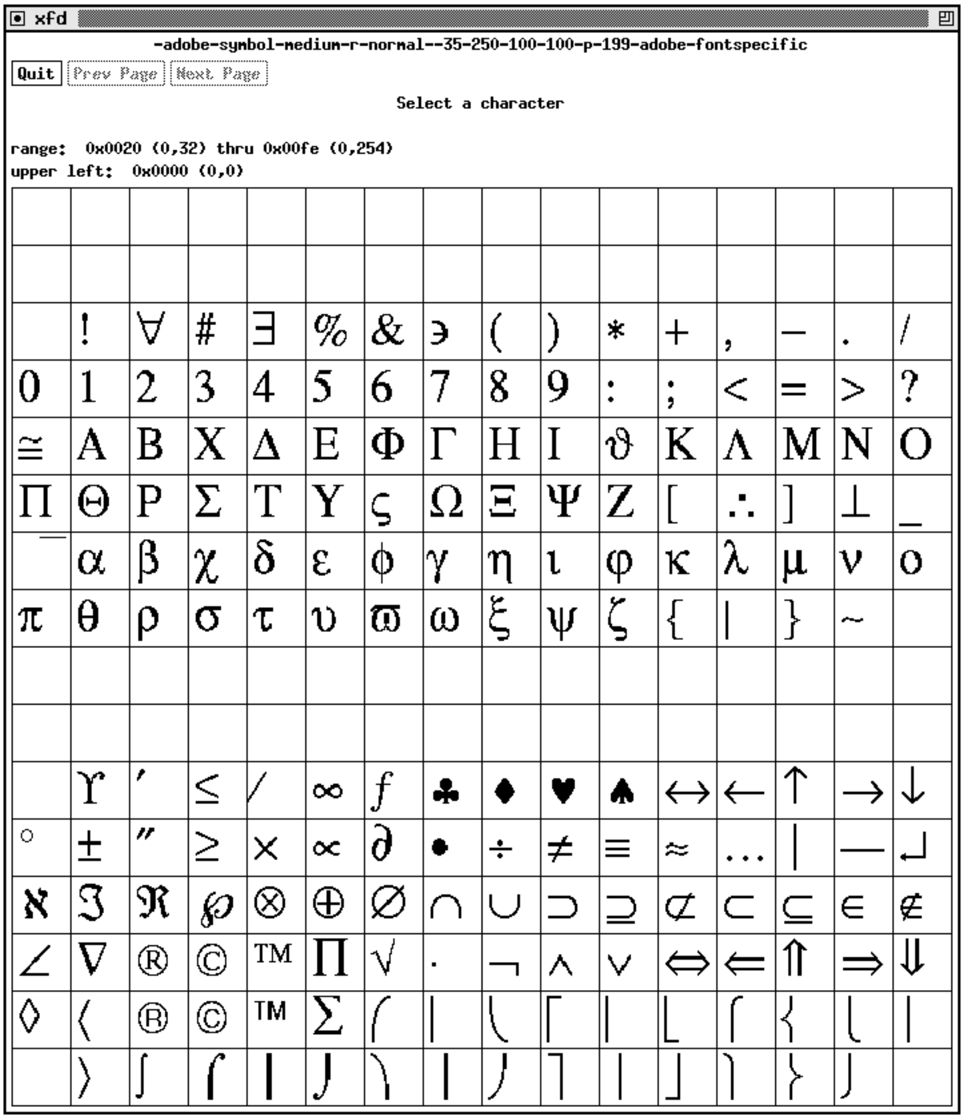
HEVEA display mode allows more control on text placement, since entering display mode means opening a HTML `TABLE` element and that tables allow to control the relative position of their sub-elements. Displays come in two flavor, horizontal displays and vertical displays. An horizontal display is a one-row table, while a vertical display is a one-column table. These tables holds display sub-elements, displays sub-elements being centered vertically in horizontal display mode and horizontally in vertical display mode.

Display mode is first opened by opening a `displaymath` environment (e.g. by `$$` or `\[`). Then, sub-displays are opened by LaTeX constructs which require them. For instance, a displayed fraction (`\frac`) opens a vertical display.

The distinction between text and display modes clearly appears while typesetting math formulas. An in-text formula such as `$\int_1^2 xdx = \frac{3}{2}$` appears as: $\int_1{}^2 xdx = 3/2$, while the same formula has a better aspect in display mode:

[2] `http://pauillac.inria.fr/~maranget/hevea//doc/symbol.html`

Figure 1: Symbol font in X



$$\int_{1}^{2} x dx = \frac{3}{2}$$

As a consequence, HEVEA is more powerful in display mode and formulas should be displayed as soon as they get a bit complicated. This rule is also true in LaTeX but it is more strict in HEVEA, since HTML capabilities to typeset formulas inside text are quite poor. In particular, it is not possible to get in-text "real" fractions or in-text limit-like subscripts.

Users should remember that HEVEA is not TeX or LaTeX and that HEVEA author neither is D. E. Knuth nor L. Lamport. Thus, some formulas may be rendered poorly. For instance, two fractions with different denominator and numerator height look strange.

$$\frac{1}{\sum_{i=0}^{N} U_i} = \frac{\sum_{i=0}^{N} U_i}{1}$$

The reason is that vertical displays in an horizontal display are HTML tables that always get centered in the vertical direction. Such a crude model cannot faithfully emulate any TeX box placement.

Users can get an idea on how HEVEA combines elements in display mode by giving the `-v` option comand line option twice, which instructs HEVEA to add a border to the `TABLE` elements introduced by displays.

3.2.4 Arrays and display mode

By contrast with formulas, which HEVEA attempts to render with text-level elements only when they appear inside paragraphs, LaTeX arrays always translate to the block-level element `TABLE`, thereby introducing non-desired line breaks before and after in-text arrays. As a consequence, in-text arrays yield an acceptable output, only while alone in a paragraph.

However, since in some sense, all HTML tables are displayed, the `array` and `tabular` environments implicitly open display mode, thus allowing a satisfactory typesetting of formulas in arrays. More precisely, array elements whose column format specification is `l`, `c` or `r` are typeset in display mode (see section B.10.2).

3.3 Warnings

When HEVEA thinks it cannot translate a symbol or construct properly, it issues a warning. This draws user attention onto a potential problem. However, rendering may be correct.

Note that all warnings can be suppressed with the `-s` (silent) option. When a warning reveals a real problem, it can often be cured by writing a specific macro. The next two sections introduce HEVEA macros, then section 4 describes how to proceed with greater detail.

3.4 Commands

Just like LaTeX, HEVEA can be seen as a macro language, macros are rewritten until no more expansion is possible. Then, either some characters (such as letters, integers…) are outputed or some internal operation (such as changing font attributes, or arranging text items in a certain manner) are performed.

This scheme favors easy extension of program capabilities by users. However, predicting program behavior and correcting errors may prove difficult, since final output or errors may occur after several levels of macro expansion. As a consequence, users can tailor HEVEA to their needs, but it remains a subtle task. Nevertheless, happy LATEX users should enjoy customizing HEVEA, since this is done by writing LATEX code.

3.5 Style choices

LATEX and HTML differ in many aspects. For instance, LATEX allows fine control over text placement, whereas HTML does not. More symbols and font attributes are available in LATEX than in HTML. Conversely, HTML has font attributes, such as color, which standard LATEX has not.

Therefore, there are many situations where HEVEA just cannot render the visual effect of LATEX constructions. Here some choices have to be made. For instance, the calligraphic letters (`\mathcal`) are rendered in red (`<FONT COLOR=red>`), and the small caps (`\scshape`) are rendered in bold font (`<B>`).

If you are not satisfied with HEVEA rendering of text style declarations, then you can choose your own, by redefining the `\cal` and `\sc` macros, using `\renewcommand`, the macro redefinition operator of LATEX. The key point is that you need not worry about HEVEA internals: just redefine the old-LATEX style text-style declarations (i.e. `\it`, `\sc`, etc.) and everything should get fine:

```
\renewcommand{\sc}{\Huge}
\renewcommand{\cal}{\em}
```

(See sections 4 and 5 on how to make such changes while leaving your file processable by LATEX, and section 9.2 for a more thourough descriton of customizing type styles).

Note that many of LATEX commands and environments are defined in the `hevea.hva` file that HEVEA loads before processing any input. These constructs are written using LATEX source code, in the end they invoke HEVEA internal commands.

Other LATEX constructs, such as LATEX key constructs or HEVEA internal commands (see section 8.4), that require special processing are defined in HEVEA source code. However, the vast majority of these definitions can be overridden by a redefinition. This may prove useless, since there is little point in redefining core constructs such as `\newcommand` for instance.

4 How to detect and correct errors

Most of the problems that occur during the translation of a given LATEX file (say `trouble.tex`) can be detected and solved at the macro-level. That is, most problems induce a macro-related warning and can be solved by writing a few macros. The best place for these macros is an user style file (say `trouble.hva`) given as argument to HEVEA.

```
# hevea trouble.hva trouble.tex
```

By doing so, the macros written specially for HEVEA are not seen by LATEX. Even better, `trouble.tex` is not changed at all.

Of course, this will be easier if the LATEX source is written in a generic style, using macros. Note that this style is recommended anyway, since it eases the changing and tuning of documents.

4.1 HEVEA does not know a macro

Consider the following LATEX source excerpt:

```
You can \raisebox{.6ex}{\em raise} text.
```

LATEX typesets this as follows:

You can *raise* text.

Since HEVEA does not know about `\raisebox`, it incorrectly processes this input. More precisely, it first prints a warning message:

```
trouble.tex:34: Unknown macro: \raisebox
```

Then, it goes on by translating the arguments of `\raisebox` as if they were normal text. As a consequence some `.6ex` is finally found in the HTML output:

You can .6ex*raise* text.

To correct this, you should provide a macro that has more or less the effect of `\raisebox`. It is impossible to write a generic `\raisebox` macro for HEVEA, because of HTML limitations. However, in this case, the effect of `\raisebox` is to raise the box *a little*. Thus, the first, numerical, argument to `\raisebox` can be ignored in a private `\raisebox` macro defined in `trouble.hva`:

```
\newcommand{\raisebox}[2]{$^{\mbox{#2}}$}
```

Now, translating the document yields:

You can $^{\textit{raise}}$ text a little.

Of course, this will work only when all `\raisebox` commands in the document raise text a little. Consider, the following example, where text is both raised a lowered a little:

```
You can \raisebox{.6ex}{\em raise}
or \raisebox{-.6ex}{\em lower} text.
```

Which LaTeX renders as follows:

You can $^{\textit{raise}}$ or $_{\textit{lower}}$ text.

Whereas, with the above definition of `\raisebox`, HEVEA produces:

You can $^{\textit{raise}}$ or $^{\textit{lower}}$ text.

A solution is to add a new macro definition in the `trouble.hva` file:

```
\newcommand{\lowerbox}[2]{$_{\mbox{#2}}$}
```

Then, `trouble.tex` itself has to be modified a little.

```
You can \raisebox{.6ex}{\em raise}
or \lowerbox{-.6ex}{\em lower} text.
```

HEVEA now produces a satisfying output:

You can $^{\textit{raise}}$ or $_{\textit{lower}}$ text.

Note that, for the document to remain LaTeX-processable, it should also contain the following definition for `\lowerbox`:

```
\newcommand{\lowerbox}[2]{\raisebox{#1}{#2}}
```

This definition can safely be placed anywhere in `trouble.tex`, since by HEVEA semantics for `\newcommand` (see section B.8.1) the new definition will not overwrite the old one.

4.2 HEVEA incorrectly interprets a macro

Sometimes HEVEA knows about a macro, but the produced HTML does not look good when seen through a browser. This kind of errors is detected while visually checking the output. However, HEVEA does its best to issue warnings when such situations are likely to occur.

Consider, for instance, this definition of `\blob` as a small black square.

```
\newcommand{\blob}{\rule[.2ex]{1ex}{1ex}}
\blob\ Blob \blob
```

Which LaTeX typesets as follows:

■ Blob ■

HEVEA always translates `\rule` as `<HR>`, ignoring size arguments. Hence, it produces the following, wrong, output:

There is not small square in the symbol font used by HEVEA. However there are other small symbols that would perfectly do the job of `\blob`, such as a bullet (`\bullet`). Thus, you may choose to give `\blob` a definition in `trouble.hva`:

```
\newcommand{\blob}{\bullet}
```

This new definition yields the following, more satisfying output:

Now, if you insist on having a square "blob", you can. It suffices to have LaTeX typeset some subparts of the document and then to include them as images, section 6 explain how to proceed.

4.3 HEVEA crashes

HEVEA failure may have many causes, including a bug. However, it may also stem from a wrong LaTeX input. Thus, this section is to be read before reporting a bug...

4.3.1 Simple cases: LaTeX also crashes

In the following source, environments are not properly balanced:

```
\begin{flushright}
\begin{quote}
This is right-flushed quoted text.
\end{flushright}
\end{quote}
```

Such a source will make both LaTeX and HEVEA choke. HEVEA issues the following error message that shows the LaTeX environment that is not closed properly:

```
trouble.tex:7: hml: DIV closes BLOCKQUOTE
trouble.tex:5: Latex environment ‘‘quote’’ is pending
Adios
```

Thus, when HEVEA crashes, it is a good idea to check that the input is correct by running LaTeX on it.

4.3.2 Complicated cases

Unfortunately, HEVEA may crash on input that does not affect LaTeX. Such errors usually relate to environment or group nesting.

Consider for instance the following "optimized" version of a quoteright environment:

```
\newenvironment{quoteright}{\quote\flushright}{\endquote}

\begin{quoteright}
This a right-flushed quotation
\end{quoteright}
```

The \quote and \flushright constructs are intended to replace \begin{quote} and \begin{flushright}, while \endquote stands for \end{quote}. Note that the closing \endflushright is omitted, since it does nothing. LaTeX accepts such an input and produces a right-flushed quotation.

However, HEVEA usually translates LaTeX environments to HTML block-level elements and it *requires* those elements to be nested properly. Here, \quote translates to <BLOCKQUOTE>, \flushright translates to <DIV ALIGN=right> and \endquote translates to </BLOCKQUOTE>. At that point, HEVEA refuses to generate obviously non-correct HTML and it crashes:

```
trouble.tex:9: hml: BLOCKQUOTE closes DIV
trouble.tex:7: Latex environment ‘‘quoteright’’ is pending
Adios
```

In this case, the solution is easy: environments must be opened and closed consistently. LaTeX style being recommended, one should write:

```
\newenvironment{quoteright}
  {\begin{quote}\begin{flushright}}
  {\end{flushright}\end{quote}}
```

And we get:

Unclosed LaTeX groups ({...}) are another source of nuisance to HEVEA. Consider the following horreur.tex file:

```
\documentclass{article}

\begin{document}
In this sentence, a group is opened now {\em and never closed.
\end{document}
```

LaTeX accepts this file, although it produces a warning:

```
# latex horreur.tex
This is TeX, Version 3.14159 (Web2C 7.2)
  ...
(\end occurred inside a group at level 1)
Output written on horreur.dvi (1 page, 280 bytes).
```

By contrast, running HEVEA on horreur.tex yields a fatal error:

```
# hevea horreur.tex
horreur.tex:5: Latex env error: ``document'' closes ``''
horreur.tex:4: Latex environment ``'' is pending
Adios
```

Thus, users should close opening braces where it belongs. Note that HEVEA error message "`Latex environment ``env'' is pending`" helps a lot in locating the brace that hurts.

4.3.3 Desperate cases

If HEVEA crashes on LATEX source (not on TEX source), then you may have discovered a bug, or this manual is not as complete as it should. In any case, please report to `Luc.Maranget@inria.fr`.

To be useful, your bug report should include LATEX code that triggers the bug (the shorter, the better) and mention HEVEA version number.

5 Making HEVEA and LATEX both happy

A satisfactory translation from LATEX to HTML often requires giving instructions to HEVEA. Typically, these instructions are macro definitions and these instructions should not be seen by LATEX. Conversely, some source that LATEX needs should not be processed by HEVEA. Basically, there are three ways to make input vary according to the processor, file loading, the `hevea` package and comments.

5.1 File loading

HEVEA and LATEX treat files differently. Here is a summary of the main differences:

- LATEX and HEVEA both load files given as arguments to `\input`, however when given the option `-e` *filename*, HEVEA does not load *filename*.
- HEVEA loads all files given as command line arguments.
- Both LATEX and HEVEA load style files given as optional arguments to `\documentstyle` and as arguments to `\usepackage`, but the files are searched by following different methods and considering different file extensions.

As a consequence, for having a file *latexonly* loaded by LATEX only, it suffices o use `\input{`*latexonly*`}` in the source and to invoke HEVEA as follows:

`# hevea -e` *latexonly*...

Having *heveaonly* loaded by HEVEA only is more simple: it suffices to invoke HEVEA as follows:

`# hevea` *heveaonly*...

Finally, if one has an HEVEA equivalent *style*`.hva` for a LATEX style file *style*`.sty`, then one should load the file as follows:

`\usepackage{`*style*`}`

This will result in, LATEX loading *style*`.sty`, while HEVEA loads *style*`.hva`. As HEVEA will not fail in case *style*`.hva` does not exist, this is another method for having a style file loaded by LATEX only.

Writing an HEVEA-specific file *file*`.hva` is the method of choice for supplying command definitions to HEVEA only. Users can then be sure that these definitions are not seen by LATEX and will not get echoed to the *image* file (see section 6).

The file *file*`.hva` can be loaded by either supplying the command-line argument *file*`.hva`, or by `\usepackage{`*file*`}` from inside the document. Which method is better depends on whether it is choosed to override or to replace the document definition. In the command-line case, definitions from *file*`.hva` are processed before the ones from the document and will override them, provided the document definitions are made using `\newcommand` (or `\newenvironment`). In the `\usepackage` case, HEVEA loads *file*`.hva` at the place where LATEX loads *file*`.sty`, hence the definitions from *file*`.hva` replace the definitions from *file*`.sty` in the strict sense.

