The Hyperlatex Markup Language

Otfried Cheong

Institute of Information and Computing Sciences
Utrecht University
Utrecht, the Netherlands

1 Introduction

Hyperlatex is a package that allows you to prepare documents in HTML, and, at the same time, to produce a neatly printed document from your input. Unlike some other systems that you may have seen, Hyperlatex is *not* a general LaTeX-to-HTML converter. In my eyes, conversion is not a solution to HTML authoring. A well written HTML document must differ from a printed copy in a number of rather subtle ways—you'll see many examples in this manual. I doubt that these differences can be recognized mechanically, and I believe that converted LaTeX can never be as readable as a document written for HTML.

This manual is for Hyperlatex 2.6, of February 2002.

The basic idea of Hyperlatex is to make it possible to write a document that will look like a flawless LaTeX document when printed and like a handwritten HTML document when viewed with an HTML browser. In this it completely follows the philosophy of `latexinfo` (and `texinfo`). Like `latexinfo`, it defines its own input format—the *Hyperlatex markup language*—and provides two converters to turn a document written in Hyperlatex markup into a DVI file or a set of HTML documents.

Obviously, this approach has the disadvantage that you have to learn a "new" language to generate HTML files. However, the mental effort for this is quite limited. The Hyperlatex markup language is simply a well-defined subset of LaTeX that has been extended with commands to create hyperlinks, to control the conversion to HTML, and to add concepts of HTML such as horizontal rules and embedded images. Furthermore, you can use Hyperlatex perfectly well without knowing anything about HTML markup.

The fact that Hyperlatex defines only a restricted subset of LaTeX does not mean that you have to restrict yourself in what you can do in the printed copy. Hyperlatex provides many commands that allow you to include arbitrary LaTeX commands (including commands from any package that you'd like to use) which will be processed to create your printed output, but which will be ignored in the HTML document. However, you do have to specify that *explicitly*. Whenever Hyperlatex encounters a LaTeX command outside its restricted subset, it will complain bitterly.

The rationale behind this is that when you are writing your document, you should keep both the printed document and the HTML output in mind. Whenever you want to use a LaTeX command with no defined HTML equivalent, you are thus forced to specify this equivalent. If, for instance, you have marked a logical separation between paragraphs with LaTeX's `\bigskip` command (a command not in Hyperlatex's restricted set, since there is no HTML equivalent), then Hyperlatex will complain, since very probably you would also want to mark this separation in the HTML output. So you would have to write

```
\texonly{\bigskip}
\htmlrule
```

to imply that the separation will be a `\bigskip` in the printed version and a horizontal rule in the HTML-version. Even better, you could define a command `\separate` in the preamble and give it a different meaning in DVI and HTML output. If you find that for your documents `\bigskip` should always be ignored in the HTML version, then you can state so in the preamble as follows. (It is also possible that you setup personal definitions like these in your personal *init.hlx* file, and Hyperlatex will never bother you again.)

```
\W\newcommand{\bigskip}{}
```

This philosophy implies that in general an existing LaTeX-file will not make it through Hyperlatex. In many cases, however, it will suffice to go through the file once, adding the necessary markup that specifies how Hyperlatex should treat the unknown commands.

2 Using Hyperlatex

Using Hyperlatex is easy. You create a file *document.tex*, say, containing your document with Hyperlatex markup (the most important LaTeX-commands, with a number of additions to make it easier to create readable HTML).

If you use the command

```
latex document
```

then your file will be processed by LaTeX, resulting in a DVI-file, which you can print as usual.

On the other hand, you can run the command

```
hyperlatex document
```

and your document will be converted to HTML format, presumably to a set of files called *document.html*, *document_1.html*, ... You can then use any HTML-viewer or WWW-browser to view the document. (The entry point for your document will be the file *document.html*.)

This document describes how to use the Hyperlatex package and explains the Hyperlatex markup language. It does not teach you *how* to write for the web. There are style guides available, which you might want to consult. Writing an on-line document is not the same as writing a paper. I hope that Hyperlatex will help you to do both properly.

This manual assumes that you are familiar with LaTeX, and that you have at least some familiarity with hypertext documents—that is, that you know how to use a WWW-browser and understand what a *hyperlink* is.

If you want, you can have a look at the source of this manual, which illustrates most points discussed here. You can also look at the documents on my home page, all of which are created using Hyperlatex.

The primary distribution site for Hyperlatex is at http://hyperlatex.sourceforge.net, the Hyperlatex home page.

There is also a mailing list for Hyperlatex, maintained at sourceforge.net. This list is for discussion (and support) of Hyperlatex and anything that relates to it. Instructions for subscribing are also on the Hyperlatex home page.

The FAQ and the mailing list are the only "official" place where you can find support for problems with Hyperlatex. I am unfortunately no longer in a position to answer mail with questions about Hyperlatex. Please understand that Hyperlatex is just a by-product of Ipe–I wrote it to be able to write the Ipe manual the way I wanted to. I am making Hyperlatex available because others seem to find it useful, and I'm trying to make this manual and the installation instructions as clear as possible, but I cannot provide any personal support. If you have problems installing or using Hyperlatex, or if you think that you have found a bug, please mail it to the Hyperlatex mailing list. One of the friendly Hyperlatex users will probably be able to help you.

A final footnote: The converter to HTML implemented in Hyperlatex is written in GNU Emacs Lisp. If you want, you can invoke it directly from Emacs (see the beginning of *hyperlatex.el* for instructions). But even if you don't use Emacs, even if you don't like Emacs, or even if you subscribe to `alt.religion.emacs.haters`, you can happily use Hyperlatex. Hyperlatex can be invoked from the shell as "hyperlatex," and you will never know that this script calls Emacs to produce the HTML document.

The Hyperlatex code is based on the Emacs Lisp macros of the `latexinfo` package.

3 About the Html output

Hyperlatex will automatically partition your input file into separate HTML files, using the sectioning commands in the input. It attaches buttons and menus to every HTML file, so that the reader can walk through your document and can easily find the information that she is looking for. (Note that HTML documentation usually calls a single HTML file a "document". In this manual we take the LaTeX point of view, and call "document" what is enclosed in a `document` environment. We will use the term *node* for the individual HTML files.) You may want to experiment a bit with the HTML version of this manual. You'll find that every `\section` and `\subsection` command starts a new node. The HTML node of a section that contains subsections contains a menu whose entries lead you to the subsections. Furthermore, every HTML node has three buttons: *Next*, *Previous*, and *Up*.

The *Next* button leads you to the next section *at the same level*. That means that if you are looking at the node for the section "Getting started," the *Next* button takes you to "Conditional Compilation," *not* to "Preparing an input file" (the first subsection of "Getting started"). If you are looking at the last subsection of a section, there will be no *Next* button, and you have to go *Up* again, before you can step further. This makes it easy to browse quickly through one level of detail, while only delving into the lower levels when you become interested. (It is possible to change this behavior so that the *Next* button always leads to the next piece of text, see Section 11.2.)

If you look at the HTML output for this manual, you'll find that there is one special node that acts as the entry point to the manual, and as the parent for all its sections. This node is called the *top node*. Everything between `\begin{document}` and the first sectioning command (such as `\section` or `\chapter`) goes into the top node.

An HTML file needs a *title*. The default title is "Untitled", you can set it to something more meaningful in the preamble[1] of your document using the `\htmltitle` command. You should use something not too long, but useful. (The HTML title is often displayed by browsers in the window header, and is used in history lists or bookmark files.) The title you specify is used directly for the top node of your document. The other nodes get a title composed of this and the section heading.

It is common practice to put a short notice at the end of every HTML node, with a reference to the author and possibly the date of creation. You can do this by using the `\htmladdress` command in the preamble, like this:

```
\htmladdress{Otfried Cheong, \today}
```

4 Trying it out

For those who don't read manuals, here are a few hints to allow you to use Hyperlatex quickly.

Hyperlatex implements a certain subset of LaTeX, and adds a number of other commands that allow you to write better HTML. If you already have a document written in LaTeX, the effort

[1] The *preamble* of a LaTeX file is the part between the `\documentclass` command and the `\begin{document}` command. LaTeX does not allow text in the preamble; you can only put definitions and declarations there.

to convert it to Hyperlatex should be quite limited. You mainly have to check the preamble for commands that Hyperlatex might choke on.

The beginning of a simple Hyperlatex document ought to look something like this:

```
\documentclass{article}
\usepackage{hyperlatex}

\htmltitle{Title of HTML nodes}
\htmladdress{Your Email address, for instance}

    more LaTeX declarations, if you want

\title{Title of document}
\author{Author document}

\begin{document}

\maketitle

This is the beginning of the document...
```

Note the use of the *hyperlatex* package. It contains the definitions of the Hyperlatex commands that are not part of LaTeX.

Those few commands are all that is absolutely needed by Hyperlatex, and adding them should suffice for a simple LaTeX document. You might try it on the *sample2e.tex* file that comes with LaTeX 2_ε, to get a feeling for the HTML formatting of the different LaTeX concepts.

Sooner or later Hyperlatex will fail on a LaTeX-document. As explained in the introduction, Hyperlatex is not meant as a general LaTeX-to-HTML converter. It has been designed to understand a certain subset of LaTeX, and will treat all other LaTeX commands with an error message. This does not mean that you should not use any of these instructions for getting exactly the printed document that you want. By all means, do. But you will have to hide those commands from Hyperlatex using the escape mechanisms.

And you should learn about the commands that allow you to generate much more natural HTML than any plain LaTeX-to-HTML converter could. For instance, `\pageref` is not understood by the Hyperlatex converter, because HTML has no pages. Cross-references are best made using the `\link` command.

