LaTeX2ε, an overview

Michel Goossens

CN Division, CERN
CH1211 Genève 23
Switzerland
goossens@cern.ch

Abstract

This article gives an overview of the new or extended user commands available with LaTeX2ε, the new LaTeX release, compared to the previous version LaTeX 2.09. After introducing the new preamble commands, the extensions for defining new commands and environments, and handling length and boxes are discussed. The new font selection commands are explained, both for text and math, and it is shown how to easily use different font families. A list of supported class and package files is given and new possibilities for controlling page contents and floats are discussed. Most of this material is described in much greater detail in 'The LaTeX Companion' [1] and in the second edition of the LaTeX Reference Manual [2].

1 Why LaTeX2ε?

Since LaTeX became generally available in 1986, its popularity has increased ever since, and many extensions have been developed. Unluckily, these extensions were included in incompatible formats, e.g., 'standard' LaTeX with and without NFSS, SLITeX, AMS-LaTeX, and so on. From the LaTeX source alone, it was difficult to determine for which of these (or other) formats a document was composed and, because different sites could have different configuration, document portability was a problem.

Already in 1989 at the Stanford TUG Conference Frank Mittelbach and Rainer Schöpf got together with Leslie Lamport to discuss these (and other) topics and they published their ideas about possible ways to evolve LaTeX in TUGBoat [3, 4]. This lead a few years later to the start of the long-term LaTeX3 project [5–8].

However, to help end the confusion for the present LaTeX users, after a meeting in Spring 1993 between Leslie Lamport and Frank Mittelbach in Mainz, it was decided to release an upgraded version of LaTeX, called LaTeX2ε, which was officially announced in August 1993 at the TUG Conference at Aston.

Its stated aims are:

- bring all extensions back under a single format;
- prevent proliferation of mutually incompatible dialects of LaTeX 2.09;
- NFSS becomes the 'standard' font selection scheme;
- make style files like amstex (formerly AMS-LaTeX format) or slides (formerly SLITeX format) into extension packages, all using the same base format;
- add a small number of often-requested features;
- retain the 'touch and feel', or the 'flavour' of LaTeX 2.09.

The first beta version of LaTeX2ε was released at the end of 1993, while the first production release is foreseen for 'Spring 1994'. After that, twice a year (in 'Spring' and 'Autumn') consolidation releases are planned, in order to keep all versions of the files in synchronization. Bug reports are handled centrally by inviting the users to fill out an electronic form, distributed with the LaTeX2ε distribution, and sending it via electronic mail to `latex-bugs@rus.uni-stuttgart.de`. Note that only bug reports referring to the last two releases will be considered. You can also subscribe to the LaTeX2ε discussion list on `LATEX-2E@DHDURZ1.BITNET` and post questions (and answers) to that list.

2 Initial and preamble commands

In this section commands which can only be used before or in the preamble will be discussed. The first two below can only appear *before* the `\documentclass` command.

2.1 Initial commands

```
\NeedsTeXFormat{format-name}[release-date]
```

This command, which is normally present in package and class files, can also be useful in user documents to make sure that the file is run with LaTeX2ε. Users who try and run it with LaTeX 2.09 or plain TeX will get a reasonably clear error message. An example is

```
\NeedsTeXFormat{LaTeX2e}[1994/02/01]
```

If you want to make sure that your document can be processed at another site, it could make sense that you include all packages and files that your document needs together with the main file. LaTeX2ε provides the following syntax to facilitate this

```
\begin{filecontents}{file-name}
    ⟨file-contents⟩
\end{filecontents}
```

When your document file is run through LaTeX2ε the body of each `filecontents` environment is written verbatim to a file whose name is given as the argument *file-name*. If a file with such a name already exists in any of the directories 'visible' to TEX only an informative message is given, the body of the environment is by-passed, and the file is not replaced.

2.2 Preamble commands

The next commands in the preamble are specifically designed to differentiate LATEX2$_\varepsilon$ documents from those needing LATEX 2.09.

`\documentclass` [*option-list*] {*class-name*} [*release-date*]

This command or 'declaration' replaces the LATEX 2.09 command `\documentstyle`.