5.2 The `hevea` package

The `hevea.sty` style file is intended to be loaded by LaTeX and not by HEVEA. It provides LaTeX with means to ignore or process some parts of the document. Note that HEVEA copes with the constructs defined in the `hevea.sty` file by default. It is important to notice that the `hevea.sty` style file from the distribution is a *package* in LaTeX 2_ε terms and that it is not compatible with old LaTeX. Moreover, the `hevea` package loads the `comment` package which must be present.

5.2.1 Environments for selecting a translator

HEVEA and LaTeX perform the following actions on source inside the `latexonly`, `verblatex`, `htmlonly`, `rawhtml`, `toimage` and `verbimage` environments:

environment	HEVEA	LaTeX
`latexonly`	ignore, `\end{`*env*`}` constructs are processed (see section 5.2.2)	process
`verblatex`	ignore	process
`htmlonly`	process	ignore
`rawhtml`	echo verbatim (see section 8.3)	ignore
`toimage`	send to the *image* file, `\end{`*env*`}` constructs and macro characters are processed (see section 6)	process
`verbimage`	send to the *image* file (see section 6)	process

As an example, this is how some text can be typeset in purple by HEVEA and left alone by LaTeX:

```
We get:
\begin{htmlonly}%
\purple purple rain, purple rain%
\end{htmlonly}
\begin{latexonly}%
purple rain, purple rain%
\end{latexonly}%
\ldots
```

We get: purple rain, purple rain...

It is impossible to avoid the spurious space in HEVEA output for the source above. This extra spaces comes from the newline character that follows `\end{htmlonly}`. Namely this construct must appear in a line of its own for LaTeX to recognize it. Anyway, better control over spaces can be achieved by using the `hevea` boolean register or comments, see sections 5.2.3 and 5.3.

Also note that environments define a scope and that style changes (and non-global definitions) are local to them. For instance, in the example above, "..." appears in black in HTML output. However, as an exception, the environments `image` and `verbimage` do not create scope. It takes a little practice of HEVEA to understand why this is convenient.

5.2.2 Why are there two environments for ignoring input?

Some scanning and analysis of source is performed by HEVEA inside the `latexonly` environment, in order to allow `latexonly` to dynamically occur inside other environments.

More specifically, `\end{`*env*`}` macros are recognized and their *env* argument is tested against the name of the environment whose opening macro `\`*env* opened the `latexonly` environment. In that case, macro expansion of `\end`*env* is performed and any further occurrence of `\end{`*env'*`}` is tested and may get expanded if it matches a pending `\begin{`*env'*`}` construct.

This enables playing tricks such as:

```
\newenvironment{latexhuge}
  {\begin{latexonly}\huge}
```

```
  {\end{latexonly}}

\begin{latexhuge}
This will appear in huge font in \LaTeX{} output only.
\end{latexhuge}
```

LaTeX output will be:

This will appear in huge font in LaTeX output only.

While there is no HEVEA output.

Since HEVEA somehow analyses input that is enclosed in the `latexonly` environment, it may choke. However, this environment is intended to select processing by LaTeX only and might contain arbitrary source code. Fortunately, it remains possible to have input processed by LaTeX only, regardless of what it is, by enclosing it in the `verblatex` environment. Inside this environment, HEVEA performs no other action than looking for `\end{verblatex}`. As a consequence, the `\begin{verblatex}` and `\end{verblatex}` constructs may only appear in the main flow of text or inside the same macro body, a bit like LaTeX `verbatim` environment.

Relations between `toimage` and `verbimage` are similar. Additionally, formal parameters `#`i are replaced by actual arguments inside the `toimage` environment (see end of section 6.3 for an example of this feature).

5.2.3 The hevea boolean register

Boolean registers are provided by the `ifthen` package (see [LaTeX, Section C.8.5] and section B.8.5 in this document). Both the `hevea.sty` style file and HEVEA define the boolean register `hevea`. However, this register initial value is *false* for LaTeX and *true* for HEVEA.

Thus, provided, both the `hevea.sty` style file and the `ifthen` packages are loaded, the "purple rain" example can be rephrased as follows:

```
We get:
{\ifthenelse{\boolean{hevea}}{\purple}{}purple rain, purple rain}\ldots
```

We get: purple rain, purple rain...

Another choice is using the TeX-style conditional macro `\ifhevea` (see Section B.16.2.4):

```
We get:
{\ifhevea\purple\fi purple rain, purple rain}\ldots
```

We get: purple rain, purple rain...

5.3 Comments

HEVEA processes all lines that start with `%HEVEA`, while LaTeX treats these lines as comments. Thus, this is a last variation on the "purple rain" example:

```
We get
%HEVEA{\purple
purple rain, purple rain%
%HEVEA}%
\ldots
```

(Note how comments are placed at the end of some lines to avoid spurious spaces in the final output.)

We get: purple rain, purple rain...

Comments thus provide an alternative to loading the `hevea` package. For user convenience, comment equivalents to the `latexonly` and `toimage` environment are also provided:

environment	comment equivalent
`\begin{latexonly}`... `\end{latexonly}`	`%BEGIN LATEX` `...` `%END LATEX`
`\begin{toimage}`... `\end{toimage}`	`%BEGIN IMAGE` `...` `%END IMAGE`

Note that LaTeX, by ignoring comments, naturally performs the action of processing text between `%BEGIN...` and `%END...` comments. However, no environment is opened and closed and no scope is created while using comment equivalents.

6 With a little help from LaTeX

Sometimes, HEVEA just cannot process its input, but it remains acceptable to have LaTeX process it, to produce a `.gif` image from LaTeX output and to include a link to this image into HEVEA output. HEVEA provides a limited support for doing this.

6.1 The *image* file

While outputting *mydoc*`.html`, HEVEA echoes some of its input to the *image* file, *mydoc*`.image.tex`.

Part of this process is done at the user's request. More precisely, the following two constructs send *text* to the *image* file:

`\begin{toimage}`
text
`\end{toimage}`

`%BEGIN IMAGE`
text
`%END IMAGE`

Additionally, `\usepackage` commands, top-level and global definitions are automatically echoed to the image file. This enables using document-specific commands in *text* above.

Output to the image file builds up a current page, which is flushed by the `\imageflush` command. This command has the following effect: it outputs a strict page break in the *image* file, increments the image counter and output a `<IMG SRC="`*pagename*`.gif">` tag in HEVEA output file, where *pagename* is build from the image counter and HEVEA output file name.

Then the `imagen` script has to be run by:

`# imagen` *mydoc*

This will process the *mydoc*`.image.tex` file through LaTeX, `dvips`, `ghostscript` and a few others tools, which must all be present (see section C.4.1), finally producing one *pagename*`.gif` file per page in the *image* file.

The usage of `imagen` is described at section C.1.4. Note that `imagen` is a simple shell script.

6.2 A toy example

Consider the "blob" example from section 4.2. Here is the active part of a `blob.tex` file:

```
\newcommand{\blob}{\rule[.2ex]{1ex}{1ex}}
\blob\ Blob \blob
```

This time, we would like `\blob` to produce a small black square, which `\rule[.2ex]{1ex}{1ex}` indeed does in LaTeX. Thus we can write:

```
\newcommand{\blob}{%
\begin{toimage}\rule[.2ex]{1ex}{1ex}%
\end{toimage}%
\imageflush}
\blob\ Blob \blob
```

Now we issue the following two commands:

```
# hevea blob.tex
# imagen blob
```

And we get:

Observe that the trick can be used to replace missing symbols by small `.gif` images. However, the cost may be prohibitive, text rendering is generally bad, fine placement is ignored and font style changes are problematic. Cost can be lowered using `\savebox`, but the other problems remain.

6.3 Including Postscript images

In this section, a technique to transform included Postscript images into included GIF images is described. Note that this technique is used by HeVeA implementation of the `graphics` package (see section B.14.1), which provides a more standard manner to include Postcript images in LaTeX documents.

Included images are easy to manage: it suffices to let LaTeX do the job. Let `round.ps` be a Postscript file, which is included as an image in the source file `round.tex` (which must load the *epsf* package):

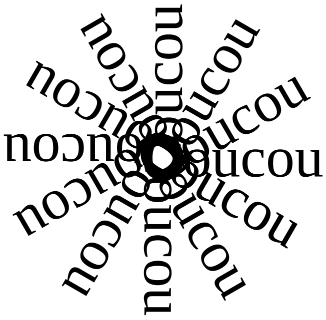
```
\begin{center}
\epsfbox{round.ps}
\end{center}
```

Then, HeVeA can have this image translated into a inlined (and centered) `.gif` image by modifying source as follows:

```
\begin{center}
%BEGIN IMAGE
\epsfbox{round.ps}
%END IMAGE
%HEVEA\imageflush
\end{center}
```

(Note that the `round.tex` file still can be processed by LaTeX, since comment equivalents of the `toimage` environment are used and that the `\imageflush` command is inside a `%HEVEA` comment — see section 5.3.)

Then, processing `round.tex` through HeVeA and `imagen` yields:



It is important to notice that things go smoothly because the `\usepackage{epsf}` command gets echoed to the *image* file. In more complicated cases, LaTeX may fail on the *image* file because it does not load the right packages or define the right macros.

However, the above solution implies modifying the original LaTeX source code. A better solution is to define the `\epsfbox` command, so that HEVEA echoes `\epsfbox` and its argument to the image file and performs `\imageflush`:

```
\newcommand{\epsfbox}[1]{%
\begin{toimage}
\epsfbox{#1}
\end{toimage}
\imageflush}
```

Such a definition must be seen by HEVEA only. So, it is best put in a separate file whose name is given as an extra argument on HEVEA command line (see section 5.1). Putting it in the document source protected inside an `%HEVEA` comment is a bad idea, because it might then get echoed to the *image* file and generate trouble when LaTeX is later run by `imagen`.

Observe that the above definition of `\epsfbox` is a definition and not a redefinition (i.e., `\newcommand` is used and not `\renewcommand`), because HEVEA does not know about `\epsfbox` by default. Also observe that this not a recursive definition, since commands do not get expanded inside the `toimage` environment.

Finally, if the Postscript image is produced from a bitmap, it is a pity to translate it back into a bitmap. A better idea is first to generate a GIF file from the bitmap source independantly and then to include a link to that GIF file in HTML output, see section 8.2 for a description of this more adequate technique.

6.4 Using filters

Some programs extend LaTeX capabilities using a filter principle. In such a scheme, the document contains source fragments for the program. A first run of the program on LaTeX source changes these fragments into constructs that LaTeX (or a subsequent stage in the paper document production chain, such as `dvips`) can handle. Here again, the rule of the game is keeping HEVEA away from the normal process: first applying the filter, then making HEVEA send the filter output to the *image* file, and then having `imagen` do the job.

Consider the `gpic` filter, for making drawings. Source for `gpic` is enclosed in `.PS`...`.PE`, then the result is available to subsequent LaTeX source as a TeX box `\box\graph`. For instance the following source, from a `smile.tex` file, draws a "Smile!" logo as a centered paragraph:

```
.PS
ellipse "{\Large\bf Smile!}"
.PE
\begin{center}
~\box\graph~
\end{center}
```

Both the image description (.PS... .PE) and usage (\box\graph) are for the *image* file, and they should be enclosed by %BEGIN IMAGE... %END IMAGE comments. Additionally, the image link is put where it belongs by an \imageflush command:

```
%BEGIN IMAGE
.PS
ellipse "{\Large\bf Smile!}"
.PE
%END IMAGE
\begin{center}
%BEGIN IMAGE
~\box\graph~
%END IMAGE
%HEVEA\imageflush
\end{center}
```

The gpic filter is applied first, then come hevea and imagen:

```
# gpic -t < smile.tex > tmp.tex
# hevea tmp.tex -o smile.html
# imagen smile
```

And we get:

Observe how the -o argument to HEVEA is used and that imagen argument is HEVEA output basename (see section C.1.1.2 for the full definition of HEVEA output basename).

In the gpic example, modifying user source cannot be totally avoided. However, writing in a generic style saves typing. For instance, users may define the following environment for centered gpic pictures in LaTeX:

```
\newenvironment{centergpic}{}{\begin{center}~\box\graph~\end{center}}
```

Source code will now be as follows:

```
\begin{centergpic}
.PS
ellipse "{\Large\bf Smile!}"
.PE
\end{centergpic}
```

HEVEA will process this source correctly, provided it is given its own definition for the centergpic environment beforehand:

```
\newenvironment{centergpic}
  {\begin{toimage}}
  {\box\graph\end{toimage}\begin{center}\imageflush\end{center}}
```

Assuming that the definition above is in a smile.hva file, the command sequence for translating smile.tex now is:

```
# gpic -t < smile.tex > tmp.tex
# hevea smile.hva tmp.tex -o smile.html
tmp.tex:5: Warning: ignoring definition of \centergpic
tmp.tex:5: Warning: not defining environment centergpic
# imagen smile
```

The warnings above are normal: they are issued when HEVEA runs across the LaTeX-intended definition of the centergpic environment and refuses to override its own definition for that environment.

7 Cutting your document into pieces with HACHA

HEVEA outputs a single `.html` file. This file can be cut into pieces at various sectional units by HACHA

7.1 Simple usage

First generate your HTML document by applying HEVEA:

```
# hevea mydoc.tex
```

Then cut *mydoc*`.html` into pieces by the command:

```
# hacha mydoc.html
```

This will generate a simple root file `index.html`. This root file holds document title, abstract and a simple table of contents. Every item in the table of contents contains a link to or into a file that holds a “cutting” sectional unit. By default, the cutting sectional unit is *section* in the *article* style and *chapter* in the *book* style.

Additionally, one level of sectioning below the cutting unit (i.e., subsections in the *article* style and sections in the *book* style) is shown as an entry in the table of contents. Sectional units above the cutting section (i.e., parts in both *article* and *book* styles) close the current table of contents and open a new one. Cross-references are properly handled, that is, the local links generated by HEVEA are changed into remote links.

The name of the root file can be changed using the `-o` option:

```
# hacha -o root.html mydoc.html
```

Some of HEVEA output get replicated in all the files generated by HACHA. Users can supply a header and a footer, which will appear at the begining and end of every page generated by HACHA. It suffices to include the following commands in the document preamble:

```
\htmlhead{header}
\htmlfoot{footer}
```

HACHA also makes every page it generates a clone of its input as regards attributes to the `<BODY ...>` opening tag and meta-information from the `<HEAD>`... `<\HEAD>` block. See section B.2 for examples of this replication feature.

7.2 Advanced usage

HACHA behavior can be altered from the document source, by using a counter and a few macros.

A document that explicitly includes cutting macros still can be typeset by LaTeX, provided it loads the `hevea.sty` style file from the HEVEA distribution. (See section 5 for details on this style file). An alternative to loading the `hevea` package is to put all cutting instructions in comments starting with `%HEVEA`.

7.2.1 Principle

HACHA recognizes all sectional units, ordered as follows, from top to bottom: *part*, *chapter*, *section*, *subsection*, *subsubection*, *paragraph* and *subparagraph*.

At any point between `\begin{document}` and `\end{document}`, there exist a current cutting sectional unit (cutting unit for short), a current cutting depth, a root file and an output file. Table of contents output goes to the root file, normal output goes to the output file. Cutting units start a new output file, whereas units comprised between the cutting unit and the cutting units plus the cutting depth add new entries in the table of contents.

At document start, the root file and the output file are HACHA output file (i.e., `index.html`). The cutting unit and the cutting depth are set to default values that depend on the document style.

7.2.2 Cutting macros

The following cutting instructions are for use in the document preamble. They command the cutting scheme of the whole document:

`\cuttingunit` This is a macro that holds the document cutting unit. You can change the default (which is *section* in the *article* style and *chapter* in the *book* style) by doing:

`\renewcommand{\cuttingunit}{`*secname*`}`.

`\tocnumber` Instruct HEVEA to put section numbers into table of content entries.

`\notocnumber` Instruct HEVEA *not* to put section numbers into table of content entries. This is the default.

`cuttingdepth` This is a counter that holds the document cutting depth. You can change the default value of 1 by doing `\setcounter{cuttingdepth}{`*numvalue*`}`. A cutting depth of zero means no other entries than the cutting units in the table of contents.

Other cutting instructions are to be used after `\begin{document}`. They all generate HTML comments in HEVEA output. These comments then act as instructions to HACHA.

`\cuthere{`*secname*`}{`*itemtitle*`}` Attempt a cut.

- If *secname* is the current cutting unit or the keyword "`now`", then a new output file is started and an entry in the current table of contents is generated, with title *itemtitle*. This entry holds a link to the new output file.
- If *secname* is above the cutting unit, then the current table of contents is closed. The output file is set to the current root file.
- If *secname* is below the cutting unit and less than the cutting depth away from it, then an entry is added in the table of contents. This entry contains *itemtitle* and a link to the point where `\cuthere` appears.
- Otherwise, no action is performed.

`\cutdef[`*depth*`]{`*secname*`}` Open a new table of contents, with cutting depth *depth* and cutting unit *secname*. If the optional *depth* is absent, the cutting depth does not change. The output file becomes the root file. Result is unspecified if whatever *secname* expands to is a sectional unit name above the current cutting unit, is not a valid sectional unit name or if *depth* does not expand to a small positive number.

`\cutend` End the current table of contents. This closes the scope of the previous `\cutdef`. The cutting unit and cutting depth are restored. Note that `\cutdef` and `\cutend` must be properly balanced.

Default settings work as follows: `\begin{document}` performs

```
\cutdef[\value{cuttingdepth}]{\cuttingunit}
```

and `\end{document}` performs `\cutend`. All sectioning commands perform `\cuthere`, with the sectional unit name as first argument and the (optional, if present) sectioning command argument (i.e., the section title) as second argument. Note that started versions of the sectioning commands also perform cutting instructions.