The following sections explain in detail what you can and cannot do in Hyperlatex.

Practically all aspects of the generated output can be customized, see Section 10.

5 A LaTeX subset — Getting started

Starting with this section, we take a stroll through the LaTeX-book [1], explaining all features that Hyperlatex understands, additional features of Hyperlatex, and some missing features. For the LaTeX output the general rule is that *no LaTeX command has been changed*. If a familiar LaTeX command is listed in this manual, it is understood both by LaTeX and the Hyperlatex converter, and its LaTeX meaning is the familiar one. If it is not listed here, you can still use it by escaping into TeX-only mode, but it will then have effect in the printed output only.

5.1 Preparing an input file

There are ten characters that LaTeX and Hyperlatex treat specially:

```
\  {  }  ~  ^  _  #  $  %  &
```

To typeset one of these, use

```
\back   \{   \}  \~{}  \^{}  \_  \#  \$  \%  \&
```

(Note that `\back` is different from the `\backslash` command of LaTeX. `\backslash` can only be used in math mode and looks like this: $\backslash$, while `\back` can be used in any mode and looks like this: \back.)

Sometimes it is useful to turn off the special meaning of some of these ten characters. For instance, when writing documentation about programs in C, it might be useful to be able to write `some_variable` instead of always having to type `some\_variable`. This can be achieved with the `\NotSpecial` command.

In principle, all other characters simply typeset themselves. This has to be taken with a grain of salt, though. LaTeX still obeys ligatures, which turns `ffi` into 'ffi', and some characters, like `>`, do not resemble themselves in some fonts (`>` looks like ¿ in roman font). The only characters for which this is critical are `<`, `>`, and `|`. Better use them in a typewriter-font. Note that `?‘` and `!‘` are ligatures in any font and are displayed and printed as ¿ and ¡.

Like LaTeX, the Hyperlatex converter understands that an empty line indicates a new paragraph. You can achieve the same effect using the command `\par`.

5.2 Dashes and Quotation marks

Hyperlatex translates a sequence of two dashes `--` into a single dash, and a sequence of three dashes `---` into two dashes `--`. The quotation mark sequences `''` and ‘‘ are translated into simple quotation marks ¨.

5.3 Simple text generating commands

The following simple LaTeX macros are implemented in Hyperlatex:

- `\LaTeX` produces LaTeX.
- `\TeX` produces TeX.
- `\LaTeXe` produces LaTeX 2_ε.
- `\ldots` produces three dots ...
- `\today` produces April 1, 2010—although this might depend on when you use it...

5.4 Emphasizing Text

You can emphasize text using `\emph` or the old-style command `\em`. It is also possible to use the construction `\begin{em}` ... `\end{em}`.

5.5 Preventing line breaks

The `~` is a special character in Hyperlatex, and is replaced by the HTML-tag for "non-breakable space".

As we saw before, you can typeset the `~` character by typing `\~{}`. This is also the way to go if you need the `~` in an argument to an HTML command that is processed by Hyperlatex, such as in the *URL*-argument of `\xlink`.

You can also use the `\mbox` command. It is implemented by replacing all sequences of white space in the argument by a single `~`. Obviously, this restricts what you can use in the argument. (Better don't use any math mode material in the argument.)

5.6 Footnotes

The footnotes in your document will be collected together and output as a separate section or chapter right at the end of your document. You can specify a different location using the `\htmlfootnotes` command, which has to come *after* all `\footnote` commands in the document.

5.7 Formulas

There is no *math mode* in HTML. (The proposed standard HTML 3 contained a math mode, but has been withdrawn. HTML-browsers that will understand math do not seem to become widely available in the near future.)

Hyperlatex understands the `$` sign delimiting math mode as well as `\(` and `\)`. Subscripts and superscripts produced using `_` and `^` are understood.

Hyperlatex now has a simply textual implementation of many common math mode commands, so simple formulas in your text should be converted to some textual representation. If you are not satisfied with that representation, you can use the `\math` command:

`\math[`HTML-*version*`]{`LaTeX-*version*`}`

In LaTeX, this command typesets the *LaTeX-version*, which is read in math mode (with all special characters enabled, if you have disabled some using `\NotSpecial`). Hyperlatex typesets the optional argument if it is present, or otherwise the LaTeX-version.

If, for instance, you want to typeset the ith element (`the \math{i}th element`) of an array as a_i in LaTeX, but as `a[i]` in HTML, you can use

```
\math[\code{a[i]}]{a_{i}}
```

By default, Hyperlatex sets all math mode material in italic, as is common practice in typesetting mathematics: "Given n points..." Sometimes, however, this looks bad, and you can turn it off by using `\htmlmathitalic{0}` (turn it back on using `\htmlmathitalic{1}`). For instance: 2^n, but H^{-1}. (In the long run, Hyperlatex should probably recognize different concepts in math mode and select the right font for each.)

It takes a bit of care to find the best representation for your formula. This is an example of where any mechanical LaTeX-to-HTML converter must fail—I hope that Hyperlatex's `\math` command will help you produce a good-looking and functional representation.

You could create a bitmap for a complicated expression, but you should be aware that bitmaps eat transmission time, and they only look good when the resolution of the browser is nearly the same as the resolution at which the bitmap has been created, which is not a realistic assumption. In many situations, there are easier solutions: If x_i is the ith element of an array, then I would rather write it as `x[i]` in HTML. If it's a variable in a program, I'd probably write `xi`. In another context, I might want to write *x_i*. To write Pythagoras's theorem, I might simply use `a^2 + b^2 = c^2`, or maybe `a*a + b*b = c*c`. To express "For any $\varepsilon > 0$ there is a $\delta > 0$ such that for $|x - x_0| < \delta$ we have $|f(x) - f(x_0)| < \varepsilon$" in HTML, I would write "For any *eps* `>` *0* there is a *delta* `>` *0* such that for `|`*x-x0*`|` `<` *delta* we have `|`*f(x)-f(x0)*`|` `<` *eps*."

5.8 Ignorable input

The percent character `%` introduces a comment in Hyperlatex. Everything after a `%` to the end of the line is ignored, as well as any white space on the beginning of the next line.

5.9 Document class

The `\documentclass` (or alternatively `\documentstyle`) and `\usepackage` commands are interpreted by Hyperlatex to select additional package files with definitions for commands particular to that class or package.

5.10 Title page

The `\title`, `\author`, `\date`, and `\maketitle` commands and the `abstract` environment are all understood by Hyperlatex. The `\thanks` command currently simply generates a footnote. This is often not the right way to format it in an HTML-document, use conditional translation to make it better (Section 6).

5.11 Sectioning

The sectioning commands `\chapter`, `\section`, `\subsection`, `\subsubsection`, `\paragraph`, and `\subparagraph` are recognized by Hyperlatex and used to partition the document into nodes. You can also use the starred version and the optional argument for the sectioning commands. The optional argument will be used for node titles and in menus. Hyperlatex can number your sections if you set the counter `secnumdepth` appropriately. The default is not to number any sections. For instance, if you use this in the preamble

```
\setcounter{secnumdepth}{3}
```

chapters, sections, subsections, and subsubsections will be numbered.

Note that you cannot use `\label`, `\index`, nor many other commands that generate HTML-markup in the argument to the sectioning commands. If you want to label a section, or put it in the index, use the `\label` or `\index` command *after* the `\section` command.

You will probably sooner or later want to start an HTML node without a heading, or maybe with a bitmap before the main heading. This can be done by leaving the argument to the sectioning command empty. (You can still use the optional argument to set the title of the HTML node.)

Do not use *only* a bitmap as the section title in sectioning commands. The right way to start a document with an image only is the following:

```
\T\section{An example of a node starting with an image}
\W\section[Node with Image]{}
\W\begin{center}\htmlimg{theimage.png}{}\end{center}
\W\htmlheading[1]{An example of a node starting with an image}
```

The `\htmlheading` command creates a heading in the HTML output just as `\section` does, but without starting a new node. The optional argument has to be a number from 1 to 6, and specifies the level of the heading (in `article` style, level 1 corresponds to `\section`, level 2 to `\subsection`, and so on).

You can use the commands `\protect` and `\noindent`. They will be ignored in the HTML-version.

5.12 Displayed material

The `center`, `quote`, `quotation`, and `verse` environment are implemented.

To make lists, you can use the `itemize`, `enumerate`, and `description` environments. You *cannot* specify an optional argument to `\item` in `itemize` or `enumerate`, and you *must* specify one for `description`.

All these environments can be nested.

The `\\` command is recognized, with and without `*`. You can use the optional argument to `\\`, but it will be ignored.

There is also a `menu` environment, which looks like an `itemize` environment, but is somewhat denser since the space between items has been reduced. It is only meant for single-line items.

Hyperlatex understands the math display environments `\[`, `\]`, `displaymath`, `equation`, and `equation*`.

6 Conditional Compilation: Escaping into one mode

In many situations you want to achieve slightly (or maybe even drastically) different behavior of the LaTeX code and the Html-output. Hyperlatex offers several different ways of letting your document depend on the mode.