There must be exactly one `\documentclass` declaration in a document, and it must come first (except for the 'initial' commands described above).

option-list: list of options that each can modify the formatting of document elements defined in the *class-name* file or in packages loaded with `\usepackage` declarations, as described below.

class-name: name of the class file (file extension `.cls`).

release-date (optional) specifies release date of the class file, using the format `YYYY/MM/DD`. If a version of the class older than this date is found, a warning is issued.

`\documentstyle` [*option-list*] {*style-name*} [*release-date*]

This command, which is supported for compatibility reasons, is similar to `\documentclass`, but it loads a 'compatibility mode' which redefines certain commands to act as they did in LATEX 2.09 and thus allows you to run your old files unchanged through LATEX2$_\varepsilon$. Note, however, that in this mode, you *cannot* use any of the LATEX2$_\varepsilon$ extensions described in this article.

You can define new or change existing document elements by loading package files with `\usepackage`, whose syntax is:

`\usepackage` [*option-list*] {*package-name*} [*release-date*]

package-name: name of the package (file extension `.sty`); a package can

- define new document elements;
- modify elements defined in the class file;
- extend the range of documents that can be processed.

option-list: list of options, each of which can modify the formatting of elements defined in the package.

release-date: (optional) earliest desired release date of package file (see `\documentclass` command above).

Any number of `\usepackage` are allowed, but LATEX2$_\varepsilon$ makes sure that each package is only loaded once. On top of processing the list of options specified in the argument *option-list* on its `\usepackage` command, each package also processes the option *option-list* on the `\documentclass` command.

`\listfiles`

To help you get an overview of the files read in by your document during processing, you can place a `\listfiles` command in the preamble of your document. This will display the list of all files used at the end of the run.

2.3 Example of document preambles

The following preamble references the article class with the (global) options twocolumn and a4paper, and loads the multicol and babel packages, the latter with the german and french options. Other document parameters (e.g., the `textheight`) can also be specified.

```
\documentclass[twocolumn,a4paper]{article}
\usepackage{multicol}
\usepackage[german,french]{babel}
\addtolength{\textheight}{2cm}
\begin{document}
    ...
\end{document}
```

The following shows three equivalent ways of specifying the loading of packages.

```
\documentclass[german]{book}
\usepackage[german]{babel}
\usepackage[german]{varioref}
\usepackage{multicol}
\usepackage{epic}
```

Somewhat less verbose is:

```
\documentclass[german]{book}
\usepackage[german]{babel,varioref}
\usepackage{multicol,epic}
```

With german as global option you can write:

```
\documentclass[a4paper,german]{book}
\usepackage{babel,varioref,multicol,epic}
```

A complex document might look something like the following:

```
\NeedsTeXFormat{LaTeX2e}[1994/05/01]
\begin{filecontents}{varioref.sty}
   ....            % Code for varioref package
\end{filecontents}
\listfiles         % print list of files referenced
\documentclass[a4paper,german]{book} % book class
\usepackage{varioref}
\begin{document}
%-------------------- front matter of document
\maketitle
\section*{...}     % e.g. section named "Preface"
\tableofcontents   % chapter with table of contents
\listoffigures     % chapter with list of figures
\listoftables      % chapter with list of tables
%-------------------- body of the document
```

```
\part{...}
\chapter{...}
   \section{...}
\chapter{...}
\part{...}
%-------------------- back matter of document
\appendix
\chapter{...}     % chapters labelled appendix
\chapter{...}
\begin{thebibliography}
  ...             % bibliography entries
\end{thebibliography}
\begin{theindex}
  ...             % index entries
\end{theindex}
\end{document}
```

Note that, to ensure that the recipient of the document can process the file correctly, the code of the varioref package is shipped with the file inside a `filecontents` environment.

3 Option processing

Options that are specified in the *option-list* argument of the `\documentclass` or `\usepackage` commands are handled as follows:

1. They are first divided into two types, *local* and *global*:
   - for a *class*, the options from its `\documentclass` command are local and there are no global options;
   - for a *package*, the options from its `\usepackage` command are local but the options from the `\documentclass` command are global.
2. The local and global options that have been declared within the current class or package are processed first, normally in their order of declarations, thus their order in *option-list* is irrelevant.
3. Any local options not declared in the current class or package are then processed. For document classes, this usually means that they are ignored, except for this fact being recorded by adding the option to a list of 'unused options'; they may, of course, be used later since they become global options for every package subsequently loaded. For packages, usually an error message is produced, giving the choice of retyping the option name in case it is incorrect.