7.2.3 Examples

Consider, for instance, a *book* document with a long chapter that you want to cut at the section level, showing subsections:

```
\chapter{A long chapter}
.....

\chapter{The next chapter}
```

Then, you should insert a `\cutdef` at chapter start and a `\cutend` at chapter end:

```
\chapter{A long chapter}
%HEVEA\cutdef[1]{section}
.....
%HEVEA\cutend
\chapter{The next chapter}
```

Then, the file that would otherwise contain the long chapter now contains the chapter title and a table of sections. No other change is needed, since the macro `section` already performs the appropriate `\cuthere{section}{...}` commands, which were ignored by default. (Also note that cutting macros are placed inside `%HEVEA` comments, for LaTeX not to be disturbed).

The `\cuthere` macro can be used to put some document parts into their own file. This may prove appropriate for long cover pages or abstracts that would otherwise go into the root file. Consider the following document:

```
\documentclass{article}

\begin{document}

\begin{abstract} A big abstract \end{abstract}
...
```

Then, you make the abstract go to its own file as it was a cutting unit by typing:

```
\documentclass{article}
\usepackage{hevea}

\begin{document}
\cuthere{\cuttingunit}{Abstract}
\begin{abstract} A big abstract \end{abstract}
...
```

(Note that, this time, cutting macros appear unprotected in the source. However, LaTeX still can process the document, since the `hevea` package is loaded).

7.3 More Advanced Usage

In this section we show how to alter some details of HACHA behavior. This includes controling output file names and the title of generated web pages and introducing arbitrary cuts.

7.3.1 Controlling output file names

When invoked as `hacha` *doc*`.html`, HACHA produces a `index.html` table of links file that points into *doc*`001.html`, *doc*`002.html`, etc. content files. This is not very convenient when one wishes to point inside the document from outside. However, the `\cutname{`*name*`}` command sets the name of the current output file name as *name*.

Consider a document cut at the section level, which contains the following important section :

```
\section{Important section}\label{important}
...
```

To make the important section goes into file `important.html`, one writes :

```
\section{Important section}\label{important}\cutname{important.html}
...
```

Then, section "Important section" can be referenced from an HEVEA unaware HTML page by :

```
In this document, there is a very
<A HREF="important#important.html">important section</A>.
```

7.3.2 Controlling page titles

When HACHA creates a web page from a given sectional unit, the title of this page normally is the name of the sectional unit. For instance, the title of this very page should be "Cutting your document into pieces with HACHA". It is possible to insert some text at the beginning of all page titles, by using the `\htmlprefix` command. Hence, by writing `\htmlprefix{\hevea{} Manual: }` in the document, the title of this page would become : "HEVEA Manual: Cutting your document into pieces with HACHA" and the title of all other pages would show the same prefix.

7.3.3 Links for the root file

The command `\toplinks{`*prev*`}{`*up*`}{`*next*`}` instructs HACHA to put links to a "previous", "up" and "next" page in the root file. The following points are worth noticing:

- The `\toplink` command must appear in the document preamble (i.e., before `\begin{document}`).
- The arguments *prev*, *up* and *next* should expand to url's, notice that these argument are processed (see section 8.1.1).
- When one of the expected argument is left empty, the corresponding link is not generated.

This feature can prove useful to relate documents that are generated independantly by HEVEA and HACHA.

7.3.4 Cutting a document anywhere

Part of a document goes to a separate file whem enclosed in a `cutflow` environment :

`\begin{cutflow}{`*title*`}`. . . `\end{cutflow}`

The content ". . . " will go into a file of its own, while the argument *title* is used as the title of the introduced HTML page.

The HTML page introduced here does not belong to the normal flow of text. Consequently, one needs an explicit reference from the normal flow of text into the content of the `cutflow` environment. This will occur naturally when the content of the `cutflow` environment. contains a `\label` construct. This look natural in the following quiz example:

```
\paragraph{A small quiz}
\begin{enumerate}
\item What is black?
\item What is white?
\item What is Dylan?
\end{enumerate}
Answers in section~\ref{answers}.
\begin{cutflow}{Answers}
\paragraph{Quiz answers}\label{answers}
\begin{enumerate}
\item Black is black.
\item White is white.
\item Dylan is Dylan.
\end{enumerate}
\end{cutflow}
```

However,introducing HTML hyperlink targets and references with the `\aname` and `\ahrefloc` commands (see section 8.1.1) will be more practical most of the time.

8 Generating HTML constructs

HEVEA output language being HTML, it is normal for users to insert hypertext constructs their documents, or to control colors.

8.1 High-Level Commands

HEVEA provides high-level commands for doing this. Users are advised to use these macros in the first place, because it is easy to write incorrect HTML and that writting HTML directly may interfeer in nasty ways with HEVEA internals.

8.1.1 Commands for Hyperlinks

A few commands for hyperlink management and included images are provided, all these commands have approriate equivalents defined by the `hevea` package (see section 5.2). Hence, a document that relies on these high-level commands still can be typeset by LaTeX, provided it loads the `hevea` package.

Macro	HEVEA	LaTeX
`\ahref{`*url*`}{`*text*`}`	make *text* an hyperlink to *url*	echo *text*
`\footahref{`*url*`}{`*text*`}`	make *text* an hyperlink to *url*	make *url* a footnote to *text*, *url* is shown in typewriter font
`\ahrefurl{`*url*`}`	make *url* an hyperlink to *url*.	typeset *url* in typewriter font
`\ahrefloc{`*label*`}{`*text*`}`	make *text* an hyperlink to *label* inside the document	echo *text*
`\aname{`*label*`}{`*text*`}`	make *text* an hyperlink target with label *label*	echo *text*
`\mailto{`*address*`}`	make *address* a “mailto” link to *address*	typeset *address* in typewriter font
`\imgsrc[`*attr*`]{`*url*`}`	insert *url* as an image, *attr* are attributes in the HTML sense	do nothing
`\home{`*text*`}`	produce a home-dir url both for output and links, output aspect is: “~*text*”	

It is important to notice that all arguments are processed. For instance, to insert a link to my home page, (`http://pauillac.inria.fr/~maranget/index.html`), you should do something like this :

```
\ahref{http://pauillac.inria.fr/\home{maranget}/index.html}{his home page}
```

Given the frequency of `~`, `#` etc. in urls, this is annoying. Moreover, the immediate solution, using `\verb`, `\ahref{\verb" ... /~maranget/..."}{his home page}` does not work, since LaTeX forbids verbatim formatting inside command arguments.

Fortunately, the `url` package provides a very convenient `\url` command that acts like `\verb` and can appear in other command arguments (unfortunately, this is not the full story, see section B.17.7). Hence, provided the `url` package is loaded, a more convenient reformulation of the example above is :

```
\ahref{\url{http://pauillac.inria.fr/~maranget/index.html}}{his home page}
```

Or even better :

```
\urldef{\lucpage}{\url}{http://pauillac.inria.fr/~maranget/index.html}
\ahref{\lucpage}{his home page}
```

It may seem complicated, but this is a safe way to have a document processed both by LaTeX and HEVEA. Drawing a line between url typesetting and hyperlinks is correct, because users may sometime want urls to be processed and some other times not. Moreover, HEVEA (optionnaly) depends on only one third party package: `url`, which as correct as it can be and well-written.

In case the `\url` command is undefined at the time `\begin{document}` is processed, the commands `\url`, `\oneurl` and `\footurl` are defined as synonymous for `\ahref`, `\ahrefurl` and `\footahref`, thereby ensuring some compatibility with older versions of HEVEA. Note that this usage of `\url` is deprecated.

8.1.2 HTML style colors

Specifying colors both for LaTeX and HEVEA should be done using the `color` package (see section B.14.2). However,one can also specify text color using special type style declarations. The `hevea.sty` style file define no equivalent for these declarations, which therefore are for HEVEA consumption only.

Those declarations follow HTML conventions for colors. There are sixteen predefined colors:

> `\black`, `\silver`, `\gray`, `\white`, `\maroon`, `\red`, `\fuchsia`, `\purple`,
> `\green`, `\lime`, `\olive`, `\yellow`, `\navy`, `\blue`, `\teal`, `\aqua`

Additionally, the current text color can be changed by the declaration `\htmlcolor{`*number*`}`, where *number* is a six digit hexadecimal number specifying a color in the RGB space. For instance, the following declarations change font color to dark gray:

```
\htmlcolor{404040}
```

8.2 More on included images

The `\imgsrc` command becomes handy when one has images both in Postscript and GIF format. As explained in section 6.3, Postscript images can be included in LaTeX documents by using the `\epsfbox` command from the `epsf` package. For instance, if `screenshot.ps` is an encapsulated Postscript file, then a `doc.tex` document can include it by:

```
\epsfbox{screenshot.ps}
```

We may very well also have a GIF version of the screenshot image (or be able to produce one easily using image converting tools), let us store it in a `screenshot.ps.gif` file. Then, for HEVEA to include a link to the GIF image in its output, it suffices to define the `\epsfbox` command in the `macro.hva` file as follows:

```
\newcommand{\epsfbox}[1]{\imgsrc{#1.gif}}
```

Then HEVEA has to be run as:

```
# hevea macros.hva doc.tex
```

Since it has its own definition of `\epsfbox`, HEVEA will silently include a link the GIF image and not to the Postscript image.

If another naming scheme for image files is prefered, there are alternatives. For instance, assume that Postscript files are of the kind *name*`.ps`, while GIF files are of the kind *name*`.gif`. Then, images can be included using `\includeimage{`*name*`}`, where `\includeimage` is a specific user-defined command:

```
\newcommand{\includeimage}[1]{\ifhevea\imgsrc{#1.gif}\else\epsfbox{#1.ps}\fi}
```

Note that this method uses the `hevea` boolean register (see section 5.2.3). If one does not wish to load the `hevea.sty` file, one can adopt the slightly more verbose definition:

```
\newcommand{\includeimage}[1]{%
%HEVEA\imgsrc{#1.gif}%
%BEGIN LATEX
\epsfbox{#1.ps}
%END LATEX
}
```

When the Postscript file has been produced by translating a bitmap file, this simple method of making a GIF image and using the `\imgsrc` command is the most adequate. It should be prefered over using the more automated *image* file mechanism (see section 6), which will translate the image back from Postscript to bitmap format and will thus degrade it.

8.3 The rawhtml environment

Any text enclosed between \begin{rawhtml} and \end{rawhtml} is echoed verbatim into the HTML output file. Similarily, \rawhtmlinput{*file*} echoes the contents of file *file*.

For avoiding to break HTML element nesting, the rawhtml environment should be used only at toplevel (i.e. not within another environment), and it should contain only HTML text that makes sense alone. For instance, writing \begin{rawhtml}<TABLE><ALIGN=right>\end{rawhtml}... \begin{rawhtml}</TABLE>\end{rawhtml} is dangerous, because HEVEA is not informed about opening and closing the block-level element TABLE. In that case, one should use the internal macros \@open and \@close of the following section.

When HEVEA is given the command line option "-O", checking and optimization of text-level elements in the whole document takes place. As a consequence, incorrect HTML introduced by using the rawhtml environment may be detected at a later stage.

For the document to remain processable by LaTeX, it must load the hevea.sty style file (see section 5.2).

8.4 Internal macros

In this section a few of HEVEA internal macros are described. Internal macros occur at the final expansion stage of HEVEA and invoke Objective Caml code.

Normally, user source code should not use them, since their behavior may change from one version of HEVEA to another and because using them incorrectly easily crashes HEVEA. However:

- Internal macros are almost mandatory for writing supplementary base style files.
- Casual usage is a convenient (but dangerous) way to finely control output (cf. the examples in the next section).
- Knowing a little about internal macros helps in understanding how HEVEA works.

The general principle of HEVEA is that LaTeX environments \begin{*env*}... \end{*env*} get translated into HTML block-level elements <*block attributes*>... </*block*>. More specifically, such block level elements are opened by the internal macro \@open and closed by the internal macro \@close. As a special case, LaTeX groups {... } get translated into HTML *groups*, which are shadow block-level elements with neither opening tag nor closing tag.

It is important to notice that primitive arguments *are* processed (except for the \@print primitive, and for some of the basic style primitives). Thus, some characters cannot be given directly (e.g. # and % must be given as \# and \%).

\@print{*text*} Echo *text* verbatim.

\@getprint{*text*} Process *text* using a special output mode that strips off HTML tags. This macro is the one to use for processed attributes of HTML tags.

\@hr[*attr*]{*width*}{*height*} Ouput an HTML horizontal rule, *attr* is attributes given directly (e.g. SIZE=3 HOSHADE), while *width* and *height* are length arguments given in the LaTeX style (e.g. 2pt or .5\linewidth).

\@open{*BLOCK*}{*attributes*} Open HTML block-level element *BLOCK* with attributes *attributes*. The block name *BLOCK* *must* be uppercase. As a special case *BLOCK* may be the empty string, then a HTML *group* is opened.

\@close{*BLOCK*} Close HTML block-level element *BLOCK*. Note that \@open and \@close must be properly balanced.

Text-level elements are managed differently. They are not seen as blocks that must be closed explicitely and they are replaced by the internal text-level declarations \@style (and \@styleattr), \@fontsize and \@fontcolor. Block-level elements (and HTML groups) delimit the effect of such declarations.

`\@style{`*SHAPE*`}` Declare the text shape *SHAPE* (which must be uppercase) as active. Text shapes are known as font style elements (`I`, `TT`, etc.) or phrase elements (`EM`, etc.) in the HTML terminology, they are part of the more general class of text-level elements.

The text-level element *SHAPE* will get opened as soon as necessary and closed automatically, when the enclosing block-level elements get closed. Enclosed block-level elements are treated properly by closing *SHAPE* before them, and re-opening *SHAPE* inside them. The next text-level constructs exhibit similar behavior with respect to block-level elements.

`\@styleattr{`*NAME*`}{`*attr*`}` Declare the text-level element *NAME* with attribute *attr* active. This primitive behaves as `\@style`, except that the opennig tag has attributes. This primitive may prove useful for introducing `SPAN` elements. Note that neither argument is processed.

`\@fontsize{`*int*`}` Declare the text-level element `FONT` with attribute `SIZE=`*int* as active. Note that *int* must be a small integer in the range `1`,`2`, ... , `7`.

`\@fontcolor{`*color*`}` Declare the text-level element `FONT` with attribute `COLOR=`*color* as active. Note that *color* must be a color attribute value in the HTML style. That is either one of the sixteen conventional colors `black`, `silver` etc, or a RGB hexadecimal color specification of the form `"#`*XXXXXX*`"` (yes, quotes are needed). Note that the argument *color* is processed, as a consequence numerical color arguments should be given as `"\#`*XXXXXX*`"`.

`\@nostyle` Close active text-level declarations and ignore further text-level declarations. The effect stops when the enclosing block-level element is closed.

`\@clearstyle` Simply close active text-level declarations.

8.5 Examples

As a first example of using internal macros, consider the following excerpt from the `hevea.hva` file that defines the LaTeX `center` environment:

```
\newenvironment{center}{\@open{DIV}{ALIGN=center}}{\@close{DIV}}
```

Another example is the definition of the `\purple` color declaration (see section 8.1.2):

```
\newcommand{\purple}{\@fontcolor{purple}}
```

HEVEA does not feature all text-level elements by default. However one can easily use them with the internal macro `\@style`. For instance this is how you can make all emphasized text blink:

```
\renewcommand{\em}{\@style{EM}\@style{BLINK}}
```

Then, here is the definition of a simplified `\imgsrc` command (see section 8.1.1), without its optional argument:

```
\newcommand{\imgsrc}[1]
  {\@print{<IMG SRC="}\@getprint{#1}\@print{">}}
```

Here, `\@print` and `\@getprint` are used to output HTML text, depending upon wether this text requires processing or not. Note that `\@open{IMG}{SRC="#1"}` is not correct, because the element `IMG` consists in a single tag, without a closing tag.

Another interesting example is the definition of the command `\@doaelement`, which HEVEA uses internaly to output `A` elements.

```
\newcommand{\@doaelement}[2]
  {{\@nostyle\@print{<A }\@getprint{#1}\@print{>}}{#2}{\@nostyle\@print{</A>}}
```

The command `\@doaelement` takes two arguments: the first argument contains the opening tag attributes; while the second element is the textual content of the `A` element. By contrast with the `\imgsrc` exemple above, tags are emitted inside groups where styles are canceled by using the `\@nostyle` declaration. Such a complication is needed, so as to avoid breaking proper nesting of text-level elements.

Finally, here is an example of direct block opening. The `bgcolor` environment from the `color` package locally changes background color (see section B.14.2.1). This environment is defined as follows:

```
\newenvironment{bgcolor}[2][CELLPADDING=10]
  {\@open{TABLE}{#1}\@open{TR}{}\@open{TD}{BGCOLOR=\@getcolor{#2}}}
  {\@close{TD}\@close{TR}\@close{TABLE}}
```

The `bgcolor` environment operates by opening a HTML table (`TABLE`) with only one row (`TR`) and cell (`TD`) in its opening command, and closing all these elements in its closing command. In my opinion, such a style of opening block-level elements in environment opening commands and closing them in environment closing commands is good style.

Notice that, the `hevea` package provides a tentative definition of the `bgcolor` environment. As a consequence and provided the `hevea` package is loaded, documents that use the `bgcolor` environment remain processable by LaTeX.

The one cell background color is forced with a `BGCOLOR` attribute. Note that the mandatory argument to `\begin{bgcolor}` is the background color expressed as a high-level color, which therefore needs to be translated into a low-level color by using the `\@getcolor` internal macro from the `color` package. Additionally, `\begin{bgcolor}` takes HTML attributes as an optional argument. These attributes are the ones of the `TABLE` element.