6.1 LaTeX versus Html mode

The easiest way to put a command or text in your document that is only included in one of the two output modes it by using a `\texonly` or `\htmlonly` command. They ignore their argument, if in the wrong mode, and otherwise simply expand it:

```
We are now in \texonly{\LaTeX}\htmlonly{HTML}-mode.
```

In cases such as this you can simplify the notation by using the `\texorhtml` command, which has two arguments:

```
We are now in \texorhtml{\LaTeX}{HTML}-mode.
```

Another possibility is by prefixing a line with `\T` or `\W`. `\T` acts like a comment in Html-mode, and as a noop in LaTeX-mode, and for `\W` it is the other way round:

```
We are now in
\T \LaTeX-mode.
\W HTML-mode.
```

The last way of achieving this effect is useful when there are large chunks of text that you want to skip in one mode—a Html-document might skip a section with a detailed mathematical analysis, a LaTeX-document will not contain a node with lots of hyperlinks to other documents. This can be done using the `iftex` and `ifhtml` environments:

```
We are now in
\begin{iftex}
  \LaTeX-mode.
\end{iftex}
\begin{ifhtml}
  HTML-mode.
\end{ifhtml}
```

In LaTeX, commands that are defined inside an enviroment are "forgotten" at the end of the environment. So LaTeX commands defined inside a `iftex` environment are defined, but then immediately forgotten by LaTeX. A simple trick to avoid this problem is to use the following idiom:

```
\W\begin{iftex}
... command definitions
\W\end{iftex}
```

Now the command definitions are correctly made in the Latex, but not in the Html version.

Instead of the `iftex` environment, you can also use the `tex` environment. It is different from `iftex` only if you have used `\NotSpecial` in the preamble.

The environment `latexonly` has been provided as a service to `latex2html` users. Its effect is the same as `iftex`.

6.2 Ignoring more input

The contents of the `comment` environment is ignored.

6.3 Flags — more on conditional compilation

You can also have sections of your document that are included depending on the setting of a flag:

```
\begin{ifset}{flag}
  Flag flag is set!
\end{ifset}

\begin{ifclear}{flag}
  Flag flag is not set!
\end{ifset}
```

A flag is simply the name of a TEX command. A flag is considered set if the command is defined and its expansion is neither empty nor the single character "0" (zero).

You could for instance select in the preamble which parts of a document you want included (in this example, parts A and D are included in the processed document):

```
\newcommand{\IncludePartA}{1}
\newcommand{\IncludePartB}{0}
\newcommand{\IncludePartC}{0}
\newcommand{\IncludePartD}{1}
  ...
\begin{ifset}{IncludePartA}
  Text of part A
\end{ifset}
  ...
\begin{ifset}{IncludePartB}
  Text of part B
\end{ifset}
  ...
\begin{ifset}{IncludePartC}
  Text of part C
\end{ifset}
  ...
\begin{ifset}{IncludePartD}
  Text of part D
\end{ifset}
  ...
```

Note that it is permitted to redefine a flag (using `\renewcommand`) in the document. That is particularly useful if you use these environments in a macro.

7 Carrying on

In this section we continue to Chapter 3 of the LATEX-book, dealing with more advanced topics.

7.1 Changing the type style

Hyperlatex understands the following physical font specifications of LATEX 2ε:

- `\textbf` for **bold**
- `\textit` for *italic*
- `\textsc` for SMALL CAPS

- `\texttt` for `typewriter`
- `\underline` for underline

In LaTeX 2$_\varepsilon$ font changes are cumulative—`\textbf{\textit{BoldItalic}}` typesets the text in a bold italic font. Different Html browsers will display different things.

The following old-style commands are also supported:

- `\bf` for **bold**
- `\it` for *italic*
- `\tt` for `typewriter`

So you can write

```
{\it italic text}
```

but also

```
\textit{italic text}
```

You can use `\/` to separate slanted and non-slanted fonts (it will be ignored in the Html-version).

Hyperlatex complains about any other LaTeX commands for font changes, in accordance with its general philosophy. If you do believe that, say, `\sf` should simply be ignored, you can easily ask for that in the preamble by defining:

```
\W\newcommand{\sf}{}
```

Both LaTeX and Html encourage you to express yourself in terms of *logical concepts* instead of visual concepts. (Otherwise, you wouldn't be using Hyperlatex but some Wysiwyg editor to create Html.) In fact, Html defines tags for *logical* markup, whose rendering is completely left to the user agent (Html client).

The Hyperlatex package defines a standard representation for these logical tags in LaTeX—you can easily redefine them if you don't like the standard setting.

The logical font specifications are:

- `\cit` for *citations*.
- `\code` for `code`.
- `\dfn` for *defining a term*.
- `\em` and `\emph` for *emphasized text*.
- `\file` for *file.names*.
- `\kbd` for `keyboard input`.
- `\samp` for `sample input`.
- `\strong` for **strong emphasis**.
- `\var` for *variables*.

7.2 Changing type size

Hyperlatex understands the LaTeX declarations to change the type size. The Html font changes are relative to the Html node's *basefont size*. (`\normalfont` being the basefont size, `\large` begin the basefont size plus one etc.)

7.3 Symbols from other languages

Hyperlatex recognizes all of LaTeX's commands for making accents. However, only few of these are are available in Html. Hyperlatex will make a Html-entity for the accents in iso Latin 1, but will reject all other accent sequences. The command `\c` can be used to put a cedilla on a letter 'c' (either case), but on no other letter. So the following is legal

```
Der K{\"o}nig sa\ss{} am wei{\ss}en Strand von Cura\c{c}ao und
nippte an einer Pi\~{n}a Colada \ldots
```

and produces

Der König saß am weißen Strand von Curaçao und nippte an einer Piña Colada ...

Not available in HTML are `Ji{\v r}\'{\i}`, or `Erd\H{o}s`. (You can tell Hyperlatex to simply typeset all these letters without the accent by using the following in the preamble:

```
\newcommand{\HlxIllegalAccent}[2]{#2}
```

Hyperlatex also understands the following symbols:

œ	\oe	å	\aa	¿	?‘
Œ	\OE	Å	\AA	¡	!‘
æ	\ae	ø	\o	ß	\ss
Æ	\AE	Ø	\O		
§	\S	©	\copyright		
¶	\P	£	\pounds		

`\quad` and `\qquad` produce some empty space.

7.4 Defining commands and environments

Hyperlatex understands definitions of new commands with the LaTeX-instructions `\newcommand` and `\newenvironment`. `\renewcommand` and `\renewenvironment` are understood as well (Hyperlatex makes no attempt to test whether a command is actually already defined or not.) The optional parameter of LaTeX 2_ε is also implemented.

If you use `\providecommand`, Hyperlatex checks whether the command is already defined. The command is ignored if the command already exists.

Note that it is not possible to redefine a Hyperlatex command that is *hard-coded* in Emacs lisp inside the Hyperlatex converter. So you could redefine the command `\cite` or the `verse` environment, but you cannot redefine `\T`. (But you can redefine most of the commands understood by Hyperlatex, namely all the ones defined in *siteinit.hlx*.)

Some basic examples:

```
\newcommand{\Html}{\textsc{Html}}

\T\newcommand{\bad}{$\surd$}
\W\newcommand{\bad}{\htmlimg{badexample_bitmap.xbm}{BAD}}

\newenvironment{badexample}{\begin{description}
  \item[\bad]}{\end{description}}

\newenvironment{smallexample}{\begingroup\small
            \begin{example}}{\end{example}\endgroup}
```

Command definitions made by Hyperlatex are global, their scope is not restricted to the enclosing environment. If you need to restrict their scope, use the `\begingroup` and `\endgroup` commands to create a scope (in Hyperlatex, this scope is completely independent of the LaTeX-environment scoping).

Note that Hyperlatex does not tokenize its input the way TeX does. To evaluate a macro, Hyperlatex simply inserts the expansion string, replaces occurrences of `#1` to `#9` by the arguments, strips one `#` from strings of at least two `#`'s, and then reevaluates the whole. Problems may occur when you try to use `%`, `\T`, or `\W` in the expansion string. Better don't do that.

7.5 Theorems and such

The `\newtheorem` command declares a new "theorem-like" environment. The optional arguments are allowed as well (but ignored unless you customize the appearance of the environment to use Hyperlatex's counters).

```
\newtheorem{guess}[theorem]{Conjecture}[chapter]
```

7.6 Figures and other floating bodies

You can use `figure` and `table` environments and the `\caption` command. They will not float, but will simply appear at the given position in the text. No special space is left around them, so put a `center` environment in a figure. The `table` environment is mainly used with the `tabular` environment below. You can use the `\caption` command to place a caption. The starred versions `table*` and `figure*` are supported as well.

7.7 Lining it up in columns

The `tabular` environment is available in Hyperlatex.

Many column types are now supported, and even `\newcolumntype` is available. The `|` column type specifier is silently ignored. You can force borders around your table (and every single cell) by using `\xmlattributes*{table}{border}` immediately before your `tabular` environment. You can use the `\multicolumn` command. `\hline` is understood and ignored.

The `\htmlcaption` has to be used right after the `\begin{tabular}`. It sets the caption for the HTML table. (In HTML, the caption is part of the `tabular` environment. However, you can as well use `\caption` outside the environment.)

If you have made the `&` character non-special, you can use the macro `\htmltab` as a replacement.

Here is an example:

```
\begin{table}[htp]
\T\caption{Keyboard shortcuts for \textit{Ipe}}
\begin{center}
\begin{tabular}{|l|lll|}
\htmlcaption{Keyboard shortcuts for \textit{Ipe}}
\hline
            & Left Mouse      & Middle Mouse  & Right Mouse      \\
\hline
Plain       & (start drawing) & move          & select           \\
Shift       & scale           & pan           & select more      \\
Ctrl        & stretch         & rotate        & select type      \\
Shift+Ctrl  &                 &               & select more type \T\\
\hline
\end{tabular}
\end{center}
\end{table}
```

The example is typeset as in Table 2.

Note that the `netscape` browser treats empty fields in a table specially. If you don't like that, put a single `~` in that field.