Finally, when the `\begin{document}` command is reached LaTeX2ε will produce a list of all global options not used by the class or any package file, and issue a warning message for each.

4 Defining new commands and environments

This section and the following describe commands and environments that are used inside the document body (i.e., after the `\begin{document}`) command. Let us first look at what is available for defining new commands and environments.

4.1 Defining commands

Commands are defined or redefined in LaTeX with:

```
\newcommand{\mycom}[narg][default]%
        {command definition}
\renewcommand{\mycom}[narg][default]%
        {command definition}
\providecommand{\mycom}[narg][default]%
        {command definition}
```

The first and second commands show enhancements with respect to LaTeX 2.09 by providing the possibility to have an *optional* argument when (re)defining a user command. The presence of such an optional parameter is flagged by the presence of the second optional argument *default*, which specifies the default value of that argument when it is not specified explicitly when the command is used. The last form is useful for general purpose files that are included in a document and over which the user does not always have control (e.g., BibTeX databases). If `\mycom` is not yet defined, the `\providecommand` will act as `\newcommand` and define it, otherwise the existing definition is kept.

The number of arguments, which *includes* the optional argument, is in the range 0≤*narg*≤9. If the command has no arguments, then the `[0]` can be omitted. Inside the *command definition* part, the arguments are referenced as `#1` to #*narg*, the optional argument, if present, being the first one.

For example, compare the following commands, with no, one mandatory, one optional, and one optional and one mandatory argument, allowing the user more freedom in each case.

```
\newcommand{\seq}{x_{0},\ldots\,x_{n}}
\newcommand{\seqm}[1]{#1_{0},\ldots\,#1_{n}}
\newcommand{\seqo}[1][k]{x_{0},\ldots\,x_{#1}}
\newcommand{\seqom}[2][k]{#2_{0},\ldots\,#2_{#1}}
$$\seq\quad\seqm{z}$$
$$\seqo\quad\seqo[l]$$
$$\seqom{y}\quad\seqom[i]{q}$$
```

This gives:

$$x_{0},\ldots\,x_{n}\quad z_{0},\ldots\,z_{n}$$

$$x_{0},\ldots\,x_{k}\quad x_{0},\ldots\,x_{l}$$

$$y_{0},\ldots\,y_{k}\quad q_{0},\ldots\,q_{i}$$

If a command should work both in math and in text mode, special care should be taken in its definition. In LaTeX2ε you have the following command:

`\ensuremath{`*math code*`}`

As its name implies `\ensuremath` ensures that its argument is always typeset in math mode by surrounding it if necessary with `$` signs. For instance, the above can be rewritten as:

```
\renewcommand{\seq}{\ensuremath{x_{0},%
                                   \ldots\,x_{n}}}
\renewcommand{\seqm}[1]{%
                   \ensuremath{#1_{0},%
                                  \ldots\,#1_{n}}}
\seq,\quad\seqm{z} or $\seq,\quad\seqm{z}$
```

$x_0, \ldots x_n, \quad z_0, \ldots z_n$ or $x_0, \ldots x_n, \quad z_0, \ldots z_n$

4.2 Defining New Environments

In LaTeX 2.09, environments are defined or redefined with the commands:

`\newenvironment{`*name*`}[`*narg*`]{`*begdef*`}{`*enddef*`}`
`\renewenvironment{`*name*`}[`*narg*`]{`*begdef*`}{`*enddef*`}`

The number of arguments is in the range `0`≤*narg*≤`9`; and, in the case of no parameters, you can omit `[0]`. Inside the definition part, *begdef*, these parameters are referenced as `#1` to `#`*narg*. If arguments are present, then they are defined when *entering* the environment by specifying them on the command `\begin{myenv}` as shown below.

`\begin{myenv}{`arg_1`}...{`arg_k`}`

When *exiting* an environment with the command `\end{myenv}` no parameters can be specified. Moreover, the parameters specified with the `\begin{myenv}` command when entering the environment (see above) are no longer available in the definition part *enddef* where you define the actions which should take place when leaving the *myenv* environment.

As with commands, in LaTeX2$_\varepsilon$ you can now also define environments with an optional (first) argument.