9 Customizing HEVEA

HEVEA can be controlled by writing LaTeX code. In this section, we examine how users can change HEVEA default behavior or add functionalities. In all this section we assume that a document `mydoc.tex` is processed, using a private command file `macros.hva`. That is, HEVEA is invoked as:

```
# hevea macros.hva mydoc.tex
```

The general idea is as follows: one redefines LaTeX constructs in `macros.hva`, using internal commands. This requires a good working knowledge of both LaTeX and HTML. Usually, one can avoid internal commands, but then, all command redefinitions interact, sometimes in very nasty ways.

9.1 Simple changes

Users can easily change the rendering of some constructs. For instance, assume that *all* quotations in a text should be emphasized. Then, it suffices to put the following redeclaration in `macros.hva`:

```
\renewenvironment{quote}
  {\@open{BLOCKQUOTE}{}\@style{EM}}
  {\@close{BLOCKQUOTE}}
```

The same effect can be achieved without using any of the internal commands:

```
\let\oldquote\quote
\let\oldendquote\endquote
\renewenvironment{quote}{\oldquote\em}{\oldendquote}
```

In some sense, this second solution is easier, when one already knows how to customize LaTeX. However, this is less safe, since the definition of `\em` can be changed elsewhere.

9.2 Changing defaults for type-styles

HEVEA default rendering of type style changes is described in section B.15.1. For instance, the following example shows the default rendering for the font shapes:

```
\itshape italic shape \slshape slanted shape
\scshape small caps shape \upshape upright shape
```

By default, `\itshape` is italics, `\slshape` is maroon italics, `\scshape` is navy blue color and `\upshape` is no style at all. All shapes are mutually exclusive, this means that each shape declaration cancels the effect of other active shape declarations. For instance, in the example, small caps shapes is navy blue and not navy blue italics.

If one wishes to change the rendering of some of the shapes (say small caps), then one should redefine the old-style `\sc` declaration. For instance, to render small caps as bold fonts, one should redefine `\sc` by `\renewcommand{\sc}{\@style{B}}` in `macros.hva`.

Hence, redefining old-style declarations using internal commands should yield satisfactory output. However, since cancelation is done at the HTML level, a declaration belonging to one component may sometimes cancel the effect of another that belongs to another component. Anyway, you might have not noticed it if I had not told you.

9.3 Changing the interface of a command

Assume for instance that the base style of `mydoc.tex` is *jsc* (the *Journal of Symbolic Computation* style for articles). For running HEVEA, the *jsc* style can be replaced by *article* style, but for a few commands whose calling interface is changed. In particular, the `\title` command takes an extra optional argument (which HEVEA should ignore anyway). However, HEVEA can process the document as it stands. One solution to insert the following lines into `macros.hva`:

```
\input{article.hva}% Force document class ‘‘article’’
\let\oldtitle=\title
\renewcommand{\title}[2][]{\oldtitle{#2}}
```

The effect is to replace `\title` by a new command which calls HEVEA `\title` with the appropriate argument.

9.4 Checking the optional argument within a command

HEVEA fully implements LaTeX 2_ε `\newcommand`. That is, users can define commands with an optional argument. Such a feature permits to write a `\epsfbox` command that has the same interface as the LaTeX command and echoes itself as it is invoked to the *image* file. To do this, the HEVEA `\epsfbox` command has to check whether it is invoked with an optional argument or not. This can be achieved as follows :

```
\newcommand{\epsfbox}[2][!*!]{%
\ifthenelse{\equal{#1}{!*!}}
{\begin{toimage}\epsfbox{#2}\end{toimage}}%No optional argument
{\begin{toimage}\epsfbox[#1]{#2}\end{toimage}}}%With optional argument
\imageflush}
```

9.5 Changing the Format of Images

Semi-automatic generation of included images is described in section 6.

Links to included images are generated by the `\imageflush` command, which calls the `\imgsrc` command :

```
\newcommand{\imageflush}[1][]
{\@imageflush\stepcounter{image}\imgsrc[#1]{\jobname\theimage\heveaimageext}}
```

That is, you may supply a HTML-style attribute to the included image, as an optional argument to the `\imageflush` command.

By default, images are GIF images, stored in "`.gif`" files. HEVEA provides direct support for the alternative PNG image file format. It suffices to invoke `hevea` as:

`# hevea png.hva` *mydoc.tex*

Then imagen must be run as:

`# imagen -png` *mydoc*

Beware that transparent colors are not shown correctly by my browser for PNG images, while they work as advertised for GIF images. This does not harm as long as the final HTML document has a white background color (see Section B.2).

10 Other output formats

It is possible to translate LaTeX file into other formats than HTML. There are two such formats: plain text and info files. This enables producing postscript, HTML, plain text and info manuals from one (LaTeX) input file.

10.1 Text

The LaTeX file is processed and converted into a plain text formatted file. It allows some pretty-printing in plain text.

To translate into text, invoke HEVEA as follow:

```
# hevea -text [-w <width>] myfile.tex
```

Then, HEVEA produces `myfiles.txt` a plain text translation of `myfile.tex`.

Additionally, the optional argument `-w <number>` sets the width of the output for text formatting. By default, The text will be 72 characters wide.

Nearly every environments have been translated, included lists and tables. The support is nearly the same as in HTML, excepted in some cases described hereafter.

Most style changes are ignored, because it is hardly possible to render them in plain text. Thus, there are no italics, bold fonts, underlinings, nor size change or colors... The only exception is for the verbatim environment that puts the text inside quotes, to distinguish it more easily.

Tables with borders are rendered in the same spirit as in LaTeX. Thus for instance, it is possible to get vertical lines between some columns only. Table rendering can be poor in case of line overflow. The only way to correct this (apart from changing the tables themselves) is to adjust the formatting width, using the the `-w` command line option.

For now, maths are not supported at all in text mode. You can get very weird results with in-text mathematical formulas. Of course, simple expressions such as subscripts remains readable. For instance, x^2 will be rendered as `x^2`, but $\int_0^1 f(x)dx$ will yield something like : `int01f(x)dx`.

10.2 Info

The file format info is also supported. Info files are text files with limited hypertext links, they can be read by using *emacs* info mode or the `info` program. Please note that HEVEA translates plain LaTeX to info, and not TeXinfo.

You can translate your LaTeX files into info file(s) as follows:

```
# hevea -info [-w <width>] myfile.tex
```

Then, HEVEA produces the file `myfile.info`, an info translation of `myfile.tex`. However, if the resulting file is too large, it is cut into pieces automatically, and `myinfo.info` now contains references for all the nodes in the others files, which are named `myfile.info-1`, `myfile.info-2`,...

The optional argument `-w` has the same meaning as for text output.

The text will be organized in nodes that follow the pattern of LaTeX sectioning commands. Menus are created to navigate through the sections easily

A table of content is produced automatically. References, indexes and footnotes are supported, as they are in HTML mode. However, the info format only allows pointers to info nodes, i.e., in HEVEA case, to sectional units. As a consequence all cross references lead to sectional unit headers.

Part B
Reference manual

This part follows the pattern of the LaTeX reference manual [LaTeX, Appendix C].

B.1 Commands and Environments

B.1.1 Command Names and Arguments

LaTeX comments that start with "`%`" and end at end of line are ignored and produce no output. Usually, HEVEA ignore such comments. However, HEVEA processes text that follows "`%HEVEA`" and some other comments have a specific meaning to it (see section 5.3).

Command names follow strict LaTeX syntax. That is, apart from `#`, `$`, `~`, `_` and `^`, they either are "\" followed by a single non-letter character or "\" followed by a sequence of letters. Additionally, the letter sequence may be preceded by "`@`" (and this is the case of many of HEVEA internal commands), or terminated by "`*`" (starred variants are implemented as plain commands).

Users are strongly advised to follow strict LaTeX syntax for arguments. That is, mandatory arguments are enclosed in curly braces `{...}` and braces inside arguments must be properly balanced. Optional arguments are enclosed in square brackets `[...]`. However, HEVEA does its best to read arguments even when they are not enclosed in curly braces. Such arguments are a single, different from "\", "{" and " ", character or a command name. Thus, constructs such as "`\'ecole`", "`$a_1$`" or "`$a_\Gamma$`" are recognized and processed as "école" "a_1" and "a_Γ". By contrast, "`a^\mbox{...}`" is not recognized and must be written "`a^{\mbox{...}}`".

Also note that, by contrast with LaTeX, comments are parsed during argument scanning, as an important consequence brace nesting is also checked inside comments.

With respect to previous versions, HEVEA has been improved as regards emulation of complicated argument passing. That is, commands and their arguments can now appear in different static text bodies. As a consequence, HEVEA correctly processes the following source:

```
\newcommand{\boite}{\textbf}
\boite{In bold}
```

The definition of `\boite` makes it reduces as `\textbf` and HEVEA succeeds in fetching the argument "`{In bold}`". We get

In bold

The above example arguably is no "legal" LaTeX, but HEVEA handles it. Of course, there remains numerous "clever" LaTeX tricks that exploits TeX internal behavior, which HEVEA does not handle. For instance consider the following source:

```
\newcommand{\boite}[1]{\textbf#1}
\boite{{In bold}, Not in Bold.}
```

LaTeX typesets the text "In bold" using bold font, leaving the rest of the text alone. While HEVEA typesets everything using bold font. Here is LaTeX output:

In bold, Not in Bold.

Note that, in most similar situations, HEVEA will likely crash.

As a conclusion of this important section, Users are strongly advised to use ordinary command names and curly braces and not to think too much the TeX way.

B.1.2 Environments

Environment opening and closing is performed like in LaTeX, with `\begin{`*env*`}` and `\end{`*env*`}`. The `*`-form of an environment is a plain environment.

It is not advised to use `\`*env* and `\end`*env* in place of `\begin{`*env*`}` and `\end{`*env*`}`.

B.1.3 Fragile Commands

Fragile commands are not relevant to HEVEA and `\protect` is defined as a null command.

B.1.4 Declarations

Scope rules are the same as in LaTeX.

B.1.5 Invisible Commands

I am a bit lost here. However spaces in the output should correspond to users expectations. Note that, to HEVEA being invisible commands is a static property attached to command name.

B.1.6 The \\ Command

The `\\` and `\\*` commands are the same, they perform a line break, except inside arrays where they end the current row. Optional arguments to `\\` and `\\*` are ignored.

B.2 The Structure of the Document

Document structure is a bit simplified with respect to LaTeX, since documents consist of only two parts. The *preamble* starts as soon as HEVEA starts to operate and ends with the `\begin{document}` construct. Then, any input occurring before `\end{document}` is translated to HTML. However, the preamble is processed and the preamble comprises the content of the files given as command line arguments to HEVEA, see section C.1.1.1). As a consequence, command and environment definitions that occur before `\begin{document}` are performed. and they remain valid during all the processing.

In particular one can define a *header* and a *footer*, by using the `\htmlhead` and `\htmlfoot` commands in the preamble. Those commands register their argument as the header and the footer of the final HTML document. The header appears first while the footer appears last in (visible) HTML output. This is mostly useful when HEVEA output is later cut into pieces by HACHA, since both header and footer are replicated at the start and end of any file generated by HACHA. For instance, to append a copyright notice at the end of all the HTML pages, it suffices to invoke the `\htmlfoot` command as follows in the document preamble:

```
\htmlfoot{\copyright to me}
```

The `\htmlhead` command cannot be used for changing anything outside of the HTML document body, there are specific commands for doing this. One can change HEVEA default (empty) atribute for the opening `<BODY ...>` tag by redefining `\@bodyargs`. For instance, you get black text on a white background, when the following declaration occurs before `\begin{document}`:

```
\renewcommand{\@bodyargs}{TEXT=black BGCOLOR=white}
```

Similarly, some elements can be inserted into the output file `HEAD` element by redefining the `\@meta` command (Such elements typically are `META`, `LINK`, etc.). As such text is pure HTML, it should be included in a `rawhtml` environment. For instance, you can specify author information as follows:

```
\let\oldmeta=\@meta
\renewcommand{\@meta}{%
\oldmeta
\begin{rawhtml}
<META name="Author" content="Luc Maranget">
\end{rawhtml}}
```

Note how `\@meta` is first bound to `\oldmeta` before being redefined and how `\oldmeta` is invoked in the new definition of `\@meta`. Namely, simply overriding the old definition of `\@meta` would imply not outputting default meta-information.

The `\@charset` command holds the value of the document character set. By default, this value is `ISO-8859-1`. To change this value, there are basically two techniques.

- You can set the charset by extracting its value from the current locale environment. This operation is performed by a companion script: `xxcharset.exe`. It thus suffices to lauch HEVEA as:

  `# hevea -exec xxcharset.exe` *other arguments*

- You can more directly redefine `\@charset` in the document preamble. The suggested technique is to include the redefinition in a "`.hva`" file, loaded as a package.

Notice though, that just changing `\@charset` will not turn HEVEA into a multi-lingual tool.

B.3 Sentences and Paragraphs

B.3.1 Spacing

Generally speaking, spaces (and single newline characters) in the source are echoed in the output. Browser then manage with spaces and line-breaks. Following LaTeX behavior, spaces after commands are not echoed. Spaces after invisible commands with arguments are not echoed either.

However this is no longer true in math mode, see section B.7.5 on spaces in math mode.

B.3.2 Paragraphs

New paragraphs are introduced by one blank line or more. Paragraphs are not indented. Thus the macros `\indent` and `\noindent` perform no action.

B.3.3 Footnotes

The commands `\footnote`, `\footnotetext` and `\footnotemark` (with or without optional arguments) are supported. The `footnote` counter exists and (re)setting it or redefining `\thefootnote` should work properly. When footnotes are issued by a combination of `\footnotemark` and `\footnotetext`, a `\footnotemark` command must be issued first, otherwise some footnotes may get numbered incorrectly or disappear.

Footnotes appear at document end in the *article* style and at every chapter end in the *book* style. If the document is then cut into smaller files by HACHA (see section 7) footnotes may go to a separate file.

Footnotes are bad. If you want to suppress them, redefine `\footnote` as follows:

```
\renewcommand{\footnote}[2][]{}
```

If you want to put then in the text flow, redefine `\footnote` as follows:

```
\renewcommand{\footnote}[2][]{~(#2)}
```

B.3.4 Accents and special symbols

When there exists an equivalent to a given LaTeX symbol, using the iso-latin1 and symbol character sets, then HEVEA outputs such an equivalent. HTML pages that show these character sets can be found in the directory `http://pauillac.inria.fr/~maranget/hevea//doc/` at `iso.html` and `symbol.html`. Otherwise, HEVEA usually issues a warning to draw user attention. Users can then choose their own equivalent for the symbol.

Commands for making accents used in non-English languages, such as `\'`, work when then produce letters from the iso-latin1 character set. Otherwise, the argument to the command is not modified (no warning here). However, it is more simple to write the document using iso-latin1. LaTeX can process such documents by loading the package `isolatin1`.

B.4 Sectioning

B.4.1 Sectioning commands

Sectioning commands from `\part` down to `\subparagraph` are defined in base style files. They accept an optional argument and have starred versions.

The non-starred sectioning commands from `\part` down to `\subsubsection` show section numbers in sectional unit headings, provided their *level* is greater than or equal to the current value of the `secnumdepth` counter. Sectional unit levels and the default value of the `secnumdepth` counter are the same as in LaTeX. Furthermore, given a sectional unit *secname*, the counter *secname* exists and the appearance of sectional units numbers can be changed by redefining `\the`*secname*. For instance, the following redefinition turn the numbering of chapters into alphabetic (uppercase) style:

```
\renewcommand{\thechapter}{\Alph{chapter}}
```

When jumping to anchors, browsers put the targeted line on top of display. As a consequence, in the following code:

```
\section{A section}
\label{section:section}
 ...
See Section~\ref{section:section}
```

Clicking on the link produced by `\ref{section:section}` will result in *not* displaying the targeted section title. A fix is writting:

```
\section{\label{section:section}A section}
 ...
See Section~\ref{section:section}
```

Note that `\label` should not be be placed last in section title (and I do not know the reason why). Have a try for this section B.4.1!

B.4.2 The Appendix

The `\appendix` command exists and should work as in LaTeX.

B.4.3 Table of Contents

HEVEA now generates a table of contents, using a procedure similar to the one of LaTeX(a "`.htoc`" file is involved). One inserts this table of contents in the main document by issuing the command `\tableofcontents`. Table of contents is controled by the counter `tocdepth`. By default, the table of contents shows sectionning units down to the subsubsection level in *article* style and down to the subsection level in *book* (or *report*) style. To include more or less sectionning units in the table of contents, one sould increase or decrease the `tocdepth` counter. It is important to notice that HEVEA produces such a table of contents, only when it has total control over cross-references. More precisely, HEVEA cannot produce the table of contents whem it reads LaTeX-produced `.aux` files. Instead, it should read its own `.haux` files. This will naturally occur if no `.aux` files are present, otherwise these `.aux` files should be deleted, or HEVEA should be instructed not to read them with the command-line option "`-fix`" (see Sections B.11.1 and C.1.1.4).

One can also add extra entries in the table of contents by using the command `\addcontentslines`, in a way similar to LaTeX homonymous command. However, hyperlinks need to be introduced explicitely, as in the following example, where an anchor is defined in the section title and refered to in the argument to `\addcontentsline` :

```
\subsection*{\aname{no:number}{Use \hacha{}}}
\addcontentsline{toc}{subsection}{\ahrefloc{no:number}{Use \hacha{}}}
```

(See Section 8.1.1 for details on commands related to hyperlinks.)

There is no list of figures nor list of tables.

Use HACHA

However, HEVEA has a more sophisticated way of producing a kind of map w.r.t. the sectioning of the document. A later run of HACHA on HEVEA output file splits it in smaller files organized in a tree whose nodes are tables of links. By contrast with LaTeX, starred sectioning commands generate entries in these tables of contents. Table of contents entries hold the optional argument to sectioning commands or their argument when there is no optional argument. Section 7 explains how to control HACHA.