A more complicated example is in Table 3

To create certain effects you can employ the `\xmlattributes` command, as for the example in Table 4 As an alternative for creating cells spanning multiple rows, you could check out the `multirow` package in the *contrib* directory.

Table 1: Keyboard shortcuts for *Ipe*

	Left Mouse	Middle Mouse	Right Mouse
Plain	(start drawing)	move	select
Shift	scale	pan	select more
Ctrl	stretch	rotate	select type
Shift+Ctrl			select more type

Table 2:

type	*style*	
smart	red	short
rather silly	puce	tall

Table 3:

7.8 Tabbing

A weak implementation of the tabbing environment is available if the HTML level is 3.2 or higher. It works using HTML `<TABLE>` markup, which is a bit of a hack, but seems to work well for simple tabbing environments.

The only commands implemented are `\=`, `\>`, `\\`, and `\kill`.

Here is an example:

while $n < (42 * x/y)$
    if n odd
        output n
    increment n
return `TRUE`

7.9 Simulating typed text

The `verbatim` environment and the `\verb` command are implemented. The starred varieties are currently not implemented. (The implementation of the `verbatim` environment is not the standard LaTeX implementation, but the one from the `verbatim` package by Rainer Schöpf).

Furthermore, there is another, new environment `example`. `example` is also useful for including program listings or code examples. Like `verbatim`, it is typeset in a typewriter font with a fixed character pitch, and obeys spaces and line breaks. But here ends the similarity, since `example` obeys the special characters `\`, `{`, `}`, and `%`. You can still use font changes within an `example` environment, and you can also place hyperlinks there. Here is an example:

```
To clear a flag, use
\begin{example}
  {\back}clear\{\var{flag}\}
\end{example}
```

(The `example` environment is very similar to the `alltt` environment of the `alltt` package. The difference is that example obeys the `%` character.)

gnats	gram	$13.65
	each	.01
gnu	stuffed	92.50
emu		33.33
armadillo	frozen	8.99

Table 4:

8 Moving information around

In this section we deal with questions related to cross referencing between parts of your document, and between your document and the outside world. This is where Hyperlatex gives you the power to write natural HTML documents, unlike those produced by any LaTeX converter. A converter can turn a reference into a hyperlink, but it will have to keep the text more or less the same. If we wrote "More details can be found in the classical analysis by Harakiri [8]", then a converter may turn "[8]" into a hyperlink to the bibliography in the HTML document. In handwritten HTML, however, we would probably leave out the "[8]" altogether, and make the *name* "Harakiri" a hyperlink.

The same holds for references to sections and pages. The Ipe manual says "This parameter can be set in the configuration panel (Section 11.1)". A converted document would have the "11.1" as a hyperlink. Much nicer HTML is to write "This parameter can be set in the configuration panel", with "configuration panel" a hyperlink to the section that describes it. If the printed copy reads "We will study this more closely on page 42," then a converter must turn the "42" into a symbol that is a hyperlink to the text that appears on page 42. What we would really like to write is "We will later study this more closely," with "later" a hyperlink—after all, it makes no sense to even allude to page numbers in an HTML document.

The Ipe manual also says "Such a file is at the same time a legal Encapsulated Postscript file and a legal LaTeX file—see Section 13." In the HTML copy the "Such a file" is a hyperlink to Section 13, and there's no need for the "—see Section 13" anymore.

8.1 Cross-references

You can use the `\label{`*label*`}` command to attach a *label* to a position in your document. This label can be used to create a hyperlink to this position from any other point in the document. This is done using the `\link` command:

`\link{`*anchor*`}{`*label*`}`

This command typesets anchor, expanding any commands in there, and makes it an active hyperlink to the position marked with *label*:

```
This parameter can be set in the
\link{configuration panel}{sect:con-panel} to influence ...
```

The `\link` command does not do anything exciting in the printed document. It simply typesets the text *anchor*. If you also want a reference in the LaTeX output, you will have to add a reference using `\ref` or `\pageref`. Sometimes you will want to place the reference directly behind the *anchor* text. In that case you can use the optional argument to `\link`:

```
This parameter can be set in the
\link{configuration
  panel}[~(Section~\ref{sect:con-panel})]{sect:con-panel} to
influence ...
```

The optional argument is ignored in the HTML-output.

The starred version `\link*` suppresses the anchor in the printed version, so that we can write

```
We will see \link*{later}[in Section~\ref{sl}]{sl}
how this is done.
```

It is very common to use `\ref{`*label*`}` or `\pageref{`*label*`}` inside the optional argument, where *label* is the label set by the link command. In that case the reference can be abbreviated as `\Ref` or `\Pageref` (with capitals). These definitions are already active when the optional arguments are expanded, so we can write the example above as

```
We will see \link*{later}[in Section~\Ref]{sl}
how this is done.
```

Often this format is not useful, because you want to put it differently in the printed manual. Still, as long as the reference comes after the `\link` command, you can use `\Ref` and `\Pageref`.

```
\link{Such a file}{ipe-file} is at
the same time ... a legal \LaTeX{}
file\texonly{---see Section~\Ref}.
```

Note that when you use LaTeX's `\ref` command, the label does not mark a *position* in the document, but a certain *object*, like a section, equation etc. It sometimes requires some care to make sure that both the hyperlink and the printed reference point to the right place, and sometimes you will have to place the label twice. The HTML-label tends to be placed *before* the interesting object—a figure, say—, while the LaTeX-label tends to be put *after* the object (when the `\caption` command has set the counter for the label). In such cases you can use the new `Label` environment. It puts the HTML-label at the beginning of the text, but the latex label at the end. For instance, you can correctly refer to a figure using:

```
\begin{figure}
  \begin{Label}{fig:wonderful}
    %% here comes the figure itself
    \caption{Isn't it wonderful?}
  \end{Label}
\end{figure}
```

A `\link{fig:wonderful}` will now correctly lead to a position immediatly above the figure, while a `Figure~\ref{fig:wonderful}` will show the correct number of the figure.

A special case occurs for section headings. Always place labels *after* the heading. In that way, the LaTeX reference will be correct, and the Hyperlatex converter makes sure that the link will actually lead to a point directly before the heading—so you can see the heading when you follow the link.

After a while, you may notice that in certain situations Hyperlatex has a hard time dealing with a label. The reason is that although it seems that a label marks a *position* in your node, the HTML-tag to set the label must surround some text. If there are other HTML-tags in the neighborhood, Hyperlatex may not find an appropriate contents for this container and has to add a space in that position (which may sometimes mess up your formatting). In such cases you can help Hyperlatex by using the `Label` environment, showing Hyperlatex how to make a label tag surrounding the text in the environment.

Note that Hyperlatex uses the argument of a `\label` command to produce a mnemonic HTML-label in the HTML file, but only if it is a legal URL.

In certain situations—for instance when it is to be expected that documents are going to be printed directly from web pages, or when you are porting a LaTeX-document to Hyperlatex—it makes sense to mimic the standard way of referencing in LaTeX, namely by simply using the

number of a section as the anchor of the hyperlink leading to that section. Therefore, the `\ref` command is implemented in Hyperlatex. It's default definition is

```
\newcommand{\ref}[1]{\link{\htmlref{#1}}{#1}}
```

The `\htmlref` command used here simply typesets the counter that was saved by the `\label` command. So I can simply write

```
see Section~\ref{sec:cross-references}
```

to refer to the current section: see Section 8.1.

8.2 Links to external information

You can place a hyperlink to a given *URL* (Universal Resource Locator) using the `\xlink` command. Like the `\link` command, it takes an optional argument, which is typeset in the printed output only:

```
\xlink{anchor}{URL}
\xlink{anchor}[printed reference]{URL}
```

In the HTML-document, *anchor* will be an active hyperlink to the object *URL*. In the printed document, *anchor* will simply be typeset, followed by the optional argument, if present. A starred version `\xlink*` has the same function as for `\link`.

If you need to use a `~` in the *URL* of an `\xlink` command, you have to escape it as `\~{}` (the *URL* argument is an evaluated argument, so that you can define macros for common *URL*'s).

8.3 Links into your document

The Hyperlatex converter automatically partitions your document into HTML-nodes. These nodes are simply numbered sequentially. Obviously, the resulting URL's are not useful for external references into your document—after all, the exact numbers are going to change whenever you add or delete a section, or when you change the `htmldepth`.

If you want to allow links from the outside world into your new document, you will have to give that HTML node a mnemonic name that is not going to change when the document is revised.

This can be done using the `\xname{`*name*`}` command. It assigns the mnemonic name *name* to the *next* node created by Hyperlatex. This means that you ought to place it *in front of* a sectioning command. The `\xname` command has no function for the LaTeX-document. No warning is created if no new node is started in between two `\xname` commands.

The argument of `\xname` is not expanded, so you should not escape any special characters (such as `_`). On the other hand, if you reference it using `\xlink`, you will have to escape special characters.

Here is an example: This section "Links into your document" in this document starts as follows.

```
\xname{hyperlatex_extlinks}
\subsection{Links into your document}
\label{sec:into-hyperlinks}
The Hyperlatex converter automatically...
```

This HTML-node can be referenced inside this document with

```
\link{External links}{sec:into-hyperlinks}
```

and both inside and outside this document with

```
\xlink{External links}{hyperlatex\_extlinks.html}
```

If you want to refer to a location *inside* an HTML-node, you need to make sure that the label you place with `\label` is a legal XML `id` attribute. In other words, it must start with a letter, and consist solely of characters from the set

```
a-z A-Z 0-9 - _ . :
```

All labels that contain other characters are replaced by an automatically created numbered label by Hyperlatex.