`\newenvironment{`*myenv*`}[`*narg*`][`*default*`]%`
`                  {`*begdef*`}{`*enddef*`}`

The default for the optional argument is given between the second pair of square brackets `[`*default*`]`. Inside the *begdef* part, which is executed when the environment *myenv* is entered, the optional argument can be accessed with `#1`, while the mandatory arguments (when present) are addressed as `#2` to `#`*narg*. When the *myenv* environment is used without an optional parameter, `#1` will contain the string specified as `[`*default*`]`.

As an example, a variant, `deflist`, of a `description` environment will be constructed. The `deflist` environment behaves somewhat like a standard LaTeX `description` environment if it is used without an optional argument. If an optional argument is specified, then the width of the description label will be put equal to the width of the argument. Thus, by specifying the widest entry in the list as an optional argument, you ensure that the description parts of all entries line up nicely.

The result below first shows the (default) behaviour of the `deflist` list and then what it looks like when using the optional argument.

```
\newenvironment{deflist}[1][\quad]%
 {\begin{list}{}{
  \renewcommand{\makelabel}[1]{\textbf{##1}\hfil}%
  \settowidth{\labelwidth}{\textbf{#1}}%
  \setlength{\leftmargin}{\labelwidth+\labelsep}}}
 {\end{list}}
\begin{deflist}
\item[First] This is a short term.
\item[Long term] This is a long term.
\item[Even longer term] A very long term.
\end{deflist}
\begin{deflist}[Even longer term]
    .....
\end{deflist}
```

First This is a short term.

Long term This is a long term.

Even longer term A very long term.

First	This is a short term.
Long term	This is a long term.
Even longer term	A very long term.

5 Playing with lengths

Lengths can be defined, set and changed by the following commands.

`\newlength{`*cmd*`}`	`\setlength{`*cmd*`}{`*len*`}`
`\addtolength{`*cmd*`}{`*len*`}`	
`\settowidth{`*cmd*`}{`*text*`}`	`\width`
`\settoheight{`*cmd*`}{`*text*`}`	`\height`
`\settodepth{`*cmd*`}{`*text*`}`	`\depth`
	`\totalheight`

The new `\settoheight` and `\settodepth` commands, in analogy with the `\settowidth` command, already present in LaTeX 2.09, allow one to 'measure' the height and depth of some TeX material. The lengths `\width`, `\height`, `\depth`, and `\totalheight` are also new in LaTeX2$_\varepsilon$, and can be used inside the box commands described in the next section.

For ease of reference an overview of TeX's units of length is given below.

`sp`	scaled point (65536 sp = 1 pt) TeX's smallest unit.
`pt`	point = $\frac{1}{72.27}$ in = 0.351 mm
`bp`	big point (72 bp = 1 in), also PostScript point
`dd`	Didôt point = $\frac{1}{72}$ of a French inch, = 0.376 mm
`mm`	millimeter = 2.845 pt
`pc`	pica = 12 pt = 4.218 mm
`cc`	cicero = 12 dd = 4.531 mm
`cm`	centimeter = 10 mm =2.371 pc

`in`	inch = 25.4 mm = 72.27 pt = 6.022 pc
`ex`	height of a small ‘x’ for the current font
`em`	width of capital ‘M’ in current font
`mu`	math mode unit (18 mu = 1 em)

The following lines show how length commands are created, defined, changed, and used. They work, most of the time, both for rigid and rubber lengths.

```
\newlength{\Mylen} Mylen = \the\Mylen
```

Mylen = 0.0pt

```
\setlength{\Mylen}{10mm} Mylen = \the\Mylen
\setlength{\Mylen}{5mm plus 1mm minus .5mm}
\par Mylen = \the\Mylen  % Use a rubber length
```

Mylen = 28.45274pt

Mylen = 14.22636pt plus 2.84526pt minus 1.42262pt

```
\setlength{\Mylen}{1em} One em is \the\Mylen;
\addtolength{\Mylen}{1pc} add one pica \the\Mylen.
```

One em is 10.0pt; add one pica 22.0pt.

```
\settowidth{\Mylen}{May} The width is \the\Mylen
\settowidth{\Mylen}{\Large May} and now \the\Mylen.

\settoheight{\Mylen}{May} The height is \the\Mylen
\settoheight{\Mylen}{\Large May} and now \the\Mylen.

\settodepth{\Mylen}{May} The depth is \the\Mylen
\settodepth{\Mylen}{\Large May} and now \the\Mylen.
```

The width is 18.33pt and now 26.39519pt.