B.5 Classes, Packages and Page Styles

B.5.1 Document Class

Both LaTeX 2_ε `\documentclass` and old LaTeX `\documentstyle` are accepted. Their argument *style* is interpreted by attempting to load a *style*`.hva` file (see C.1.1.1 to see where HEVEA searches for files). Presently, only the style files `article.hva`, `seminar.hva`, `book.hva` and `report.hva` exist, the latter two being equivalent.

If one of the reckognized styles has already been loaded at the time when `\documentclass` or `\documentstyle` is executed, then no attempt to load a style file is made. This allows to override the document style file by giving one of the four recognized style files of HEVEA as command line arguments (see section 2.2).

Conversely, if HEVEA attempt to load *style*`.hva` fails, then a fatal error is flagged, since it can be sure that the document cannot be processed.

B.5.2 Packages and Page Styles

HEVEA reacts to `\usepackage[`*options*`]{`*pkg*`}` in the following way:

1. The whole `\usepackage` command with its arguments gets echoed to the *image* file (see 6).
2. HEVEA attempt to load file *pkg*`.hva`, (see section C.1.1.1 on where HEVEA searches for files).

Note that HEVEA will not fail if it cannot load *pkg*`.hva` and that no warning is issued in that case.

The HEVEA distribution contains implementations of some packages, such as `verbatim`, `colors`, `graphics`, etc.

In some situations it may not hurt at all if HEVEA does not implement a package, for instance HEVEA does not provide an implementation for the packages `isolatin1` or `fullpage`...

Users needing an implementation of a package that is widely used and available are encouraged to contact the author. Experienced users may find it fun to attempt to write package implementations by themselves.

B.5.3 The Title Page and Abstract

All title related commands exist, with the following peculiarities:

- The `\title` command must appear in the preamble for the title to appear in HTML document header.
- When not present the date is left empty. The `\today` command generates will work properly only if `hevea` is invoked with the `-exec xxdate.exe` option. Otherwise `\today` generates nothing and a warning is issued.

The `abstract` environment is present is all base styles, including the *book* style. The `titlepage` environment does nothing.

B.6 Displayed Paragraphs

Displayed-paragraph environments translate to block-level elements.

In addition to the environments described in this section, HEVEA implements the `center`, `flushleft` and `flushright` environments. HEVEA also implements the corespondant TEX style declaration `\centering` `\raggedright` and `\raggedleft`, but these declarations may not work as expected, when they do not appear directly inside a displayed-paragraph environment or inside an array element.

B.6.1 Quotation and Verse

The `quote` and `quotation` environments are the same thing: they translate to `BLOCKQUOTE` elements. The `verse` environment is not supported.

B.6.2 List-Making environments

The `itemize`, `enumerate` and `description` environments translate to the `UL`, `OL`, and `DL` elements and this is the whole story.

As a consequence, no control is allowed on the appearances of these environments. More precisely optional arguments to `\item` do not function properly inside `itemize` and `enumerate`. Moreover, item labels inside `itemize` or numbering style inside `enumerate` are browser dependent.

However, customized lists can be produced by using the the `list` environment (see next section).

B.6.3 The `list` and `trivlist` environments

The `list` environment translates to the `DL` element. Arguments to `\begin{list}` are handled as follows:

`\begin{list}`{*default_label*}{*decls*}

The first argument *default_label* is the label generated by an `\item` command with no argument. The second argument, *decls* is a sequence of declarations. In practice, the following declarations are relevant:

`\usecounter`{*counter*} The counter *counter* is incremented by `\refstepcounter` by every `\item` command with no argument, before it does anything else.

`\renewcommand{\makelabel}[1]{...}` The command `\item` executes `\makelabel{`*label*`}`, where *label* is the item label, to print its label. Thus, users can change label formatting by redefining `\makelabel`. The default definition of `\makelabel` simply echoes *label*.

As an example, a list with an user-defined counter can be defined as follows:

```
\newcounter{coucou}
\begin{list}{\thecoucou}{%
\usecounter{coucou}%
\renewcommand{\makelabel}[1]{\textbf{#1}.}}
...
\end{list}
```

This yields:

1. First item.

2. Second item.

The `trivlist` environment is also supported. It is equivalent to the `description` environment.

B.6.4 Verbatim

The `verbatim` and `verbatim*` environments translate to the `PRE` element. Inside `verbatim*`, spaces are replaced by underscores ("␣").

Similarly, `\verb` and `\verb*` translate to the `CODE` text element.

The `alltt` environment is supported.

B.7 Mathematical Formulas

B.7.1 Math Mode Environment

The three ways to use math mode (`$`...`$`, `\(`...`\)` and `\begin{math}`...`\end{math}`) are supported. The three ways to use display math mode (`$$`...`$$`, `\[`...`\]` and `\begin{displaymath}`... `\end{displaymath}`) are also supported. Furthermore, `\ensuremath` behaves as expected.

The `equation`, `eqnarray`, `eqnarray*` environments are supported. Equation labeling and numbering is performed in the first two environments, using the `equation` counter. Additionally, numbering can be suppressed in one row of an `eqnarray`, using the `\nonumber` command.

Math mode is not as powerful in HEVEA as in LaTeX. The limitations of math mode can often be surpassed by using math display mode. As a matter of fact, math mode is for in-text formulas. From the HTML point of view, this means that math mode does not close the current flow of text and that formulas in math mode must be rendered using text-level elements only. By contrast, displayed formulas can be rendered using block-level elements. This means that HEVEA have much more possibilities in display context than inside normal flow of text. In particular, stacking text elements one above the over is possible only in display context.

B.7.2 Common Structures

HEVEA admits, subscript (`_`), superscripts (`^`) and fractions (`\frac{`*numer*`}{`*denom*`}`). The best effect is obtained in display mode, where HTML `TABLE` element is extensively used. By contrast, when not in display mode, HEVEA uses only `SUB` and `SUP` text-level elements to render superscrits and subscript, and the result may not be very satisfying.

However, simple subscripts and superscripts, such as `x_i` or `x^2`, are always rendered using the `SUB` and `SUP` text-level elements and their appearance should be correct even in in-text formulas.

When occurring outside math mode, characters `_` and `^` act as ordinary characters and get echoed to the output. However, a warning is issued.

The n^{th} root command `\sqrt` is not supported. The "root" symbol is not necessary, thanks to fractional exponents. For instance, the `\sqrt` command can be defined as follows:

```
\newcommand{\sqrt}[2][2]{\left(#2\right)^{1/#1}}
```

An attempt is made to render all ellipsis constructs (`\ldots`, `\cdots`, `\vdots` and `\ddots`). The effect may be strange for the latter two.

B.7.3 Mathematical symbols

Symbols that can be printed using browser iso-latin1 or symbol fonts are translated. Other symbols are undefined most of the time. Attempting to translate them will thus generate "Unknown macro" warnings. Then, users can choose their own replacement for these symbols. These personal definitions are best placed in an ad-hoc style file, given as a command line argument to HEVEA. A suggested replacement is a mix of colors and available symbols.

When given the `-nosymb` option, HEVEA silently replaces symbols that cannot be rendered by iso-latin1 only by text equivalents. These equivalents are English words by default, or French words when the `-francais` option is set.

Log-like functions and variable sized-symbols are recognized and their subscripts and superscripts are put where they should in display mode. Subscript and superscript placement can be changed using the `\limits` and `\nolimits` commands. Big delimiters are also handled.

B.7.4 Putting one thing above the other

The commands `\stackrel`, `\underline` and `\overline` are recognized. They produce sensible output in display mode. In text mode, these macros call the `\textstackrel`, `\textunderline` and `\textoverline` macros. These macros perform the following default actions, which can be changed by redefining them:

`\textstackrel` Performs ordinary superscripting.

`\textunderline` Underlines its argument, using the `U` text-level element.

`\textoverline` Sends a warning message to the console and echoes its argument in the output.

Math accents (`\hat`, `\tilde`, etc.) are not handled by default. However, the distribution includes a `mathaccents.hva` file that provides definitions for almost all math accents commands, except `\check` and `\breve`. Rendering is far from perfect and changes from display to text mode. More precisely, the accent is put (too far) above the symbol in display mode, and as an ordinary superscript in text mode.

If such a rendering is considered too ugly, one should not load the `mathaccents.hva` file and write alternative definitions. For instance, the following custom definitions issue color changes:

```
\newcommand{\tilde}[1]{{\blue#1}}
\newcommand{\vec}[1]{{\red#1}}
```

Of course, such a trick probably requires looking closely at HTML output to check whether the document is still understandable or not. It may be better to stay with a poorly formatted document that remains closer to universally understood notations for mathematics.

B.7.5 Spacing

By contrast with LaTeX, space in the input matters in math mode. One or more spaces are translated to one space. Furthermore, spaces after commands (such as `\alpha`) are echoed except for invisible commands (such as `\tt`). This allows users to control space in their formulas, output being near to what can be expected.

Explicit spacing commands (`\,`, `\!`, `\:` and `\;`) are recognized, the first two commands do nothing, while the others two output one space.

B.7.6 Changing Style

Letters are italicized inside math mode and this cannot be changed. The appearance of other symbols can be changed using LaTeX 2_ε style changing commands (`\mathbf`, etc.). The commands `\boldmath` and

`\unboldmath` are not recognized. Whether symbols belonging to the symbol font are affected by style changes or not is browser dependent.

The `\cal` declaration and the `\mathcal` command (that yield calligraphic letters in LaTeX) exist. They yield red letters by default.

Observe that this does not corresponds directly to how LaTeX manage style in math mode and that, in fact, style cannot really change in math mode.

Math style changing declarations `\displaystyle` and `\textstyle` do nothing when HEVEA is already in the requested mode, otherwise they issue a warning. This is so because HEVEA implements displayed maths as tables, which require to be both opened and closed and introduce line breaks in the output. As a consequence, warnings on `\displaystyle` are to be taken seriously.

The commands `\scriptstyle` and `\scriptscriptstyle` perform type size changes.

B.8 Definitions, Numbering

B.8.1 Defining Commands

HEVEA understands command definitions given in LaTeX style. Such definitions are made using `\newcommand`, `\renewcommand` and `\providecommand`. These three constructs accept the same arguments and have the same meaning as in LaTeX, in particular it is possible to define an user command with one optional argument. However, HEVEA is more tolerant: if command *name* already exists, then a subsequent `\newcommand{`*name*`}`... is ignored. If macro *name* does not exists, then `\renewcommand{`*name*`}`... performs a definition of *name*. In both cases, LaTeX would crash, HEVEA just issues warnings.

The behavior of `\newcommand` allows to shadow document definition, provided the new definitions are processed before the document definitions. This is easily done by grouping the shadowing definition in a specific style file given as an argument to HEVEA (see section 5.1). Conversely, changes of base macros (i.e., the ones that HEVEA defines before loading any user-specified file) must be performed using `\renewcommand`.

Scoping rules apply to macros, as they do in LaTeX. Environments and groups define a scope and command definition are local to the scope they occur.

It is worth noticing that HEVEA also partly implements TeX definitions (using `\def`) and bindings (using `\let`), see section B.16.2 for details.

B.8.2 Defining Environments

HEVEA accepts environment definitions and redefinitions by `\newenvironment` and `\renewenvironment`. The support is complete and should conform to [LaTeX, Sections C.8.2].

Environments define a scope both for commands and environment definitions.

B.8.3 Theorem-like Environments

New theorem-like environments can also be introduced and redefined, using `\newtheorem` and `\renewtheorem`.

Note that, by contrast with plain environments definitions, theorem-like environment definitions are global definitions.

B.8.4 Numbering

LaTeX counters are (fully ?) supported. In particular, defining a counter *cmd* with `\newcounter{`*cmd*`}` creates a macro `\the`*cmd* that outputs the counter value. Then the `\the`*cmd* command can be redefined. For instance, section numbering can be turned into alphabetic style by:

```
\renewcommand{\thesection}{\alph{section}}
```

Note that TeX style for counters is not supported at all and that using this style will clobber the output. However, HEVEA implements the *calc* package that makes using TeX style for counters useless in most situations (see section B.17.3).

B.8.5 The `ifthen` Package

The `ifthen` package is partially supported. The one unsupported construct is the `\lengthtest` test expression, which is undefined.

As a consequence, HEVEA accepts the following example from the LaTeX manual:

```
\newcounter{ca}\newcounter{cb}%
\newcommand{\printgcd}[2]{%
  \setcounter{ca}{#1}\setcounter{cb}{#2}%
  Gcd(#1,#2) =
  \whiledo{\not\(\value{ca}= \value{cb}\)}%
    {\ifthenelse{\value{ca}>\value{cb}}%
      {\addtocounter{ca}{-\value{cb}}}%
      {\addtocounter{cb}{-\value{ca}}}%
    gcd(\arabic{ca}, \arabic{cb}) = }%
  \arabic{ca}.}%
For example: \printgcd{54}{30}
```

For example: Gcd(54,30) = gcd(24, 30) = gcd(24, 6) = gcd(18, 6) = gcd(12, 6) = gcd(6, 6) = 6.

Additionally, a few boolean registers are defined by HEVEA. Some of them are of interest to users.

`hevea` Initial value is `true`. The `hevea.sty` style file also defines this register with initial value *false*.

`mmode` This register value reflects HEVEA operating mode, it is *true* in math-mode and *false* otherwise.

`display` This register value reflects HEVEA operating mode, it is *true* in display-mode and *false* otherwise.

`french` This register value reflects the `-french` command line option internally (see Section C.1.1.4).

`footer` Initial value is `true`. When set false, HEVEA does not insert its footer "*This document has been translated by HEVEA*".

Finally, note that HEVEA also recognized à la TeX conditional macros (see section B.16.2.4). Such macros are fully compatible with the boolean registers of the `ifthen` package, as it is the case in LaTeX.

B.9 Figures and Other Floating Bodies

Figures and tables are put where they appear in source, regardless of their placement arguments. They are outputed inside a `BLOCKQUOTE` element and they are separated from enclosing text by two horizontal rules.

Captions and cross referencing are handled. However captions are not moved at end of figures: instead, they appear where the `\caption` commands occur in source code. The `\suppressfloats` command does nothing and the figure related counters (such as `topnumber`) exist but are useless.

Marginal notes are not handled and the `\marginpar` command does not exist. If their document holds `\marginpar` command, users should probably define it as a null command:

```
\newcommand{\marginpar}[1]{}
```

B.10 Lining It Up in Columns

B.10.1 The `tabbing` Environment

Limited support is offered. The `tabbing` environment translate to a flexible `tabular`-like environment. Inside this environment, the command `\kill` ends a row, while commands `\=` and `\>` start a new column. All other tabbing commands do not even exist.

B.10.2 The array and tabular environments

These environments are supported, using HTML TABLE element, rendering is satisfactory in most (not too complicated) cases. By contrast with LaTeX, some of the array items always are typeset in display mode. Whether an array item is typeset in display mode or not depends upon its column specification, the `l`, `c` and `r` specifications open display mode while the remaining `p` and `@` do not. The `l`, `c`,`r` and `@` specifications disable word wrap, while the `p` specification enables it.

Entries in a column whose specification is `l` (resp. `c` or `r`) get left-aligned (resp. centered or right-aligned) in the horizontal direction. They will get top-aligned in the vertical direction if there are other column specifications in the same array that specify vertical alignement constraints (such as "`p{`*wd*`}`", see below). Otherwise, vertical alignement is unspecified.

Entries in a column whose specification is `p{`*wd*`}` get left-aligned in the horizontal direction and top-aligned in the vertical direction and a paragraph break reduces to one line break inside them. This is the only occasion where HEVEA makes a distinction between LR-mode and paragraph mode. Also observe that the length argument *wd* to the `p` specification is ignored.

Some LaTeX array features are not supported at all:

- Optional arguments to `\begin{array}` and `\begin{tabular}` are ignored.
- The command `\vline` does not exists.

Some others are partly rendered:

- Spacing between columns is different.
- `@` formatting specifications in `\multicolumn` argument are ignored.
- If a `|` appears somewhere in the column formatting specification, then the array is shown with borders.
- The command `\hline` does nothing if the array has borders (see above). Otherwise, an horizontal rule is outputed.
- The command `\cline` ignores its argument and is equivalent to `\hline`.
- Similarly the command `\extracolsep` issues a warning and ignores its argument.

Additionally, the `tabular*` environment is recognized and gets renderered as an HTML table with an advisory width attribute.

By default, HEVEA implements the `array` package (see [LaTeX-bis, Section 5.3] and section B.17.2 in this document), which significantly extends the `array` and `tabular` environments.

B.11 Moving Information Around

B.11.1 Files

In some situations, HEVEA uses some of the ancillary files generated by LaTeX. More precisely, while processing file *mydoc*`.tex`, the following files may be read:

`.aux` The file *mydoc*`.aux` contains cross-referencing information, such as figure or section numbers. If this file is present, HEVEA reads it and put such numbers (or labels) inside the links generated by the `\ref` command. If the `.aux` file is not present, or if the `hevea` command is given the "`-fix`" option, HEVEA will instead use `.haux` files (see below).

`.haux` Such files are HEVEA equivalents of `.aux` files. Indeed, they are simplified `.aux` files. As a consequence, two runs of HEVEA might be needed to get cross references right.

`.htoc` This file contains a formatted table of contents. It is produces while reading the `.haux` file. As consequence a table of contents is available only when the `.haux` file is read.

`.bbl` The file *mydoc*`.bbl` is generated by BibTeX. It is read by the `\bibliography` command.

`.hidx` **and** `.hind` HEVEA computes its own indexes, using `.hidx` files for storing index references and, using `.hind` files for storing formatted indexes. Index formatting significantly departs from the one of LaTeX. Again, several runs of HEVEA might be needed to get indexes right.

HEVEA does not fail when it cannot find an auxiliary file. When another run of HEVEA is needed, a warning is issued, and it is user's responsability to rerun HEVEA. However, using the convenient `-fix` command line option is provided makes HEVEA rerun itself.