The previous paragraph starts with

```
\label{label_urls}
\cindex[label]{\verb+\label+}
If you want to refer to a location \emph{inside} an \Html-node,...
```

You can therefore refer to that position from any document using

```
\xlink{refer to that position}{hyperlatex\_extlinks.html\#label\_urls}
```

(Note that `#` and `_` have to be escaped in the `\xlink` command.)

8.4 Bibliography and citation

Hyperlatex understands the `thebibliography` environment. Like LaTeX, it creates a chapter or section (depending on the document class) titled "References". The `\bibitem` command sets a label with the given *cite key* at the position of the reference. This means that you can use the `\link` command to define a hyperlink to a bibliography entry.

The command `\Cite` is defined analogously to `\Ref` and `\Pageref` by `\link`. If you define a bibliography like this

```
\begin{thebibliography}{99}
   \bibitem{latex-book}
   Leslie Lamport, \cit{\LaTeX: A Document Preparation System,}
   Addison-Wesley, 1986.
\end{thebibliography}
```

then you can add a reference to the LaTeX-book as follows:

```
... we take a stroll through the
\link{\LaTeX-book}[~\Cite]{latex-book}, explaining ...
```

Furthermore, the command `\htmlcite` generates the printed citation itself (in our case, `\htmlcite{latex-book}` would generate "[1]"). The command `\cite` is approximately implemented as `\link{\htmlcite{#1}}{#1}`, so you can use it as usual in LaTeX, and it will automatically become an active hyperlink, as in "[1]". (The actual definition allows you to use multiple cite keys in a single `\cite` command.)

Hyperlatex also understands the `\bibliographystyle` command (which is ignored) and the `\bibliography` command. It reads the *.bbl* file, inserts its contents at the given position and proceeds as usual. Using this feature, you can include bibliographies created with BibTeX in your HTML-document! It would be possible to design a WWW-server that takes queries into a BibTeX database, runs BibTeX and Hyperlatex to format the output, and sends back an HTML-document.

The formatting of the bibliography can be customized by redefining the bibliography environment `thebibliography` and the Hyperlatex macro `\htmlbibitem`. The default definitions are

```
\newenvironment{thebibliography}[1]%
   {\chapter{References}\begin{description}}{\end{description}}
\newcommand{\htmlbibitem}[2]{\label{#2}\item[{[#1]}]}
```

If you use BibTEX to generate your bibliographies, then you will probably want to incorporate hyperlinks into your *.bib* files. No problem, you can simply use `\xlink`. But what if you also want to use the same *.bib* file with other (vanilla) LATEX files, which do not define the `\xlink` command? What if you want to share your *.bib* files with colleagues around the world who do not know about Hyperlatex?

One way to solve this problem is by using the BibTEX `@preamble` command. For instance, you put this in your BibTEX file:

```
@preamble("
  \providecommand{\url}[1]{#1}
  ")
```

Then you can put a *URL* into the *note* field of a BibTEX entry as follows:

```
note = "\url{ftp://nowhere.com/paper.ps}"
```

Now your BibTEX file will work fine with any LATEX documents, typesetting the *URL* as it is.

In your Hyperlatex source, however, you could define `\url` any way you like, such as:

```
\newcommand{\url}[1]{\xlink{#1}{#1}}
```

This will turn the *note* field into an active hyperlink to the document in question.

8.5 Splitting your input

The `\input` command is implemented in Hyperlatex. The subfile is inserted into the main document, and typesetting proceeds as usual. You have to include the argument to `\input` in braces. `\include` is understood as a synonym for `\input` (the command `\includeonly` is ignored by Hyperlatex).

8.6 Making an index or glossary

The Hyperlatex converter understands the `\index` command. It collects the entries specified, and you can include a sorted index using `\htmlprintindex`. This index takes the form of a menu with hyperlinks to the positions where the original `\index` commands where located.

You may want to specify a different sort key for an index intry. If you use the index processor `makeindex`, then this can be achieved in LATEX by specifying `\index{sortkey@entry}`. This syntax is also understood by Hyperlatex. The entry

```
\index{index@\verb+\index+}
```

will be sorted like "`index`", but typeset in the index as "`\verb+\index+`".

However, not everybody can use `makeindex`, and there are other index processors around. To cater for those other index processors, Hyperlatex defines a second index command `\cindex`, which takes an optional argument to specify the sort key. (You may also like this syntax better than the `\index` syntax, since it is more in line with the general LATEX-syntax.) The above example would look as follows:

```
\cindex[index]{\verb+\index+}
```

The *hyperlatex.sty* style defines `\cindex` such that the intended behavior is realized if you use the index processor `makeindex`. If you don't, you will have to consult your *Local Guide* and redefine `\cindex` appropriately. (That may be a bit tricky—ask your local TEX guru for help.)

The index in this manual was created using `\cindex` commands in the source file, the index processor `makeindex` and the following code (more or less):

```
\W \section*{Index}
\W \htmlprintindex
\T \input{hyperlatex.ind}
```

You can generate a prettier index format more similar to the printed copy by using the `makeidx` package donated by Sebastian Erdmann. Include it using

```
\W \usepackage{makeidx}
```

in the preamble.

8.7 Screen Output

You can use `\typeout` to print a message while your file is being processed.

9 Designing it yourself

In this section we discuss the commands used to make things that only occur in HTML-documents, not in printed papers. Practically all commands discussed here start with `\html`, indicating that the command has no effect whatsoever in LaTeX.

9.1 Making menus

The `\htmlmenu` command generates a menu for the subsections of a section. Its argument is the depth of the desired menu. If you use `\htmlmenu{2}` in a subsection, say, you will get a menu of all subsubsections and paragraphs of this subsection.

If you use this command in a section, no automatic menu for this section is created.

A typical application of this command is to put a "master menu" (the analog of a table of contents) in the top node, containing all sections of all levels of the document. This can be achieved by putting `\htmlmenu{6}` in the text for the top node.

You can create a menu for a section other than the current one by passing the number of that section as the optional argument, as in `\htmlmenu[0]{6}`, which creates a full table of contents. (The optional argument uses Hyperlatex's internal numbering–not very useful except for the top node, which is always number 0.)

Some people like to close off a section after some subsections of that section, somewhat like this:

```
\section{S1}
text at the beginning of section S1
  \subsection{SS1}
  \subsection{SS2}
closing off S1 text

\section{S2}
```

This is a bit of a problem for Hyperlatex, as it requires the text for any given node to be consecutive in the file. A workaround is the following:

```
\section{S1}
text at the beginning of section S1
\htmlmenu{1}
\texonly{\def\savedtext}{closing off S1 text}
  \subsection{SS1}
```

```
  \subsection{SS2}
\texonly{\bigskip\savedtext}

\section{S2}
```

9.2 Rulers and images

The command `\htmlrule` creates a horizontal rule spanning the full screen width at the current position in the HTML-document.

The command `\htmlimg{`*URL*`}{`*Alt*`}` makes an inline bitmap with the given *URL*. If the image cannot be rendered, the alternative text *Alt* is used. Both *URL* and *Alt* arguments are evaluated arguments, so that you can define macros for common *URL*'s (such as your home page). That means that if you need to use a special character (`~` is quite common), you have to escape it (as `\~{}` for the `~`).

This is what I use for figures in the Ipe Manual that appear in both the printed document and the HTML-document:

```
\begin{figure}
  \caption{The Ipe window}
  \begin{center}
    \texorhtml{\Ipe{window.ipe}}{\htmlimg{window.png}}
  \end{center}
\end{figure}
```

(`\Ipe` is the command to include "Ipe" figures.)

9.3 Adding raw Xml

Hyperlatex provides a number of ways to access the XML-tag level.

The `\xmlent{`*entity*`}` command creates the XML entity description `&`*entity*`;`. It is useful if you need symbols from the ISO Latin 1 alphabet which are not predefined in Hyperlatex. You could, for instance, define a macro for the fraction 1/4 as follows:

```
\T \newcommand{\onequarter}{$1/4$}
\W \newcommand{\onequarter}{\xmlent{##188}}
```

The most basic command is `\xml{`*tag*`}`, which creates the XML tag `<`*tag*`>`. This command is used in the definition of most of Hyperlatex's commands and environments, and you can use it yourself to achieve effects that are not available in Hyperlatex directly. Note that `\xml` looks up any attributes for the tag that may have been set with `\xmlattributes`. If you want to avoid this, use the starred version `\xml*`.

Finally, the `rawxml` environment allows you to write plain XML, if you so desire. Everything between `\begin{rawxml}` and `\end{rawxml}` will simply be included literally in the XML output. Alternatively, you can include a file of XML literally using `\xmlinclude`.

9.4 Turning TEX into bitmaps

Sometimes the only sensible way to represent some LATEX concept in an HTML-document is by turning it into a bitmap. Hyperlatex has an environment `image` that does exactly this: In the HTML-version, it is turned into a reference to an inline bitmap (just like `\htmlimg`). In the LATEX-version, the `image` environment is equivalent to a `tex` environment. Note that running the Hyperlatex converter doesn't create the bitmaps yet, you have to do that in an extra step as described below.

The `image` environment has three optional and one required arguments:

```
\begin{image}[attr][resolution][font_resolution]{name}
  TEX material ...
\end{image}
```

For the LaTeX-document, this is equivalent to

```
\begin{tex}
  TEX material ...
\end{tex}
```

For the Html-version, it is equivalent to

```
\htmlimg{name.png}{}
```

The optional *attr* parameter can be used to add Html attributes to the `img` tag being created. The other two parameters, *resolution* and *font_resolution*, are used when creating the `png`-file. They default to 100 and 300 dots per inch.