The height is 6.73pt and now 9.6912pt.

The depth is 2.18pt and now 3.13919pt.

‘Rubber’ (variable) lengths are very useful for placing information on the page.

`\fill`

This is a rubber length with a natural length of zero. It can stretch to any positive value and its value should not be changed!

`\stretch{`*dec_num*`}`

This is a more useful rubber length, since `\fill` is equivalent to `\stretch{1}`. More generally, `\stretch{`*dec_num*`}` has a stretchability of *dec_num* times `\fill`. It can be used to fine-tune the positioning of text horizontally or vertically.

Examples of the use of these stretchable lengths for controlling the horizontal and vertical page layout are given below.

```
\newcommand{\HS}[1][1.]{\hspace{\stretch{#1}}}
\begin{center}
left \hfill                              right\\
left \HS[.5]\fbox{$\frac{1}{3}$}\hfill right\\
left \HS         middle \hfill          right\\
left \hrulefill\ middle \hrulefill\     right\\
left \dotfill\                          right\\
left \dotfill\ \HS[.5] \dotfill\        right\\
left \dotfill\ \HS     \dotfill\        right\\
left \dotfill\ \HS[2.] \dotfill\        right
\end{center}
```

```
left                                            right
left          [1/3]                             right
left                  middle                    right
left ________________ middle __________________ right
left .......................................... right
left .................        ................. right
left ..............              .............. right
left ..........                       .......... right
```

```
\documentclass{article}
\usepackage{times}
\thispagestyle{empty}
\newcommand{\HRule}{\rule{\linewidth}{1mm}}
\setlength{\parindent}{0mm}
\setlength{\parskip}{0mm}
\begin{document}
  \vspace*{\stretch{1}}
  \HRule
  \begin{flushright}
    \Huge Geoffrey Chaucer\\[5mm]
          The Canterbury Tales
  \end{flushright}
  \HRule
  \vspace*{\stretch{2}}
  \begin{center}
    \Large\textsc{London 1400}
  \end{center}
\end{document}
```

Geoffrey Chaucer

The Canterbury Tales

LONDON 1400

5.1 Page Markup—Several Kinds of Boxes

Boxes are at the very heart of TEX's basic typesetting paradigm, and LATEX provides several commands which make it easy to make use of this functionality.

```
\mbox{text}  \makebox[width][pos]{text}
\fbox{text}  \framebox[width][pos]{text}
```

In addition to centreing the text with positional argument `[c]` (the default), you can position the text flush left (`[l]`) or flush right (`[r]`). LATEX2$_\varepsilon$ also offers you an `[s]` specifier that will stretch your *text* from the left margin to the right margin of the box provided it contains some stretchable space. As already mentioned in the previous section, LATEX2$_\varepsilon$ also allows you to make use of four special length parameters inside the *width* argument of the box commands: `\width`, `\height`, `\depth`, and `\totalheight`. These parameters specify the natural size of the *text*, where `\totalheight` is the sum of `\height` and `\depth`.

The examples below show how these various parameters are used to control the layout in the box. Note that use is also made of the calc package, which allows arithmetic operations in the arguments of the commands.

```
\framebox{A few words of advice}             \par
\framebox[\width + 6mm][s]{A few words of advice}
\par  \framebox[1.5\width]{A few words of advice}
```

A few words of advice

A   few   words   of   advice

A few words of advice

```
\rule[lift]{width}{total_height}
```

Rules come in handy for controlling the height of a box. Together with the new LATEX commands for measuring the height and depth of your boxes, they allow you to perform micro-typographic adjustments for tuning the visual presentation of your document elements.

```
\newsavebox{\Maybox}\savebox{\Maybox}{\Large May}
\newlength{\Mdp}\settodepth{\Mdp}{%
                                \usebox{\Maybox}}
\newlength{\Mht}\settoheight{\Mht}{%
                                \usebox{\Maybox}}
\addtolength{\Mht}{\Mdp}

\framebox[1.6\width+1em][s]{\usebox{\Maybox}}
\quad
\framebox[1.6\width+1em][s]{\usebox{\Maybox}%
          \rule[-2\Mdp]{0mm}{2\Mht}}
```

May   May

Zero-width boxes are also useful in other circumstances.

```
\begin{center}
A centred sentence.\makebox[0cm][l]{$^{123}$}\\
Some more text in the middle.              \\
\makebox[0cm][r]{$^{321}$}A centred sentence.\\
\end{center}
\noindent\makebox[0cm][r]{\(\Leftrightarrow\)}%
As seen in the margin of the current line, boxes
with a vanishing width can stick out in the margin.
```

A centred sentence.^{123}
Some more text in the middle.
^{321}A centred sentence.