B.11.2 Cross-References

The LaTeX `\label` and `\ref` are changed by HEVEA into HTML anchors and local links. Spaces in the arguments to these commands are better avoided.

Additionally, numerical references to sectional units, figures, tables, etc. are shown, as they would appear in the `.dvi` file. Numerical references to pages (such as generated by `\pageref`) are not shown; only an link is generated.

While processing a document *mydoc*`.tex`, cross-referencing information can be computed in two different, mutually exclusive, ways, depending on whether LaTeX has been previously run or not:

- If there exists a file *mydoc*`.aux`, then cross-referencing information is extracted from that file. Of course, the *mydoc*`.aux` file has to be up-to-date, that is, it should be generated by running LaTeX as many times as necessary. (For HEVEA needs, one run is probably sufficient).
- If no *mydoc*`.aux` file exists, then HEVEA expect to find cross-referencing information in the file *mydoc*`.haux`.

When using its own *mydoc*`.haux` file, HEVEA will output a new *mydoc*`.haux` file at the end of its processing. This new *mydoc*`.haux` file contains actualized cross referencing information. Hence, in that case, HEVEA may need to run twice to get cross-references right. Note that, just like LaTeX, HEVEA issues a warning then the cross-referencing information it generates differs from what it has read at start-up, and that it does not fail if *mydoc*`.haux` does not exist.

Observe that if a non-correct *mydoc*`.aux` file is present, then cross-references will apparently be wrong. However the links are correct.

B.11.3 Bibliography and Citations

The `\cite` macro is supported. Its optional argument is correctly handled. Citation labels are extracted from the `.aux` file if present, from the `.haux` file otherwise. Note that these labels are put there by LaTeX in the first case, and by HEVEA in the second case, when they process the `\bibitem` command.

The `\bibliography` command is recognized, it loads the `.bbl` file which should thus have been generated before, using the appropriate combination of LaTeX and BibTeX runs.

The `thebibliography` environment is recognized.

The `\nocite` and `\bibliographystyle` macros exist and do nothing.

B.11.4 Splitting the Input

The `\input` and `\include` commands exist and they perform exactly the same operation of searching (and then processing) a file, whose name is given as an argument. See section C.1.1.1 on how HEVEA searches files. However, in the case of the `\include` command, the file is searched only when previously given as an argument to the `\includeonly` command.

Note the following features:

- TeX syntax for `\input` is not supported. That is, one should write `\input{`*filename*`}`.

- If *filename* is excluded with the `-e` command line option (see section C.1.1.4), then HEVEA does not attempt to load *filename*. Instead, it echoes `\input{`*filename*`}` and `\include{`*filename*`}` commands into the *image* file. This sounds complicated, but this is what you want!
- HEVEA does not fails when it cannot find a file, it just issues a warning.

The `\listfiles` command is a null command.

B.11.5 Index and Glossary

Glossaries are not handled (who uses them ?).

While processing a document *mydoc*`.tex`, index entries go into the file *mydoc*`.hidx`, while the formatted index gets written into the file *mydoc*`.hind`. As with LaTeX, two runs of HEVEA are normally needed to format the index. However, if all index producing commands (normally `\index`) occur before the index formatting command (normally `\printindex`), then only one run is needed.

As in LaTeX, index processing is not enabled by default and some package has to be loaded explicitely in the document preamble. To that aim, HEVEA provides the standard package `makeidx`, and two extended packages that allow the production of several indexes (see section B.17.5).

Formatting of indexes in HEVEA departs from LaTeX behavior. More precisely the `theindex` environment does not exist. Instead, indexes are formatted using special `indexenv` environments. Those details do not normally concern users. However, the number of columns in the presentation of the index can be controled by setting the value of the `indexcols` counter (default value is two).

B.11.6 Terminal Input and Output

The `\typeout` command echos its argument on the terminal, macro parameter `#`*i* are replaced by their values. The `\typein` command is not supported.

B.12 Line and Page Breaking

B.12.1 Line Breaking

The advisory line breaking command `\linebreak` will produce a line break if it has no argument or if its optional argument is 4. The `\nolinebreak` command is a null command.

The `\\` and `\\*` commands output a `<BR>` tag, except inside arrays where the close the current row. Their optional argument is ignored. The `\newline` command outputs a `<BR>` tag.

All other line breaking commands, declarations or environments are silently ignored.

B.12.2 Page Breaking

They are no pages in the physical sense in HTML. Thus, all these commands are ignored.

B.13 Lengths, Spaces and Boxes

B.13.1 Length

All length commands are ignored, things go smoothly when LaTeX syntax is used (using the `\newlength`, `\setlength`, etc. commands, which are null macros). Of course, if lengths are really important to the document, rendering will be poor.

Note that TeX length syntax is not at all recognized. As a consequence, writing things like `\textwidth=10cm` will clobber the output. Users can correct such misbehavior by adopting LaTeX syntax, here they should write `\setlength{\textwidth}{10cm}`.

B.13.2 Space

The `\hspace`, `\vspace` and `\addvspace` spacing commands and their starred versions recognize positive explicit length arguments. Such arguments get converted to a number of non-breaking spaces or line breaks. Basically, the value of `1em` or `1ex` is one space or one line-break. For other length units, a simple conversion based upon a 10pt font is used.

HEVEA cannot interpret more complicated length arguments or perform negative spacing. In these situations, a warning is issued and no output is done.

Spacing commands without arguments are recognized. The `\enspace`, `\quad` and `\qquad` commands output one, two and four non-breaking spaces, while the `\smallskip`, `\medskip` and `\bigskip` output one, one, and two line breaks.

Stretchable lengths do not exist, thus the `\hfill` and `\vfill` macros are undefined.

B.13.3 Boxes

Box contents is typeset in text mode (i.e., non-math and non-display mode). Both LaTeX boxing commands `\mbox` and `\makebox` commands exist. However `\makebox` generates a specific warning, since HEVEA ignore the length and positioning instructions given as optional argument.

Similarly, the boxing with frame `\fbox` and `\framebox` commands are recognized and `\framebox` issues a warning. When in display mode, `\fbox` frames its argument by enclosing it in a table with borders. Otherwise, `\fbox` calls the `\textfbox` command, which issues a warning and typesets its argument inside a `\mbox` (and thus no frame is drawn). Users can alter the behavior of `\fbox` in non-display mode by redefining `\textfbox`.

Boxes can be saved for latter usage by storing them in *bins*. New bins are defined by `\newsavebox{`*cmd*`}`.

Then some text can be saved into *cmd* by `\sbox{`*cmd*`}{`*text*`}` or `\begin{lrbox}{`*cmd*`}` *text* `\end{lrbox}`. The text is translated to HTML, as if it was inside a `\mbox` and the resulting output is stored. It is retrieved (and outputed) by the command `\usebox{`*cmd*`}`. The `\savebox` command reduces to `\sbox`, ignoring its optional arguments.

The `\rule` commands translate to a HTML horizontal rule (`<HR>`) regardless of its arguments.

All other box-related commands do not exist.

B.14 Pictures and Colors

B.14.1 The `picture` environment and the `graphics` Package

It is possible to have pictures and graphics processed by `imagen` (see section 6.1). In the case of the `picture` environment it remains users responsability to explicitly choose source chunks that will get rendered as GIF images. In the case of the commands from the `graphics` package, this choice is made by HEVEA. In both cases, the `imagen` script has to be run by hand. (However, note that HEVEA runs `imagen` when given the `-fix` command-line option.)

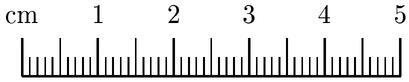
For instance consider the following picture:

```
\newcounter{cms}
\setlength{\unitlength}{1mm}
\begin{picture}(50,10)
\put(0,7){\makebox(0,0)[b]{cm}}
\multiput(10,7)(10,0){5}{\addtocounter{cms}{1}\makebox(0,0)[b]{\arabic{cms}}}
\multiput(1,0)(1,0){49}{\line(0,1){2.5}}
\multiput(5,0)(10,0){5}{\line(0,1){5}}
\thicklines
\put(0,0){\line(1,0){50}}
\multiput(0,0)(10,0){6}{\line(0,1){5}}
\end{picture}
```

Users should enclose *all* picture elements in a `toimage` environment (or inside `%BEGIN IMAGE`... `%END IMAGE` comments) and insert an `\imageflush` command, where they want the image to appear in HTML output:

```
%BEGIN IMAGE
\newcounter{cms}
\setlength{\unitlength}{1mm}
\begin{picture}(50,10)
  ...
\end{picture}
%END IMAGE
%HEVEA\imageflush
```

This will result in normal processing by LaTeX and image inclusion by HEVEA:



All commands from the graphics package are implemented using the automatic image inclusion feature. More precisely, the outermost invocations of the `\includegraphics`, `\scalebox`, etc. commands are sent to the image *image* file and there will be one GIF image per outermost invocation of these commands.

For instance, consider a document `doc.tex` that loads the `graphics` package and that includes some (scaled) images by:

```
\begin{center}
\scalebox{.5}{\includegraphics{round.ps}}
\scalebox{.75}{\includegraphics{round.ps}}
\includegraphics{round.ps}
\end{center}
```

Then, issuing the following two commands:

```
# hevea doc.tex
# imagen doc
```

yields HTML that basically consists in three image links, the images being generated by `imagen`.

B.14.2 The `color` Package

HEVEA partly implements the `color` package. Implemented commands are `\definecolor`, `\color` and `\textcolor`. Other commands from the `color` package do not exist. At startup, colors `black`, `white`, `red`, `green`, `blue`, `cyan`, `yellow` and `magenta` are pre-defined.

Colors are defined by `\definecolor{`*name*`}{`*model*`}{`*spec*`}`, where *name* is the color name, *model* is the color model used, and *spec* is the color specification according to the given model. Defined colors are used by the declaration `\color{`*name*`}` and by the command `\textcolor{`*name*`}{`*text*`}`, which change text color. Please note that, the `\color` declaration accepts color specifications directly when invoked as `\color[`*model*`]{`*spec*`}`. The `\textcolor` command has a similar feature.

As regards color models, HEVEA implements the `rgb`, `cmyk`, `hsv` and `hls` color models. In those models, color specifications are floating point numbers less than one. For instance, here is the definition for the `red` color:

```
\definecolor{red}{rgb}{1, 0, 0}
```

The `named` color model is also supported, in this model color specification are just names... Named colors are the ones of `dvips`.

GreenYellow, Yellow, Goldenrod, Dandelion, Apricot, Peach, Melon, YellowOrange, Orange, BurntOrange, Bittersweet, RedOrange, Mahogany, Maroon, BrickRed, Red, OrangeRed, RubineRed, WildStrawberry, Salmon, CarnationPink, Magenta, VioletRed, Rhodamine, Mulberry, RedViolet, Fuchsia, Lavender, Thistle, Orchid, DarkOrchid, Purple, Plum, Violet, RoyalPurple, BlueViolet, Periwinkle, CadetBlue, CornflowerBlue, MidnightBlue, NavyBlue, RoyalBlue, Blue, Cerulean, Cyan, ProcessBlue, SkyBlue, Turquoise, TealBlue, Aquamarine, BlueGreen, Emerald, JungleGreen, SeaGreen, Green, ForestGreen, PineGreen, LimeGreen, YellowGreen, SpringGreen, OliveGreen, RawSienna, Sepia, Brown, Tan, Gray, Black, White.

There are at least three ways to use colors from the `named` model.

1. Define a color name for them.
2. Specify the named color model as an optional argument to `\color` and `\textcolor`.
3. Use the names directly (HEVEA implements the `color` package with the `usenames` option given).

That is:

1. `\definecolor{rouge-brique}{named}{BrickRed}\textcolor{rouge-brique}{Text as a brick}`.
2. `\textcolor[named]{BrickRed}{Text as another brick}`.
3. `\textcolor{BrickRed}{Text as another brick}`.

Colors should be used carefully. Too many colors hinders clarity and some of the colors may not be readable on the document background color.

B.14.2.1 The `bgcolor` environment

Due to limitations in HTML 3.2 (and 4.0 transitional viewed as 3.2!), it is not possible to implement the `\colorbox` command for changing the background color inside a paragraph.

However, HEVEA features a `bgcolor` environment, for changing the backgroud color of some subparts of the document. The `bgcolor` environment is a displayed environment and it normally starts a new line. Simple usage is `\begin{bgcolor}{`*color*`}`... `\end{bgcolor}`, where *color* is a color defined with `\definecolor`. Hence the following source yield a paragraph with a red background:

```
\begin{bgcolor}{red}
\color{yellow}Yellow letters on a red backgroud
\end{bgcolor}
```

The `bgcolor` environment is implemented by a `TABLE` element, it takes an optional argument that is used as an attribute for this `TABLE` element (default value is `CELLPADDING=10`). For instance, the following source:

```
\begin{bgcolor}[CELLPADDING=0]{yellow}
\color{red}Red letters on a yellow backgroud
\end{bgcolor}
```

will be typeset on a yellow background and without padding:

B.14.2.2 From High-Level Colors to Low-Level Colors

High-level colors are color names defined with `\definecolor`. Low-level colors are HTML-style colors. That is, they are either one of the sixteen conventional colors black, silver etc., or a RGB hexadecimal color specification of the form `"#XXXXXX"`.

One changes the high-level *high-color* into a low-level color by `\@getcolor{`*high-color*`}`. Low-level colors are appropriate inside HTML attributes and as arguments to the `\@fontcolor` internal macro. An example of `\@getcolor` usage can be found at the end of section 8.5.

B.15 Font Selection

B.15.1 Changing the Type Style

All LaTeX 2ε declarations and environments for changing type style are recognized. Aspect is rather like LaTeX 2ε output, but there is no guarantee.

As HTML does not provide the same variety of type styles as LaTeX, some type style get rendered by using colors. For instance, the slanted shape yields maroon italics and small caps yields navy blue. Here is how HEVEA implements text-style declarations by default:

`\itshape`	italics
`\slshape`	maroon italics
`\scshape`	navy blue
`\upshape`	no style

`\ttfamily`	typewriter font
`\sffamily`	purple
`\rmfamily`	no style

`\bfseries`	bold
`\mdseries`	no style

Text-style commands also exists, they are defined as `\mbox{\`*decl*`...}`. For instance, `\texttt` is defined as a command with one argument whose body is `\mbox{\ttfamily#1}`. Finally, the `\emph` command for emphasized text also exists.

As in LaTeX, type styles consists in three components: *shape*, *series* and *family*. However this distinction does not exist in HTML: one specifies a type style and that's all. HEVEA implements the three components by making one declaration to cancel the effect of other declarations of the same kind.

Old style declarations are also recognized, they translate to text-level elements. However, no elements are canceled when using old style declaration. Thus, the source "`{\sl\sc slanted and small caps}`" yields navy blue italics. Users need probably not worry about this. However this has an important practical consequence: to change the default rendering of type styles, one should redefine old style declaration in order to benefit from the cancelation mechanism. See section 9.2 for a more thorough description.

B.15.2 Changing the Type Size

All declarations, from `\tiny` to `\Huge` are recognized. Output is not satisfactory inside headers elements generated by sectioning commands.

B.15.3 Special Symbols

The `\symbol{`*num*`}` outputs character number *num* from the iso-latin1 character set. This departs from LaTeX, which output symbol number *num* in the current font.

B.16 Extra Features

This section describes HEVEA functionalities that extends on plain LaTeX, as defined in [LaTeX]. Most of the features described here are performed by default.

B.16.1 Accents in maths

Loading the `mathaccents.hva` style files enables default typesetting of the math accents commands (`\hat`, `\tilde`,...), see Section B.7.4.

B.16.2 TeX macros

Normally, HEVEA does not recognize constructs that are specific to TeX. However, some of the internal commands of HEVEA are homonymous to TeX macros, in order to enhance compatibility. Note that full compatibility with TeX is not guaranteed.

B.16.2.1 À la TEX macros definitions

The \def construct for defining commands is supported. It is important to notice that HEVEA semantics for \def follows TEX semantics. That is, defining a command that already exists with \def succeeds. This is an important change with respect to previous versions of HEVEA, where \def had the same semantics as \newcommand.

Delimiting characters in command definition are supported. Consider the following example from the TEX Book:

```
\def\Look{\textsc{Look}}
\def\x{\textsc{x}}
\def\cs AB#1#2C$#3\$ {#3{ab#1}#1 c\x #2}
\cs AB {\Look}{}C${And \$}{look}\$ 5.
```

It yields: And $lookabLOOKLOOK cX5.

Please note that delimiting characters are supported as far as I could, problems are likely with delimiting characters which include spaces or command names, in particular the command name \{. One can include \{ in a command argument by using the grouping characters {... }:

```
\def\frenchquote(#1){<<~\emph{#1}~>> (in French)}
He said \frenchquote(Alors cette accolade ouvrante {``\{''}~?).
```

Yields:

He said << *Alors cette accolade ouvrante "{" ?* >> (in French).

Another source of incompatibilty with TEX is that substitution of macros parameters is not performed at the same moment by HEVEA and TEX. However, things should go smoothly at the first level of macro expansion, that is when the delimiters appear in source code at the same level as the macro that is to parse them. For instance, the following source will give different results in LATEX and in HEVEA:

```
\def\cs#1A{``#1''}
\def\othercs#1{\cs#1A}
\othercs{coucouA}
```

LATEX output is "coucou"A, while HEVEA output is "coucouA". Here is LATEX output: "coucou"A Please note that in most situations this discrepancy will make HEVEA crash.

B.16.2.2 The \let construct

HEVEA also processes a limited version of \let:

\let *macro-name1* = *macro-name2*

The effect is to bind *macro-name1* to whatever *macro-name2* is bound to at the time \let is processed. This construct may prove very useful in situations where one whishes to slightly modify basic commands. See sections 9.3 and B.2 for examples of using \let in such a situation.