Here is an example:

```
\W\begin{quote}
\begin{image}{eqn1}
  \[
  \sum_{i=1}^{n} x_{i} = \int_{0}^{1} f
  \]
\end{image}
\W\end{quote}
```

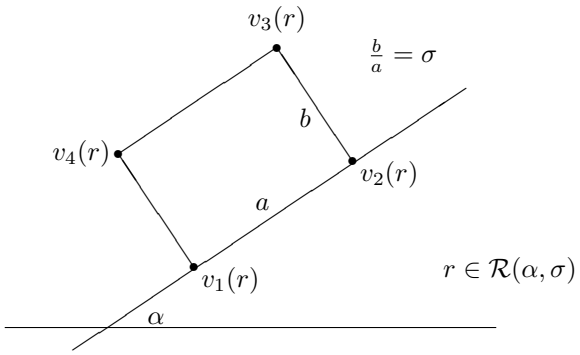
produces the following output:

$$\sum_{i=1}^{n} x_i = \int_0^1 f$$

We could as well include a picture environment. The code

```
\begin{center}
  \begin{image}[][80]{boxes}
    \setlength{\unitlength}{0.1mm}
    \begin{picture}(700,500)
      \put(40,-30){\line(3,2){520}}
      \put(-50,0){\line(1,0){650}}
      \put(150,5){\makebox(0,0)[b]{$\alpha$}}
      \put(200,80){\circle*{10}}
      \put(210,80){\makebox(0,0)[lt]{$v_{1}(r)$}}
      \put(410,220){\circle*{10}}
      \put(420,220){\makebox(0,0)[lt]{$v_{2}(r)$}}
      \put(300,155){\makebox(0,0)[rb]{$a$}}
      \put(200,80){\line(-2,3){100}}
      \put(100,230){\circle*{10}}
      \put(100,230){\line(3,2){210}}
      \put(90,230){\makebox(0,0)[r]{$v_{4}(r)$}}
      \put(410,220){\line(-2,3){100}}
      \put(310,370){\circle*{10}}
      \put(355,290){\makebox(0,0)[rt]{$b$}}
      \put(310,390){\makebox(0,0)[b]{$v_{3}(r)$}}
      \put(430,360){\makebox(0,0)[l]{$\frac{b}{a} = \sigma$}}
      \put(530,75){\makebox(0,0)[l]{$r \in {\cal R}(\alpha, \sigma)$}}
    \end{picture}
  \end{image}
\end{center}
```

creates the following image.



It remains to describe how you actually generate those bitmaps from your Hyperlatex source. This is done by running LaTeX on the input file, setting a special flag that makes the resulting Dvi-file contain an extra page for every `image` environment. Furthermore, this LaTeX-run produces another file with extension *.makeimage*, which contains commands to run `dvips` and `ps2image` to extract the interesting pages into Postscript files which are then converted to `image` format. Obviously you need to have `dvips` and `ps2image` installed if you want to use this feature. (A shellscript `ps2image` is supplied with Hyperlatex. This shellscript uses `ghostscript` to convert the Postscript files to `ppm` format, and then runs `pnmtopng` to convert these into `png`-files.)

Assuming that everything has been installed properly, using this is actually quite easy: To generate the `png` bitmaps defined in your Hyperlatex source file *source.tex*, you simply use

```
hyperlatex -image source.tex
```

Note that since this runs latex on *source.tex*, the Dvi-file *source.dvi* will no longer be what you want!

For compatibility with older versions of Hyperlatex, the `gif` environment is equivalent to the `image` environment. To produce `gif` images instead of `png` images, the command `\imagetype{gif}` can be put in the preamble of the document.

10 Controlling Hyperlatex

Practically everything about Hyperlatex can be modified and adapted to your taste. In many cases, it suffices to redefine some of the macros defined in the *siteinit.hlx* package.

10.1 Siteinit, Init, and other packages

When Hyperlatex processes the `\documentclass{class}` command, it tries to read the Hyperlatex package files *siteinit.hlx*, *init.hlx*, and *class.hlx* in this order. These package files implement most of Hyperlatex's functionality using LaTeX-style macros. Hyperlatex looks for these files in the directory *.hyperlatex* in the user's home directory, and in the system-wide Hyperlatex directory selected by the system administrator (or whoever installed Hyperlatex). *siteinit.hlx* contains the standard definitions for the system-wide installation of Hyperlatex, the package *class.hlx* (where *class* is one of *article*, *report*, *book* etc) define the commands that are different between different LaTeX classes.

System administrators can modify the default behavior of Hyperlatex by modifying *siteinit.hlx*. Users can modify their personal version of Hyperlatex by creating a file *~/.hyperlatex/init.hlx* with definitions that override the ones in *siteinit.hlx*. Finally, all these definitions can be overridden by redefining macros in the preamble of a document to be converted.

To change the default depth at which a document is split into nodes, the system administrator could change the setting of `htmldepth` in *siteinit.hlx*. A user could define this command in her personal *init.hlx* file. Finally, we can simply use this command directly in the preamble.

10.2 Splitting into nodes and menus

Normally, the HTML output for your document *document.tex* are created in files *document_?.html* in the same directory. You can change both the name of these files as well as the directory using the two commands `\htmlname` and `\htmldirectory` in the preamble of your source file:

```
\htmldirectory{directory}
\htmlname{basename}
```

The actual files created by Hyperlatex are called

directory/basename.html, *directory/basename_1.html*, *directory/basename_2.html*,

and so on. The filename can be changed for individual nodes using the `\xname` command.

Hyperlatex automatically partitions the document into several nodes. This is done based on the LaTeX sectioning. The section commands `\chapter`, `\section`, `\subsection`, `\subsubsection`, `\paragraph`, and `\subparagraph` are assigned levels 0 to 5.

The counter `htmldepth` determines at what depth separate nodes are created. The default setting is 4, which means that sections, subsections, and subsubsections are given their own nodes, while paragraphs and subparagraphs are put into the node of their parent subsection. You can change this by putting

```
\setcounter{htmldepth}{depth}
```

in the preamble. A value of 0 means that the full document will be stored in a single file.

The individual nodes of an HTML document are linked together using *hyperlinks*. Hyperlatex automatically places buttons on every node that link it to the previous and next node of the same depth, if they exist, and a button to go to the parent node.

Furthermore, Hyperlatex automatically adds a menu to every node, containing pointers to all subsections of this section. (Here, "section" is used as the generic term for chapters, sections, subsections, ...) This may not always be what you want. You might want to add nicer menus, with a short description of the subsections. In that case you can turn off the automatic menus by putting

```
\setcounter{htmlautomenu}{0}
```

in the preamble. On the other hand, you might also want to have more detailed menus, containing not only pointers to the direct subsections, but also to all subsubsections and so on. This can be achieved by using

```
\setcounter{htmlautomenu}{depth}
```

where *depth* is the desired depth of recursion. The default behavior corresponds to a *depth* of 1.

10.3 Customizing the navigation panels

Normally, Hyperlatex adds a "navigation panel" at the beginning of every HTML node. This panel has links to the next and previous node on the same level, as well as to the parent node.

The easiest way to customize the navigation panel is to turn it off for selected nodes. This is done using the commands `\htmlpanel{0}` and `\htmlpanel{1}`. All nodes started while `\htmlpanel` is set to 0 are created without a navigation panel.

If you wish to add additional fields (such as an index or table of contents entry) to the navigation panel, you can use `\htmlpanelfield` in the preamble. It takes two arguments, the text to show in the field, and a label in the document where clicking the link should take you. For instance, the navigation panels for this manual were created by adding the following two lines in the preamble:

```
\htmlpanelfield{Contents}{hlxcontents}
\htmlpanelfield{Index}{hlxindex}
```

Furthermore, the navigation panels (and in fact the complete outline of the created HTML files) can be customized to your own taste by redefining some Hyperlatex macros. When it formats an HTML node, Hyperlatex inserts the macro `\toppanel` at the beginning, and the two macros `\bottommatter` and `bottompanel` at the end. When `\htmlpanel{0}` has been set, then only `\bottommatter` is inserted.

The macros `\toppanel` and `\bottompanel` are responsible for typesetting the navigation panels at the top and the bottom of every node. You can change the appearance of these panels by redefining those macros. See *bluepanels.hlx* for their default definition.

You can use `\htmltopname` to change the name of the top node.

If you have included language packages from the babel package, you can change the language of the navigation panel using, for instance, `\htmlpanelgerman`.

The following commands are useful for defining these macros:

- `\HlxPrevUrl`, `\HlxUpUrl`, and `\HlxNextUrl` return the URL of the next node in the backwards, upwards, and forwards direction. (If there is no node in that direction, the macro evaluates to the empty string.)
- `\HlxPrevTitle`, `\HlxUpTitle`, and `\HlxNextTitle` return the title of these nodes.
- `\HlxBackUrl` and `\HlxForwUrl` return the URL of the previous and following node (without looking at their depth)
- `\HlxBackTitle` and `\HlxForwTitle` return the title of these nodes.
- `\HlxThisTitle` and `\HlxThisUrl` return title and URL of the current node.
- The command `\EmptyP{expr}{A}{B}` evaluates to `A` if `expr` is not the empty string, to `B` otherwise.

10.4 Changing the formatting of footnotes

The appearance of footnotes in the HTML output can be customized by redefining several macros:

The macro `\htmlfootnotemark{`*n*`}` typesets the mark that is placed in the text as a hyperlink to the footnote text. See the file *siteinit.hlx* for the default definition.