$\Leftrightarrow$As seen in the margin of the current line, boxes with a vanishing width can stick out in the margin.

5.2 Moving boxes

Boxes can be moved up or down by the command:

```
\raisebox{lift}[depth][height]{contents}
```

The simple example below shows its principle of use.

```
\begin{flushleft}
x111x \raisebox{-1ex}{downward} x222x     \\
x333x \raisebox{1ex}{upward} x444x        \\[1em]
x111x \raisebox{-1ex}[0cm][0cm]{downward} x222x\\
x333x \raisebox{1ex}[0cm]{upward} x444x
\end{flushleft}
```

x111x downward x222x
x333x upward x444x

x111x downward x222x
x333x upward x444x

A more useful example is the generation of text 'between' two rows in a table (by 'hiding' the boxes' content from TEX.)

```
\begin{center}
\begin{tabular}{|c|c|c|}                \hline
        & \multicolumn{2}{c|}{title}\\\cline{2-3}
\raisebox{1.5ex}[0cm][0cm]{100}
        & A        & B            \\\hline
20000000 & 10       & 10           \\\hline
\end{tabular}
\end{center}
```

100	title	
	A	B
20000000	10	10

Finally, when your printer driver allows it, you can rotate boxes. In this case the use of the various box dimension parameters becomes apparent.

```
\newcommand{\DoT}[1]{\begin{turn}{45}#1\end{turn}}
x1 \DoT{\fbox{Not so Good.}} x2
\DoT{\raisebox{\depth}{\fbox{I like it.}}} x3
\DoT{\raisebox{-\height}{\fbox{Bad Choice.}}} x4
```

x1 Not so Good. x2 I like it. x3 Bad Choice. x4

5.3 Placing parboxes and minipages

In LATEX 2.09, boxes, which can contain more than one paragraph are defined as follows.

```
\parbox[pos]{width}{text}
\begin{minipage}[pos]{width}
    text
\end{minipage}
```

A simple example of its use is the following

```
\parbox{.3\linewidth}{This is the
 contents of the left-most parbox.}
\hfill Centerline \hfill
\parbox{.3\linewidth}{This is the right-most
 parbox. Note that the typeset text looks
 sloppy because \LaTeX{} cannot nicely balance
 the material in these narrow columns.}
```

This is the contents of the left-most parbox.	Centerline	This is the right-most parbox. Note that the typeset text looks sloppy because LATEX cannot nicely balance the material in these narrow columns.

5.4 Generalized parboxes and minipages

Sometimes it is helpful to predefine the vertical dimension of a paragraph box. For this LATEX2ε has additional optional arguments for `minipage` and `\parbox`.

```
\parbox[pos][height][inner-pos]{width}{text}
\begin{minipage}[pos][height][inner-pos]{width}
    text
\end{minipage}
```

The *inner-pos* determines the position of *text* within the box. It can be `t`, `c`, `b`, or `s`. If not specified, the value of *pos* will be used. You can think of *height* and *inner-pos* as the vertical equivalent of the *width* and *pos* arguments of a `\makebox`. If you use the `s` position the *text* will be vertically stretched to fill the given *height*. Thus, in this case you are responsible for providing vertically stretchable space if necessary using, for example, `\vspace` or `\vfill` commands.