B.16.2.3 The \global construct

It is possible to escape scope and to make global definitions and bindings by using the TEX construct \global. The \global construct is significant before \def and \let constructs.

Also note that \gdef is equivalent to \global\def.

B.16.2.4 TeX Conditional Macros

The `\newif\if`*name*, where *name* is made of letters only, creates three macros: `\if`*name*, `\`*name*`true` and `\`*name*`false`. The latter two set the *name* condition to *true* and *false*, respectively. The `\if`*name* command tests the condition *name*:

`\if`*name*
$text_1$
`\else`
$text_2$
`\fi`

Text $text_1$ is processed when *name* is *true*, otherwise $text_2$ is processed. If $text_2$ is empty, then the `\else` keyword can be omitted.

Note that HEVEA also implements LaTeX `ifthen` package and that TeX simple conditional macros are fully compatible with LaTeX boolean registers. More precisely, we have the following correspondences:

TeX	LaTeX
`\newif\if`*name*	`\newboolean{`*name*`}`
`\`*name*`true`	`\setboolean{`*name*`}{true}`
`\`*name*`false`	`\setboolean{`*name*`}{false}`
`\if`*name* $text_1$`\else` $text_2$`\fi`	`\ifthenelse{\boolean{`*name*`}}{`$text_1$`}{`$text_2$`}`

B.16.2.5 Other TeX Macros

HEVEA implements the macros `\unskip` and `\endinput`. It also supports the `\csname`… `\endcsname` construct.

B.16.3 Command Definition inside Command Definition

If one strictly follows the LaTeX manual, only commands with no arguments can be defined inside other commands. Parameters (i.e., `#`*n*) occurring inside command bodies refer to the outer definition, even when they appear in nested command definitions. That is, the following source:

```
\newcommand{\outercom}[1]{\newcommand{\insidecom}{#1}\insidecom}
\outercom{outer}
```

yields this output:

outer

Nevertheless, nested commands with arguments are allowed. Standard parameters `#`*n* still refer to the outer definition, while nested parameters `##`*n* refer to the inner definition. That is, the source:

```
\newcommand{\outercom}[1]{\newcommand{\insidecom}[1]{##1}\insidecom{inner}}
\outercom{outer}
```

yields this output:

inner

B.16.4 Date and time

Date and time support is not enabled by default, for portability and simplicity reasons.

However, HEVEA source distribution includes a simple (`sh`) shell script `xxdate.exe` that activates date and time support. The `hevea` command, should be invoked as :

```
# hevea -exec xxdate.exe ...
```

This will execute the script `xxdate.exe`, whose output is then read by HEVEA. As a consequence, standard LaTeX counters `year`, `month`, `day` and `\ime` are defined and LaTeX command `\today` works properly. Additionnally the following counters and commands are defined :

Counter `weekday`	day of week, 0…6
Counter `Hour`	hour, 00…11
Counter `hour`	hour, 00…23
Counter `minute`	minute, 00…59
Counter `second`	second, 00…61 (According to `date` man page!)
Command `\ampm`	AM or PM
Command `\timezone`	Time zone
Command `\heveadate`	Output of the "`date`" Unix command

Note that I choosed to add an extra option (and not an extra "`\@exec`" primitive) for security reasons. You certainly do not want to enable HEVEA to execute silently an arbitrary program without being conscious of that fact. Moreover, the `hevea` program does not execute `xxdate.exe` by default since it is difficult to write such a script in a portable manner.

Windows users should enjoy the same features with the version of `xxdate.exe` included in the Win32 distribution.

B.16.5 Fancy sectioning commands

Loading the `fancysection.hva` file will radically change the style of sectionnal units headers: they appear over a green backgroud, the backgrould color saturation decreases as the sectioning commands themselves do. Additionnaly, the document background color is white.

The `fancysection.hva` file is intended to be loaded after the document base style. Thus, to use fancy section style in `doc.tex` whose base style is *article* you should issue the command:

```
# hevea article.hva fancysection.hva doc.tex
```

You can also make a `doc.hva` file that contains the two lines:

```
\input{article.hva}
\input{fancysection.hva}
```

And then launch `hevea` as:

```
# hevea doc.hva doc.tex
```

Sectioning command background colors can be changed by redefining the corresponding colors (`part`, `chapter`, `section`,…). For instance, you get various mixes of red and orange by:

```
\input{article.hva}
\input{fancysection.hva}
\definecolor{part}{named}{BrickRed}
\definecolor{section}{named}{RedOrange}
\definecolor{subsection}{named}{BurntOrange}
```

(See section B.14.2 for details on the `named` color model that is used above.)

Another choice is issuing the command `\colorsections{`*hue*`}`, where *hue* is a hue value to be interpreted in the HSV model. For instance,

```
\input{article.hva}
\input{fancysection.hva}
\colorsections{20}
```

will yield sectionnal headers on a red-orange background.

Table 1: Column specifications from the `array` package

`m{`*width*`}`	Equivalent to the `p` column specification (the *width* argument is ignored, entries are typeset in paragraph mode with paragraph breaks being reduced to a single line break), except that the entries are centered vertically.
`b{`*width*`}`	Equivalent to the `p` column specification, except that the entries are bottom-aligned vertically.
`>{`*decl*`}`	Can be used before `l`, `c`, `r`, `p{...}`, `m{...}` or `b{...}`. It inserts *decl* in front of the entries in the corresponding column.
`<{`*decl*`}`	Can be used after `l`, `c`, `r`, `p{...}`, `m{...}` or `b{...}`. It inserts *decl* after entries in the corresponding column.
`!{`*decl*`}`	Equivalent to `@{`*decl*`}`

B.16.6 HEVEA as a Back-End for VideoC

HEVEA is one of the back-ends of the VideoC system for producing educational CDROM to teach programming languages. VideoC author is Christian Queinnec and the documentation is available at:

`http://www-spi.lip6.fr/~queinnec/VideoC/VideoC.html`.

VideoC translates LaTeX source into a variety of formats, including HTML. VideoC source may contain some special constructs for typesetting source code or to annotate text in sophisticated ways. HEVEA internal engine implements some of the core constructs needed by VideoC. The rest of VideoC constructs are implemented by the `.hva` files from VideoC distribution.

B.17 Implemented Packages

HEVEA distribution includes ".hva" packages that are implementations of LaTeX packages. Packages described in the "*Blue Book*" (`makeidx`, `ifthen`, `graphics` —and `graphicx`!—, `color`, `alltt`) are provided. Additionnaly, quite a few extra packages are provided. I provide no full documentation for these packages, users should refer to the first pages of the package documentation, which can usually be found in the book [LaTeX-bis], in your local LaTeX installation or in a TeX CTAN-archive.

At the moment, package options are ignored.

B.17.1 AMS compatibility

HEVEA `amsmath` package defines some of the constructs of the `amsmath` package. At the moment, supported constructs are the `cases` environment and matrix environments [LaTeX-bis, Section 8.4], the environments for multi-line displayed equations (`gather`, `split`,...) [LaTeX-bis, Section 8.5] and the `\numberwithin` command [LaTeX-bis, Section 8.6.2].

B.17.2 The `array` and `tabularx` Packages

The `array`[3] package is described in [LaTeX-bis, Section 5.3] and in the local documentation of modern LaTeX installations. It is a compatible extension of LaTeX arrays (see B.10.2). Basically, it provides new column specifications and a `\newcolumntype` construct for user-defined column specifications. Table 1 gives a summary of the new column specifications and of how HEVEA implements them.

[3] `ftp://ftp.tex.ac.uk/tex-archive/macros/latex/required/tools/array.dtx`

Note that *centered*, *top-aligned* or *bottom-aligned* in the vertical direction, do not have exactly the same meaning in LaTeX and in HTML. However, the aspect is the same when all columns agree w.r.t. vertical alignment. Ordinary column types (`c`, `l` and `r`) do not specify vertical alignment, which therefore becomes browser dependent.

The `>{`*decl*`}` and `<{`*decl*`}` constructs permit the encoding of TeX `\cases` macro as follows:

```
\def\cases#1{\left\{\begin{array}{l>{$}l<{$}}#1\end{array}\right.}
```

(This is an excerpt of the `latexcommon.hva` file.)

New column specifications are defined by the `\newcolumntype` construct:

`\newcolumntype{`*col*`}[`*narg*`]{`*body*`}`

Where *col* is one letter, the optional *narg* is a number (defaults to `0`), and `body` is built up with valid column specifications and macro-argument references (`#`*int*). Examples are:

```
\newcolumntype{C}{>{\bf}c}
\newcolumntype{E}[1]{*{#1}{c}}
\begin{tabular}{CE{3}}\hline
one & two & three & four \\
five & six & seven & eight \\ \hline
\end{tabular}
```

The column specification `C` means that entries will be typeset centered and using bold font, while the column specifications `E{`*num*`}` stands for *num* centered columns. We get:

one	two	three	four
five	six	seven	eight

HEVEA implements column specifications with commands defined in the `\newcommand` style. Thus, they have the same behavior as regards double definition, which is not performed and induces a warning message. Thus, a column specification that is first defined in a `macro.hva` specific file, overrides the document definition.

The `tabularx`[4] package [LaTeX-bis, Section 5.3.5] provides a new tabular environment `tabularx` and a new column type `X`. HEVEA makes the former equivalent to `tabular` and the latter equivalent to `p{`*ignored*`}`. By contrast with the subtle array formatting that the `tabularx` package performs, this may seem a crude implementation. However, rendering is usually correct, although different.

More generally and from the HTML point of view such sophisticated formatting is browser job in the first place. However, the HTML definition allows suggested widths or heights for table entries and table themselves. From HEVEA point of view, drawing the border line between what can be specified and what can be left to the browser is not obvious at all. At the moment HEVEA choice is not to specify too much (in particular, all length arguments, either to column specifications or to the arrays themselves, are ignored). As a consequence, the final, browser viewed, aspect of arrays will usually be different from their printed aspect.

B.17.3 The `calc` Package

LaTeX source[5] and documentation.

This package enables using traditional, infix, notation for arithmetic operations inside the *num* argument to the `\setcounter{`*name*`}{`*num*`}` and `\addtocounter{`*name*`}{`*num*`}` constructs (see [LaTeX-bis, Section A.4])

The `calc` package provides a similar extension of the syntax of the *len* argument to the `\setlength` and `\addtolength` constructs. HEVEA does not implement this extension, since it does not implement length registers in the first place.

[4]`ftp://ftp.tex.ac.uk/tex-archive/macros/latex/required/tools/tabularx.dtx`

[5]`ftp://ftp.tex.ac.uk/tex-archive/macros/latex/required/tools/calc.dtx`

B.17.4 The `comment` Package

LaTeX source[6].

The implementation for this package provides two commands, `\excludecomment` and `\includecomment`, for (re-)defining new environments that ignore their content or that do nothing. The comment environment is also defined as an environment of the first kind.

B.17.5 Multiple Indexes with the `index` and `multind` package

HEVEA supports several simultaneous indexes, following the scheme of the `index`[7] package, which is present in modern LaTeX distributions. This scheme is backward compatible with the standard indexing scheme of LaTeX.

Support is not complete, but the most useful commands are available. More precisely, HEVEA knows the following commands:

`\newindex`{*tag*}{*ext*}{*ignored*}{*indexname*} Declare an index. The first argument *tag* is a tag to select this index in other commands; *ext* is the extension of the index information file generated by LaTeX (e.g., `idx`); *ignored* is ignored by HEVEA; and *indexname* is the title of the index. If given the `idx` option. HEVEA attempts to read file *mydoc.ext*. There also exists a `\renewindex` commands that takes the same arguments and that can be used to redefine previously declared indexes.

`\makeindex` Perform `\newindex{default}{idx}{ind}{Index}`. z

`\index[`*tag*`]`{*arg*} Act as the LaTeX `\index` command except that the information extracted from *arg* goes to the *tag* index. The *tag* argument defaults to `default`, thereby yielding standard LaTeX behavior for the `\index` command without an optional argument. There also exists a stared-variant `\index*` that Additionally typesets *arg*.

`\printindex[`*tag*`]` Compute, format and output index whose tag is *tag*. The *tag* argument defaults to `default`.

The `multind`[8] package provides another scheme for multiple indexes. Note that the "`multind`" style for indexes is not backward compatible with LaTeX default indexing scheme. I would recommend using the "`index`" package.

B.17.6 Multiple Bibliographies with the `multibib` package

. HEVEA provides a slighty uncomplete implementation of the `multibib` package. The one non-implemented feature is the simultaneous definition of more than one bibliography. That is one cannot invoke `\newcites` as follows :

```
\newcites{suf1, suf2}{Title1, Title2}
```

Instead, one should perform to calls to the `\newcites` command :

```
\newcites{suf1}{Title1}
\newcites{suf2}{Title2}
```

B.17.7 The `url` package

LaTeX source[9].

This package in fact provides a enhanced `\verb` command that can appear inside other command arguments. This command is named "`\url`", but it can be used for any verbatim text, including DOS-like path names. Hence, one can insert urls in one's document without worrying about LaTeX active characters :

[6] `ftp://ftp.tex.ac.uk/tex-archive/macros/latex/contrib/other/comment/`
[7] `ftp://theory.lcs.mit.edu/pub/tex/index/`
[8] `ftp://ftp.tex.ac.uk/tex-archive/macros/latex209/contrib/misc/multind.sty`
[9] `ftp://ftp.tex.ac.uk/tex-archive/macros/latex/contrib/other/misc/url.sty`

```
This is a complicated url: \url{http://foo.com/~user#label%coucou}.
```

which gets typeset as: "This is a complicated url: http://foo.com/~user#label%coucou."

Main use for the \url command is to specify urls as arguments to HEVEA commands for hyperlinks (see section 8.1.1) :

```
\hevea{} home page is
\ahrefurl{\url{http://pauillac.inria.fr/~maranget/hevea/}}
```

It yields : "HEVEA home page is http://pauillac.inria.fr/~maranget/hevea/".

However the \url command is fragile, as a consequence it cannot be used inside \footahref first argument (This is a LaTeX problem, not an HEVEA one). The url package solves this problem by providing the \urldef command for defining commands whose body is typeset by using \url:

```
\urldef{\heveahome}{\url}{http://pauillac.inria.fr/~maranget/hevea/}
```

Such a source defines the robust command \heveahome as the intended url. Hence the following source works as expected :

```
Have a look at \footurl{\heveahome}{\hevea{} home page}
```

It yields: "Have a look at HEVEA home page[10]".

Using \url inside command definitions with a #*i* argument is a bad idea, since it gives "verbatim" a rather random meaning. Unfortunately, in some situations (e.g, no %, no #), it may work in LaTeX. By contrast, it does not work in HEVEA. In such situations, \urldef should be used.

HEVEA implementation is somehow compatible at the "programming level". Thus, users can define new commands whose argument is understood verbatim. The urlhref.hva style file from the distribution takes advantage of this to define the \url command, so that it both typesets an url and inserts a link to it. The urlhref.hva style file (which is an HEVEA style file and not a LaTeX style file) can be adequate for bibliographic references, which often use \url for its typesetting power. Of course, loading urlhref.hva only makes sense when all arguments to \url are urls...

B.17.8 Verbatim Text : the moreverb and verbatim Packages

These two packages provide new commands and environments for processing verbatim text.

I recommend using moreverb[11] rather than verbatim[12], since the LaTeX implementation of the former is much more compatible with HEVEA than the implementation of the latter.

B.17.9 Typesetting Computer Languages: the listings Package

I strongly recommand using the listings[13]. Learning the user interface requires a little effort, but it is worth it.

HEVEA features a quite compatible implementation, please refer to the original package documentation. Do not hesitate to report discrepancies. Note that HEVEA does not produce very compact HTML in case you use this package. This can be cured, at some price in runtime cost, by giving hevea the command line option "-O" (see Section C.1.1.4).

B.17.10 Experimental Implementations

The fancyverb and colortbl packages are partly implemented.

[10] http://pauillac.inria.fr/~maranget/hevea/
[11] ftp://ftp.tex.ac.uk/tex-archive/macros/latex/contrib/supported/moreverb/
[12] ftp://ftp.tex.ac.uk/tex-archive/macros/latex/required/tools/verbatim.dtx
[13] ftp://ftp.tex.ac.uk/tex-archive/macros/latex/contrib/supported/listings

Part C

Practical information

C.1 Usage

C.1.1 HEVEA usage

The `hevea` command has two operating modes, normal mode and filter mode. Operating mode is determined by the nature of the last command line argument.

C.1.1.1 Command line arguments

The `hevea` command interprets its arguments as names of files and attempts to process them. Given an argument *filename* there are two cases:

- If *filename* is *base*`.tex` or *base*`.hva`, then a single attempt to open *filename* is made.
- In other cases, a first attempt to open *filename*`.tex` is made. In case of failure, a second attempt to open *filename* is made.

In all attempts, implicit filenames are searched along `hevea` search path, which consist in:

1. the current directory "`.`",
2. user-specified directories (with the `-I` command line option),
3. `hevea` library directory.
4. one of the sub-directories `html`, `text` or `info` from `hevea` library directory, depending upon `hevea` output format,

The `hevea` library directory is fixed at compile-time (this is where `hevea` library files are installed) and typically is `/usr/local/lib/hevea`. However, this compile-time value can be overridden by setting the `HEVEADIR` shell environment variable.

C.1.1.2 Normal mode

If the last argument has an extension that is different from `.hva` or has no extension, then it is interpreted as the name of the *main input file*. The main input file is the document to be translated and normally contains the `\documentclass` command. In that case two *basenames* are defined:

- The input basename, *basein*, is defined as the main input file name, with extension removed when present.
- The output basename, *baseout*, is *basein* with leading directories omitted. However the output basename can be changed, using the `-o` option (see the section on options below).