The environment `thefootnotes` generates the HTML node with the footnote text. Every footnote is formatted with the macro `\htmlfootnoteitem{`*n*`}{`*text*`}`. The default definitions are

```
\newenvironment{thefootnotes}%
   {\chapter{Footnotes}
    \begin{description}}%
   {\end{description}}
\newcommand{\htmlfootnoteitem}[2]%
   {\label{footnote-#1}\item[(#1)]#2}
```

10.5 Setting Html attributes

If you are familiar with HTML, then you will sometimes want to be able to add certain HTML attributes to the HTML tags generated by Hyperlatex. This is possible using the command `\xmlattributes`. Its first argument is the name of an HTML tag (in lower case!), the second argument can be used to specify attributes for that tag. The declaration can be used in the preamble as well as in the document. A new declaration for the same tag cancels any previous declaration, unless you use the starred version of the command: It has effect only on the next occurrence of the named tag, after which Hyperlatex reverts to the previous state.

All the HTML-tags created using the `\xml`-command can be influenced by this declaration. There are, however, also some HTML-tags that are created directly in the Hyperlatex kernel and that do not look up any attributes here. You can only try and see (and complain to me if you need to set attribute for a certain tag where Hyperlatex doesn't allow it).

Some common applications:

HTML 3.2 allows you to specify the background color of an HTML node using an attribute that you can set as follows. (If you do this in *init.hlx* or the preamble of your file, all nodes of your document will be colored this way.) Note that this usage is deprecated, you should be using a style sheet instead.

```
\xmlattributes{body}{bgcolor="#ffffe6"}
```

The following declaration makes the tables in your document have borders.

```
\xmlattributes{table}{border}
```

A more compact representation of the list environments can be enforced using (this is for the `itemize` environment):

```
\xmlattributes{ul}{compact}
```

The following attributes make section and subsection headings be centered.

```
\xmlattributes{h1}{align="center"}
\xmlattributes{h2}{align="center"}
```

10.6 Making characters non-special

Sometimes it is useful to turn off the special meaning of some of the ten special characters of LaTeX. For instance, when writing documentation about programs in C, it might be useful to be able to write `some_variable` instead of always having to type `some\_variable`, especially if you never use any formula and hence do not need the subscript function. This can be achieved with the `\NotSpecial` command. The characters that you can make non-special are

```
~  ^  _  #  $  &
```

For instance, to make characters `$` and `^` non-special, you need to use the command

```
\NotSpecial{\do\$\do\^}
```

Yes, this syntax is weird, but it makes the implementation much easier.

Note that whereever you put this declaration in the preamble, it will only be turned on by `\begin{document}`. This means that you can still use the regular LaTeX special characters in the preamble.

Even within the `iftex` environment the characters you specified will remain non-special. Sometimes you will want to return them their full power. This can be done in a `tex` environment. It is equivalent to `iftex`, but also turns on all ten special LaTeX characters.

10.7 CSS, Character Sets, and so on

An HTML-file can carry a number of tags in the HTML-header, which is created automatically by Hyperlatex. There are two commands to create such header tags:

`\htmlcss` creates a link to a cascaded style sheet. The single argument is the URL of the style sheet. The tag will be added to every node *created after* the command has been processed. Use an empty argument to turn of the CSS link.

`\htmlcharset` tags the HTML-file as being encoded in a particular character set. Use an empty argument to turn off creation of the tag.

Here is an example:

```
\htmlcss{http://www.w3.org/StyleSheets/Core/Modernist}
\htmlcharset{EUC-KR}
```

11 Extending Hyperlatex

As mentioned above, the `documentclass` command looks for files that implement LaTeX classes in the directory *~/.hyperlatex* and the system-wide Hyperlatex directory. The same is true for the `\usepackage{package}` commands in your document.

Some support has been implemented for a few of these LaTeX packages, and their number is growing. We first list the currently available packages, and then explain how you can use this mechanism to provide support for packages that are not yet supported by Hyperlatex.

11.1 The *frames* package

If you `\usepackage{frames}`, your document will use frames, like this manual. The navigation panel shown on the left hand side is implemented by `\HlxFramesNavigation`, modify it if you prefer a different layout.

11.2 The *sequential* package

Some people prefer to have the *Next* and *Prev* buttons in the navigation panels point to the sequentially adjacent nodes. In other words, when you press *Next* repeatedly, you browse through the document in linear order.

The package *sequential* provides this behavior. To use it, simply put

```
\W\usepackage{sequential}
```

in the preamble of the document (or in your *init.hlx* file, if you want this behavior for all your documents).

11.3 Xspace

Support for the `xspace` package is already built into Hyperlatex. The macro `\xspace` works as it does in LaTeX.

11.4 Longtable

The `longtable` environment allows for tables that are split over multiple pages. In HTML, obviously splitting is unnecessary, so Hyperlatex treats a `longtable` environment identical to a `tabular` environment. You can use `\label` and `\link` inside a `longtable` environment to create cross references between entries.

11.5 Tabularx

The X column type is implemented.

11.6 Using color in Hyperlatex

From the `color` package: `\color`, `\textcolor`, `\definecolor`.

From the `pstcol` package: `\newgray`, `\newrgbcolor`, `\newcmykcolor`.

From the `colortbl` package: `\columncolor`, `\rowcolor`.

11.7 Babel

Thanks to Eric Delaunay, the babel package is supported with English, French, German, Dutch, Italian, and Portuguese modes. If you need support for a different language, try to implement it yourself by looking at the files *english.hlx*, *german.hlx*, etc.

For instance, the german mode implements all the ¨-commands of the babel package. In addition, it defines the macros for making quotation marks. So you can easily write something like this:

> Der König saß da und überlegte sich, wieviele Öchslegrade wohl der weiße Wein haben würde, als er plötzlich ≪Majesté≫ rufen hörte.

by writing:

```
Der K"onig sa"z da  und "uberlegte sich, wieviele
"Ochslegrade wohl der wei"ze Wein haben w"urde, als er pl"otzlich
"<Majest\'e"> rufen h"orte.
```

You can also switch to German date format, or use German navigation panel captions using `\htmlpanelgerman`.

11.8 Documenting code

The `cppdoc` package can be used to document code in C++ or Java. This is experimental, and may either be extended or removed in future Hyperlatex distributions. There are far more powerful code documentation tools available—I'm playing with the `cppdoc` package because I find a simple tool that I understand well more helpful than a complex one that I forget to use and therefore don't use.

The package defines a command `cppinclude` to include a C++ or Java header file. The header file is stripped down before it is interpreted by Hyperlatex, using certain comments to control the inclusion:

- A comment starting with `/**` and up to `*/` is included.
- Any line starting with `//+` is included.
- A comment of the form `//--` is converted to `\begin{cppenv}`, and the following code is not stripped. This environment is ended using `//--`. All known class names inside this environment will be converted to links.
- A comment of the form `///` can be used at the end of the first line of a method. The method name will be extracted as the argument to `\cppmethod`,. The method declaration needs to be followed by a `/**` or `//+` comment documenting the method.

Note that the `cppenv` environment and the `\cppmethod` command are not provided by `cppdoc`. You have to define them in your document. A simple definition would be:

```
\newenvironment{cppenv}{\begin{example}}{\end{example}}
\newcommand{\cppmethod}[1]{\paragraph{#1}}
```

You can use `\cpplabel` to put a label in the section documenting a certain class. `\cpplabel{Engine}` will place an ordinary label `class:Engine` in the document, and will also remember that `Engine` is the name of a class known in the project (and will therefore be converted to a link inside a `cppenv` environment and the argument to `\cppmethod`).

The command `\cppclass` takes a single class name as an argument, and creates a link if a label for that class has been defined in the document.

If you use `\cppextras`, then the vertical bar character is made active. You can use a pair of vertical bars as a shortcut for the `\cppclass` command.

11.9 Writing your own extensions

Whenever Hyperlatex processes a `\documentclass` or `\usepackage` command, it first saves the options, then tries to find the file *package.hlx* in either the *.hyperlatex* or the systemwide Hyperlatex directories. If such a file is found, it is inserted into the document at the current location and processed as usual. This provides an easy way to add support for many LaTeX packages by simply adding LaTeX commands. You can test the options with the `ifoption` environment (see *babel.hlx* for an example).

To see how it works, have a look at the package files in the distribution.

If you want to do something more ambitious, you may need to do some Emacs lisp programming. An example is *german.hlx*, that makes the double quote character active using a piece of Emacs lisp code. The lisp code is embedded in the *german.hlx* file using the `\HlxEval` command.

Note that Hyperlatex now provides rudimentary support for counters. The commands `\setcounter`, `\newcounter`, `\addtocounter`, `\stepcounter`, and `\refstepcounter` are implemented, as well as the `\the`*countername* command that returns the current value of the counter. The counters are used for numbering sections, you could use them to number theorems or other environments as well.

If you write a support file for one of the standard LaTeX packages, please share it with us.

11.10 Macro names

You may wonder what the rationale behind the different macro names in Hyperlatex is. Here's the answer:

- A few macros like `\link`, `\xlink` and environments like `menu`, `rawxml`, `example`, `ifhtml`, `iftex`, `ifset` provide additional functionality to the markup language. They are understood by Hyperlatex and LaTeX (assuming `\usepackage{hyperlatex}`, of course).
- `\xml` and `\html...` macros allow the user to influence the generation of XML (HTML) output. They are meant to be used in Hyperlatex documents, but have no effect on the LaTeX output. They are understood by Hyperlatex and LaTeX (but are dummies in LaTeX).
- `\Hlx...` macros are understood by Hyperlatex, but not by LaTeX (they are not defined in *hyperlatex.sty*). They are meant for defining macros and environments in Hyperlatex without resorting to Lisp, making Hyperlatex styles easier to customize and maintain. They are used in *siteinit.hlx*, *init.hlx*, etc., and not normally used in Hyperlatex documents (you can use them inside of `ifhtml` environments or other escapes that stop LaTeX from complaining about them)

12 Changes in Hyperlatex

Changes from 2..6 to 2.7 Hyperlatex has been moved to sourceforge.net. Image support was changed to remove reliance on GIF images

Changes from 2.5 to 2.6 Hyperlatex has moved to producing XHTML 1.0. The migration is not complete, and Hyperlatex's output will not (yet) pass an XHTML checker. This version is released only since I've been using it so long and it was stable (for me).