As with the other box commands you can use `\height`, `\totalheight`, and so on to refer to the natural dimensions of the box when specifying the optional argument.

```
xx \fbox{\parbox[b][1.5\height][s]
         {30mm}{Some text on top. \par\vfill
                In the middle. \par\vfill
                And a few lines on the
                bottom of the box.}}
   \fbox{\parbox[b][\height+\baselineskip][s]
         {30mm}{This time a few lines on the
                top of the box. But only one
                line \par\vfill down here.}} xx
```

xx	Some text on top. In the middle. And a few lines on the bottom of the box.	This time a few lines on the top of the box. But only one line down here.	xx

5.5 Manipulating Boxed Material

Material can be typeset once and then stored inside a named box, so that its contents can be retrieved later. LATEX offers the following commands for dealing with this situation.

`\newsavebox{`*cmd*`}`	declare box
`\sbox{`*cmd*`}{`*text*`}`	fill box
`\savebox{`*cmd*`}[`*width*`][`*pos*`]{`*text*`}`	fill box
`\usebox{`*cmd*`}`	use contents
`\begin{lrbox}{`*cmd*`}` *text* `\end{lrbox}`	fill box

Note that the environment `lrbox` is an addition in LATEX2ε. *cmd* should be a box register previously allocated with `\newsavebox`. The environment `lrbox` will save *text* in this box for later use with `\usebox`. Leading and trailing spaces are ignored. Thus, `lrbox` is basically the environment form of `\sbox`. You can make good use of this environment if you want to save the body of some environment in a box for further processing. For example, the following code defines the environment `fminipage` that works like a `minipage` but surrounds its body with a frame. Note the use of the optional argument for controlling the width of the boxed minipage, and the fact that verbatim material can be used inside. To be able to do the arithmetic operations you will also need to have the calc package loaded.

```
\newsavebox{\fminibox}
\newlength{\fminilength}
\newenvironment{fminipage}
    [1][\linewidth]% default width is \linewidth
  {\setlength{\fminilength}%
        {#1-2\fboxsep-2\fboxrule}%
   \begin{lrbox}{\fminibox}%
   \begin{minipage}{\fminilength}}
  {\end{minipage}\end{lrbox}%
   \noindent\fbox{\usebox{\fminibox}}}

\begin{fminipage}
  In this environment verbatim text like
  \verb=\fminibox= can be used.
\end{fminipage}
```

In this environment verbatim text like `\fminibox` can be used.

```
\begin{fminipage}[.5\linewidth]
  ....
\end{fminipage}
```

In this environment verbatim text like `\fminibox` can be used.

5.6 For hackers only: a list with two optional parameters

What if we want to define a command or environment with, e.g., *two* optional arguments? Suppose we want a list where we are able to specify not only the width of the label, but also whether the list should be 'dense' or not, i.e., we want a syntax like:

```
\begin{Description}[<margin>][<style>]
```

In this case we use a trick (and the packages calc and ifthen) and introduce a multiple-step definition. The example also shows how one can parameterize the various typographic parameters for the list so that they can be controlled more easily (e.g., the alignment of the label, its font, and the width of the margin).

```
\newcommand{\Descriptionlabel}[1]{%
       \mbox{\Descriptionfont #1}\hfil}

\newcommand\Descriptionfont{\itshape}
\newcommand\Descriptionmargin{}

\newenvironment{Description}[1][\kern\leftmargin]
 {\renewcommand\Descriptionmargin{#1}\xdescription}
 {\endlist}

\newcommand{\xdescription}[1][normal]{%
 \list{}{\settowidth{\labelwidth}%
  {\mbox{\Descriptionfont\Descriptionmargin}}%
  \setlength{\itemindent}{0pt}%
  \setlength{\leftmargin}{\labelwidth+\labelsep}%
  \let\makelabel\Descriptionlabel
  \ifthenelse{\equal{#1}{compact}}%
   {\setlength{\itemsep}{0pt}%
    \setlength{\topsep}{.5\topsep}}{}%
 }}

Text before text before text before text before
\begin{Description}
  \item[First] This is the first item in the list.
  \item[Veryyy long] This is a veryyy long item.
  \item[] This is an empty item.
\end{Description}

\begin{Description}[Veryyy long]
  ...
\end{Description}

\begin{Description}[Veryyy long][compact]
  ...
\end{Description}
```

Text before text before text before text before

First This is the first item in the list.

Veryyy long This is a veryyy long item.

This is an empty item.

First This is the first item in the list.

Veryyy long This is a veryyy long item.

This is an empty item.

First This is the first item in the list.
Veryyy long This is a veryyy long item.
This is an empty item.