HEVEA will attempt to load the main input file. Ancillary files from a previous run of LaTeX (i.e., `.aux`, `.bll` and `.idx` files) will be searched as *basein*.*ext*. The output base name governs all files produced by HEVEA. That is, HTML output of HEVEA normally goes to the file *baseout*`.html`, while cross-referencing information goes into *baseout*`.haux`. Furthemore, if an *image* file is generated (cf. section 6), its name will be *baseout*`.image.tex`.

Thus, in the simple case where the `hevea` command is invoked as:

```
# hevea file.tex
```

The input basename is `file` and the output basename also is `file`. The main input file is searched once along `hevea` search path as `file.tex`. HTML output goes into file `file.html`, in the current directory.

In the more complicated case where the `hevea` command is invoked as:

```
# hevea ./dir/file
```

The input base name is `./dir/file` and the output basename is `file`. The main input file is loaded by first attempting to open file `./dir/file.tex`, then file `./dir/file`. HTML output goes into file `file.html`, in the current directory.

The `article.hva`, `seminar.hva`, `book.hva` and `report.hva` base style files from HEVEA library are special. Only the first base style file is loaded and the `\documentclass` command has no effect when a base style file is already loaded. This feature allows to override the document base style. Thus, a document `file.tex` can be translated using the *article* base style as follows:

```
# hevea article.hva file.tex
```

C.1.1.3 Filter mode

If there is no command line argument, or if the last command line argument has the extension `.hva`, then there is neither input base name nor output base name, the standard input is read and output normally goes to the standard output. Output starts immediately, whithout waiting for `\begin{document}`. In other words `hevea` acts as a filter.

Please note that this operating mode is just for translating isolated LaTeX constructs. The normal way to translate a full document *file*`.tex` being "`hevea` *file*`.tex`" and not "`hevea <` *file*`.tex >` *file*`.html`".

C.1.1.4 Options

The `hevea` command recognizes the following options:

`-version` Show `hevea` version and exit.

`-v` Verbose flag, can be repeated to increase verbosity. However, this is mostly for debug.

`-s` Suppress warnings.

`-e` *filename* Prevent `hevea` from loading any file whose name is *filename*. Note that this option applies to all files, including `hevea.hva` and base style files.

`-fix` Iterate HEVEA until a fixpoint is found. Additionnaly, images get generated automatically.

`-O` Optimize HTML by calling `esponja` (see section C.1.3).

`-exec` *prog* Execute file *prog* and read the output. The file *prog* must have execution permission and is searched by following the searching rules of `hevea`.

`-francais` Set French mode. This has three consequences:

- Some words inserted by LaTeX (such as "Chapter", "Bibliography", ...) are replaced by French word.
- Text replacement for symbols are in French (see the `-nosymb` option below).
- `hevea` sets the boolean register `french` to *true*.

`-nosymb` Avoid symbol font. In this mode, symbols are replaced by text-only equivalents. By default, these equivalent are in English.

`-noiso` Do not output (iso-latin1) characters whose code is above 127 (i.e. output ascii only). These characters are replaced by HTML entities. This option is mostly useful for generating HTML that will be displayed properly by *Netscape Communicator* on a Macintosh.

`-pedantic` Be strict in interpreting HTML definition. In particular, this option disable size and color changes inside `<PRE>`... `</PRE>`, which are otherwise performed.

`-I` *dirname* Add *dirname* to the search path.

-o *name* Make *name* the output basename. However, if *name* is *base*.html, then the output basename is *base*.

-help Print version number and a short help message.

The following options select and control alternative output formats (see section 10):

-text Output plain text. Output file extension is .txt.

-info Output info format. Output file extension is .info.

-w *width* Set the line width for text or info output, defaults to 72.

Part A of this document is a tutorial introduction to HEVEA, while HEVEA reference manual is part B.

C.1.2 HACHA usage

The hacha command interprets its argument *base*.html as the name of a HTML source file to cut into pieces. It also recognizes the following options:

-v Be a little verbose.

-o *filename* Make HACHA output go into file *filename* (defaults to index.html).

-tocbis Add a small table of contents at the begining of every file.

-nolinks Do not insert Previous/Up/Next links in generated pages.

-hrf Output a *base*.hrf file, showing in which output files are the anchors from the input file gone. The format of this summary is one "*anchor*\t*file*" line per anchor. This information may be needed by other tools.

-help Print version number and a short help message.

Section 7 of the user manual explains how to alter HACHA default behavior.

C.1.3 esponja usage

The program esponja is part of HEVEA and is designed to optimize hevea output. However, esponja can also be used alone to optimize text-level elements in HTML files. Since esponja fails to operate when it detects incorrect HTML, it can be used as a partial HTML validator.

C.1.3.1 Operating modes

With no argument, esponja acts as a filter, it reads the standard input and writes on the standard output. Otherwise, esponja interprets its arguments as names of files and attempt to process them. It is important to notice that esponja will *replace* files by their optimized versions.

Hence, to optimize file foo.html into foo_opt.html, one should invoke esponja as follows.

```
# esponja < foo.html > foo_opt.html
```

By contrast, invoking esponja as

```
# esponja foo.html
```

will alter foo.html. Of course, if esponja does not succeed in making foo.html any smaller or if esponja fails, the original foo.html is left unchanged. Note that this feature allows to optimize all HTML files in a given directory by:

```
# esponja *.html
```

C.1.3.2 Options

The command `esponja` recognizes the following options:

`-v` Be verbose, can be repeated to increase verbosity.

`-n` Do not alter input files. Instead, `esponja` output for file *input* goes to file *input*`.esp`. Option `-n` implies option `-v`.

`-u` Output `esponja` intermediate version of HTML. In most occasions, this amouts to pessimize instead of to optimize. It may yield challenging input for other HTML optimizers.

C.1.4 `imagen` usage

The command `imagen` is a simple shell script that translates a LaTeX document into many `.gif` images. The `imagen` script relies on much software to be installed on your computer, see Section C.4.1.

It is a companion program of HEVEA, which must have been previously run as:

`# hevea`... *base*`.tex`
or
`# hevea`... `-o` *base*`.html`...

(In both cases, *base* is HEVEA output basename.) When told to do so (see section 6) HEVEA echoes part of its input into the *base*`.image.tex` file.

The `imagen` script should then be run as:

`# imagen` *base*

The `imagen` script produces one *basennn*`.gif` image file per page in the *base*`.image.tex` file.

This is done by first calling `latex` on *base*`.image.tex`, yielding one `dvi` file. Then, `dvips` translates this file into one single Postscript file that contains all the images, or into one Postscript file per image, depending upon your version of `dvips`. Postscript files are interpreted by ghostscript (`gs`) that outputs `ppm` images, which are then fed into a series of transformations that change them into `.gif` files.

The `imagen` script recognizes the following options:

`-mag` *nnnn* Change the enlarging ratio that is applied while translating DVI into Postscript. More precisely, `dvips` is run with `-x`*nnnn* option. Default value for this ration is 1414, this means that, by default, `imagen` magnifies LaTeX output by a factor of 1.414.

`-extra` *command* Insert *command* as an additional stage in `imagen` `ppm` to `gif` production chain. *command* is an Unix filter that expects a `ppm` image in its standard input and outputs a `ppm` image on its standard output. A sensible choice for *command* is one command from the `netpbm` package or several such commands piped together.

`-quant` *number* Add an extra color quantization step in `imagen` `ppm` image production chain, where *number* is the maximal number of colors in the produced images. This option may be needed in response of a failure in the image production chain. It can also help in limiting image files size.

`-gif` Output GIF images. This is the default.

`-png` Output PNG images in place of GIF images. PNG is sometimes prefered for legal reasons. PNG image files have a "`.png`" extension. Note that `hevea` should have been previously run as `hevea png.hva` *base*`.tex` (so that the proper "`.png`" filename extension is given to image file references from within the HTML document). Beware that the `pnmtopng` program looks to be plagued by bugs in old versions of the `netpbm` package.

`-pnm` Output PPM images. This option mostly serves debugging purposes. Experimented users can also take advantage of it for performing additional image transformation or adopting exotic image formats.

The first three options enable users to correct some misbehaviors. For instance, when the document base style is *seminar*, image orientation may be wrong and the images are too small. This can be cured by invoking `imagen` as:

```
# imagen -extra "pnmflip -ccw" -mag 2000 base
```

Sometimes `imagen` crashes because the PPM transformation chain does not cope with images with more than 256 colors by default. The solution is to re-issue the `imagen` command as:

```
# imagen -quant 256 base
```

More information on producing images can be found in section 6.

C.1.5 Using `make`

Here is a typical `Makefile` for translating a `doc.tex` source file into HTML. The file is first translated into `doc.html` by `hevea`, which also reads the specific style file `macros.hva`. Then, `hacha` cuts `doc.html` into several, `doc001.html`, `doc002.html`, etc. also producing the table of links file `index.html`.

```
HEVEA=hevea
HEVEAOPTS=-fix
HACHA=hacha
#document base name
DOC=doc
index.html: $(DOC).html
        $(HACHA) -o index.html $(DOC).html

$(DOC).html: macros.hva $(DOC).tex
        $(HEVEA) $(HEVEAOPTS) macros.hva $(DOC).tex
clean:
        rm -f $(DOC).html index.html $(DOC)[0-9][0-9][0-9].html $(DOC).haux
```

Note that the `clean` rule removes all the `doc001.html`, `doc002.html`, etc. files produced by `hacha`. Also note that `make clean` also removes the `doc.haux` file. Thanks to the `-fix` options, `hevea` will run the appropriate number of times.

When the *image* file feature is used, one can use the following, extended, `Makefile`:

```
HEVEA=hevea
HEVEAOPTS=-fix
HACHA=hacha
IMAGEN=imagen
#document base name
DOC=doc
index.html: $(DOC).html
        $(HACHA) -o index.html $(DOC).html

$(DOC).html: macros.hva $(DOC).tex
        $(HEVEA) $(HEVEAOPTS) macros.hva $(DOC).tex

clean:
        rm -f $(DOC).html index.html $(DOC)[0-9][0-9][0-9].html $(DOC).haux
        rm -f $(DOC).htoc
        rm -f $(DOC).image.*
        rm -f $(DOC)[0-9][0-9][0-9].gif
```

Observe that the `clean` rule now also gets rid of `doc.image.tex` and of the various files produced by `imagen`. Note the following, useful feature : when given the `-fix` option, `hevea` will itself run `imagen`, if needed.

C.2 Browser configuration

By default, HEVEA uses the FACE=symbol attribute to the <FONT ...> tag. This attribute is non-standard in HTML-3.2 and is now standard (and already deprecated) in HTML-4.0

The symbol font is the one available on the Linux Red Hat distribution and seems to be present on many Unix installations. A good way to know whether your browser can show HEVEA symbols or not is comparing figure 1 and the web page located at symbol.html[14].

Microsoft Explorer displays the symbol font by default.

By default, *Netscape Communicator* does not display symbol fonts as intended by HEVEA. On an Unix system, the following procedure instructs *Netscape Communicator* to do so:

- Add the following line to your .Xdefaults file:

```
Netscape*documentFonts.charset*adobe-fontspecific:   iso-8859-1
```

- Issue a xrdb .Xdefaults command.
- Restart *Netscape*.
- Furthermore, be carefull to enable using document fonts while disabling dynamic fonts : select the Edit/Preference/Appearance/Fonts menu, and then check the appropriate box, which should be something like "Use document-specified fonts".

On a Macintosh, choose **Western (Mac Roman)** in the item **Document Encoding** from the *Preferences* menu. This will work only if the document does not contain iso-latin1 characters above 127. That is, this will work only with documents that are generated by HEVEA with the -noiso option enabled (see section C.1.1.4).

More details on browser configuration can be found at http://hutchinson.belmont.ma.us/tth/Xfonts.html (A page from Ian Hutchinson's tth documentation). In particular the *Simple fix* of the .Xdefaults file above does not seem to work on *Mozilla 4.78*, and you should rely on the more complicated fix described in the web page.

More details on symbol fonts can be found in section 3.2.2 of this manual. In particular, note that symbol fonts may be completely avoided with the -nosymb option.

C.3 Availability

C.3.1 Internet stuff

HEVEA home page is http://pauillac.inria.fr/~maranget/hevea. It contains links to the on-line manual[15] and to the distribution[16].

The author can be contacted at Luc.Maranget@inria.fr.

C.3.2 Law

HEVEA can be freely used and redistributed without modifications. Modifying and redistributing HEVEA implies a few constraints. More precisely, HEVEA is distributed under the terms of the Q Public License, but HEVEA binaries include the Objective Caml runtime system, which is distributed under the Gnu Library General Public License (LGPL). See the LICENSE[17] file for details.

The manuel itself is distributed under the terms of the Free Document Dissemination Licence[18].

[14] http://pauillac.inria.fr/~maranget/hevea//doc/symbol.html
[15] http://pauillac.inria.fr/~maranget/hevea/doc/
[16] ftp://ftp.inria.fr/INRIA/moscova/hevea
[17] ftp://ftp.inria.fr/INRIA/moscova/hevea/LICENSE
[18] http://pauillac.inria.fr/~lang/licence/v1/fddl.html

C.4 Installation

C.4.1 Requirements

The programs `hevea` and `hacha` are written in Objective Caml[19]. Thus, you really need Objective Caml (the more recent version, the better) to compile them. However, a Red Hat 7.2 binary distribution[20] is also available, it does not require an Objective Caml installation and can be installed on most Red Hat Linux PC's.

HEVEA users may instruct the program not to process a part of the input (see section 6). Instead, this part is processed into a `.gif` file and HEVEA outputs a link to the image file. LaTeX source is changed into `.gif` images by the `imagen` script, which basically calls, LaTeX, `dvips`, `ghostscript`[21] and a few tools from the image processing package `netpbm`[22].

To benefit from the full functionality of HEVEA, you need all this software. However, HEVEA runs without them, but then you will to manage to produce images by yourself.

C.4.2 Principles

The details are given in the `README` file from the distribution. Basically, HEVEA should be given a library directory. The installation procedure stores the `hevea.hva` and base style files in this directory. There are two compilation modes, the `opt` mode selects the native code Ocaml compiler `ocamlopt`, while the `byte` mode selects the bytecode Ocaml compiler `ocamlc`. In HEVEA case, `ocamlopt` produces code that is up to three times as fast as the one produced by `ocamlc`. Thus, default compilation mode is `opt`, however it may be the case on some systems that only `ocamlc` is available.

Note that the `hevea.sty` file is simply copied to HEVEA library directory. It remains users responsibility to make it accessible to LaTeX.

C.5 Other LaTeX to HTML translators

This short section gives pointers to a few other translators. I performed not extensive testing and make no thorough comparison.

LaTeX2html LaTeX2html is a full system. It is written in perl and calls LaTeX when in trouble. As a consequence, LaTeX2html is powerful but it may fail on large documents, for speed and memory reasons. More information on LaTeX2html can be found at

`http://www-dsed.llnl.gov/files/programs/unix/latex2html/`

TTH The principle behind TTH is the same as the one of HEVEA: write a fast translator as a lexer, use symbol fonts and tables. However, there are differences, TTH accepts both TeX and LaTeX source, TTH is written in C and the full source is not available (only `lex` output is available). Additionally, TTH insist on not using any kind of LaTeX generated information and will show proper cross-reference labels, even when no `.aux` file is present. TTH output is a single document, whereas HACHA can cut the output of HEVEA into several files. (however there exists a commercial version of TTH that provides this extra functionality). TTH can be found at

`http://hutchinson.belmont.ma.us/tth/`.

`htmlgen` The `htmlgen` translator is specialized for producing the Caml manuals. This is HEVEA direct ancestor and I owe much to its author, X. Leroy. See [htmlgen] for a description of `htmlgen` and a (bit outdated) discussion on LaTeX to HTML translation.

[19] `http://caml.inria.fr/ocaml/`
[20] `ftp://ftp.inria.fr/INRIA/moscova/hevea/hevea-1.07-1.i386.rpm`
[21] `http://www.cs.wisc.edu/~ghost/index.html`
[22] `ftp://wuarchive.wustl.edu/graphics/graphics/packages/NetPBM`

A fairly complete list of LaTeX to HTML translators can be found at:

`http://www.loria.fr/services/tex/english/outils.html`

C.6 Acknowledgements

The following people contributed to HEVEA development:

- Philip A.Viton, maintains a window (win32) port of HEVEA.
- Christian Queinnec wrote an extra lexer to translate code snippets produced by its tool VideoC for writing pedagogical documents on programming. The very principle he introduced for interfacing the `videoc` lexer with HEVEA main lexer is now used extensively throughout HEVEA source code.
- Pierre Boulet, by using HEVEA as a stage in his tool MlDoc for documenting Objective Caml source code, forced me into debugging HEVEA implementation of the `alltt` environment.
- Nicolas Tessaud implemented the `-text` and `-info` output modes (see section 10).
- Georges Mariano maintains the hevea mailing list, asked for many feature, and argued a lot to have them implemented.
- Many users contributed by sending bug reports.

References

[LaTeX-bis] M. Gooseens, F. Mittelbach, A. Samarin. *The LaTeX Companion* Addison-Websley, 1994.

[LaTeX] L. Lamport. *A Document Preparation System System, LaTeX, User's Guide and Reference Manual.* Addison-Websley, 1994.

[htmlgen] X. Leroy. *Lessons learned from the translation of documentation from LaTeX to HTML.* ERCIM/W4G Int. Workshop on WWW Authoring and Integration Tools, 1995. Available on the web at `http://cristal.inria.fr/~xleroy/w4g.html`

[HTML-3.2] D. Ragget. *HTML 3.2 Reference Specification.* Available on the web at `http://www.w3.org/TR/REC-html32`, 1997.

[HTML-4.0] D. Ragget, A. Le Hors and I. Jacobs. *HTML 4.0 Reference Specification.* Available on the web at `http://www.w3.org/TR/REC-html40`, 1997.

Index