- DTD declaration now refers to XHTML.
- Labels that you want to be visible externally must respect XML rules for the id attribute.
- Removed optional argument of `\htmlrule`. Roll your own if you need it.
- `\htmlimage` is deprecated, and replaced by `\htmlimg{url}{alt}`, since the alternate text is now mandatory in HTML.
- Using small style sheet to implement and distinguish `verse`, `quotation`, and `quote` environments.
- Replaced deprecated `<menu>` tag by `<ul>`.
- Creating `<tbody>` tags for tables.
- `\htmlsym` renamed to `\xmlent` (but old version still supported).
- Experimental package `hyperxml` for creating XML files.
- Handle DOS files (with CRLF) cleanly.

Changes from 2.4 to 2.5

- Index was missing from LaTeX docs.
- Fixed bug in German/French/Portuguese month names in `\today`.
- New `cppdoc` package to document code.
- `example` environment is no longer automatically indented.
- Started some work on generating correct XHTML 1.0. A few commands starting with `\html` have been renamed to start with `\xml` (you can find them all in the index), but for the important ones, the old version still works and will continue to work indefinitely. The `ifhtmllevel` environment has been removed. The XML tags generated by Hyperlatex are now in lower case.
- Changed BibTeX trick to use `@preamble` and `\providecommand`.
- `\htmlimage` works inside the argument of `\section`. The contents of the `<title>` tag is now properly cleansed.

Changes from 2.3 to 2.4

- Included current directory in search for *.hlx* files.
- Can use `\begin{verbatim}` inside `\newenvironment`.
- More attractive blue navigation panel (you can use a simpler style using `\usepackage{simplepanels}`). It is now easy to add index or contents fields to the panels using `\htmlpanelfield`.
- Fixed Y2K bug.
- Added Portuguese and Italian to Babel.
- `emulate` and `multirow` packages degraded to "contrib" status. They probably need a volunteer to be maintained/fixed.
- `\providecommand` added.
- `\input{\name}` should work now.
- Will print number of issues warnings at the end.
- `\cite` understands the optional argument and accepts whitespace after the comma.
- Support for CSS and character set tagging.

- `\htmlmenu` takes an optional argument to indicate the section for which we want the menu (makes FAQ 2.1 obsolete).
- Obsolete and useless Javascript stuff replaced by simpler frames that do not use Javascript.

Changes from 2.2 to 2.3

- Added possibility of making `<META>` tags.
- Compatibility with GNU Emacs 20.
- Lots and lots of improvements by Eric Delaunay, including support for color packages, support for more column types and `\newcolumntype` for tabular environments, and a real Babel system that can handle multiple languages, even in the same document.
- Allow *.htm* file extension for brain-damaged file systems.
- Bugfixes, and new commands `\HlxThisUrl`, `\HlxThisTitle`, `\htmltopname` by Sebastian Erdmann.
- Makeidx package by Sebastian Erdmann.
- Improved GIF generation by Rolf Niepraschk (based on "Goossens/Rahtz/Mittelbach: The LaTeX Graphics Companion" pp. 455).
- (2.3.1) Fixed bug in tabular.
- (2.3.1) Moved tabbing environment into main Hyperlatex code.
- (2.3.1) Array environment.
- (2.3.2) Fixed `\.` bug—it wasn't processed as a macro.

Changes from 2.1 to 2.2

- Extended counters considerably, implementing counters within other counters. Some special `\html...` commands where replaced by counters, such as `\htmlautomenu`, `\htmldepth`.
- `\htmlref`{label} returns the counter that was stepped before the label was defined.
- Sections can now be numbered automatically by setting the counter `secnumdepth`.
- Removed searching for packages in Emacs lisp, instead provided `\HlxEval` command.
- Added a package for making a frame based document with Javascript. Needed to put some support in the Hyperlatex kernel.
- Extended the `Emulate` package with dummy declarations of many LaTeX commands.
- `\cite{key1,key2,key3}` works now.
- Counter arguments in `\newtheorem` now work.
- Made additional icon bitmaps *greynext.xbm*, *greyprevious.xbm*, and *greyup.xbm*. These are greyed out versions of the normal icons and used when the links are not active (when there is no next or previous node). They have to be installed on the server at the same place as the old icons.

Changes from 2.0 to 2.1

- Bug fixes.
- Added rudimentary support for counters.
- Added support for creating packages that define active characters. Created a basic implementation for `\usepackage[german]{babel}`.

Changes from 1.4 to 2.0 Hyperlatex 2.0 is a major new revision. Hyperlatex now consists of a kernel written in Emacs lisp that mainly acts as a macro interpreter and that implements some low-level functionality. Most of the Hyperlatex commands are now defined in the system-wide initialization file *siteinit.hlx*. This will make it much easier to customize, update, and improve Hyperlatex.

- Made Hyperlatex kernel deal only with macro processing and fundamental tasks. High-level functionality has been moved to the Hyperlatex macro level in *siteinit.hlx*.
- The preamble is now parsed properly, and the treatment of the classes and packages with `\documentclass` and `\usepackage` has been revised to allow for easier customization by loading macro packages.
- Added Peter D. Mosses's `tabbing` package to distribution.
- Changed `ps2gif` to use `netpbm`'s version of `ppmtogif`, which makes `giftrans` unnecessary.
- Added explanation of some features to the manual.
- The `\index` command now understands the *sortkey@entry* syntax of `makeindex`.
- Fixed the problem that forced one to put a space at the end of commands.
- The `\xlabel` command has been removed. `\label` has been extended to include its functionality.
- And many others...

Changes from 1.3 to 1.4 Hyperlatex 1.4 introduces some incompatible changes, in particular the ten special characters. There is support for a number of HTML 3 features.

- All ten special LaTeX characters are now also special in Hyperlatex. However, the `\NotSpecial` command can be used to make characters non-special.
- Some non-standard-LaTeX commands (such as `\H`, `\+`, `\*`, `\S`, `\C`, `\minus`) are no longer recognized by Hyperlatex to be more like standard Latex.
- The `\htmlmathitalics` command has disappeared (it's now the default, unless we use `<math>` tags.)
- Within the `example` environment, only the four characters `%`, `\`, `{`, and `}` are special now.
- Added the starred versions of `\link*` and `\xlink*`.
- Added `\texorhtml`.
- The `\set` and `\clear` commands have been removed, and their function has been taken over by `\newcommand`.
- Added `\htmlheading`, and the possibility of leaving section headings empty in HTML.
- Added math mode support.
- Added tables using the `<table>` tag.
- ...and many other things.

Changes from 1.2 to 1.3 Hyperlatex 1.3 fixes a few bugs.

Changes from 1.1 to 1.2 Hyperlatex 1.2 has a few new options that allow you to better use the extended HTML tags of the `netscape` browser.

- `\htmlrule` now has an optional argument.
- The optional argument for the `\htmlimage` command and the `gif` environment has been extended.
- The `center` environment now uses the *center* HTML tag understood by some browsers.
- The font changing commands have been changed to adhere to LaTeX 2_ε. The font size can be changed now as well, using the usual LaTeX commands.

Changes from 1.0 to 1.1

- The only change that introduces a real incompatibility concerns the percent sign `%`. It has its usual LaTeX-meaning of introducing a comment in Hyperlatex 1.1, but was not special in Hyperlatex 1.0.
- Fixed a bug that made Hyperlatex swallow certain ISO characters embedded in the text.
- Fixed HTML tags generated for labels such that they can be parsed by `lynx`.

- The commands `\+verb+` and `\=` are now shortcuts for `\verb+`*verb*`+` and `\back`.
- It is now possible to place labels that can be accessed from the outside of the document using `\xname` and `\xlabel`.
- The navigation panels can now be suppressed using `\htmlpanel`.
- If you are using LaTeX 2_ε, the Hyperlatex input mode is now turned on at `\begin{document}`. For LaTeX2.09 it is still turned on by `\topnode`.
- The environment `gif` can now be used to turn Dvi information into a bitmap that is included in the Html-document.

13 Acknowledgments

Thanks to everybody who reported bugs or who suggested (or even implemented!) useful new features. This includes Eric Delaunay, Jay Belanger, Sebastian Erdmann, Rolf Niepraschk, Roland Jesse, Arne Helme, Bob Kanefsky, Greg Franks, Jim Donnelly, Jon Brinkmann, Nick Galbreath, Piet van Oostrum, Robert M. Gray, Peter D. Mosses, Chris George, Barbara Beeton, Ajay Shah, Erick Branderhorst, Wolfgang Schreiner, Stephen Gildea, Gunnar Borthne, Christophe Prudhomme, Stefan Sitter, Louis Taber, Jason Harrison, Alain Aubord, Tom Sgouros, René van Oostrum, Robert Withrow, Pedro Quaresma de Almeida, Bernd Raichle, Adelchi Azzalini, Alexander Wolff.

14 Copyright

References

[1] Leslie Lamport, *LaTeX: A Document Preparation System,* Second Edition, Addison-Wesley, 1994.

[2] at `http://www.gnu.org/copyleft/gpl.html`

Index

Contents