6 Font commands—an overview

This section covers the user commands in LaTeX2ε for specifying fonts, both in text as in mathematics. We also mention some of the more popular fonts packages and say a few words on compatibility with LaTeX 2.09.

The first question you can naturally ask yourself is why new font commands were introduced at all. To answer this question let us mention that LaTeX 2.09 font commands had a few idiosyncrasies:

- their syntax, i.e., `{\it foo}` rather than `\it{foo}`, which is unlike the syntax of (most) other LaTeX commands (safe the size-changing series), which are specified with arguments;
- the font commands were not *orthogonal*, e.g., `\bf\sf` produces medium-weight sans, i.e., only the inner font command is honored;
- some fonts substitutions were taking place 'behind our backs', e.g., `\tiny\tt` produces tiny roman, since it was assumed that at such a small size the difference is hardly visible, so that one can as well use a font already loaded;
- italic corrections must be introduced by hand, e.g., one has to write `{\em my text\/}`, and even this is not correct in all circumstances.

LaTeX2ε addresses these problems by introducing the following new text font commands:

`\textmd{`This is medium text`}`
`\textbf{`**This is bold text**`}`
`\textup{`This is upright text`}`
`\textit{`*This is italic text*`}`
`\textsl{`*This is slanted text*`}`
`\textsc{`THIS IS SMALL CAPS TEXT`}`
`\textrm{`This is roman text`}`
`\textsf{`This is sans text`}`
`\texttt{This is typewriter text}`

Plus `\emph{`*This is emphasized text*`}`.

The size changing commands remain unchanged (i.e., `\large`, `\scriptsize`, etc. are still valid).

These commands do not have the problems of the LaTeX 2.09 commands, because:

- their syntax is the same as for the other LaTeX commands;
- `\textbf{\textsf{`**text**`}}` produces bold sans;

- `{\tiny\texttt{text}}` produces tiny typewriter;
- `\emph{`*text*`}` does *not* need `\/`.

Note that there are still some restrictions, for instance, `\textbf{\texttt{text}}` produces medium typewriter for lack of a bold Computer Modern typewriter font, but at least LaTeX2ε warns you about the substitution.

In the area of math fonts, LaTeX2ε provides the following new commands:

`\mathnormal{`*This is normal math italic*`}`
`\mathcal{`MATH CALLIGRAPHIC`}`
`\mathrm{`This is roman in math`}`
`\mathbf{`**This is bold in math**`}`
`\mathsf{`This is sans in math`}`
`\mathit{`*This is text italic in math*`}`
`\mathtt{This is typewriter in math}`

Note that these commands do not work outside mathematics.

It is now relatively easy (if you have the fonts) to replace Computer Modern with other font families. Various packages for popular fonts are already available, for example:

- In the area of PostScript fonts `\usepackage{times}` provides Adobe Times, `\usepackage{palatino}` Adobe Palatino, `\usepackage{lucidbrb}` Y&Y's LucidBright and LucidaNewMath, etc.;
- `\usepackage{amssymb}` provides the AMS fonts;
- `\usepackage{pandora}` allows you to use the Pandora fonts;
- `\usepackage{euler}` lets you experiment with Hermann Zapf's Euler font family.

LaTeX 2.09's old font commands (`\rm`, `\bf`, etc.) are still available in LaTeX2ε, but they are *not* part of the 'kernel'. They are now defined in the document class files, where the definitions of the size changing commands, like `\huge`, `\tiny`, have always resided. It is thus up to the document designer to define how the old font commands behave. Note, however, that for the 'standard classes' (article, book, etc.) the old font commands behave as they did in LaTeX 2.09.

One more word about about LaTeX 2.09 compatibility. A document beginning with `\documentstyle` is run in *compatibility mode*, which emulates LaTeX without NFSS. If you want to emulate LaTeX with NFSS you should say:

```
\documentstyle[newlfont]{...}
```

7 Standard Classes in LaTeX2ε

This section discusses the files that come with the LaTeX2ε distribution and lists some of the packages which are already adapted to LaTeX2ε.

Files associated to LaTeX2ε are characterized by the extensions:

name`.cls` for class files;
name`.clo` for external option files;
name`.sty` for package files
name`.cfg` for runtime configuration files

The 'standard' document classes distributed with LaTeX2ε are article, report, book, letter, slide, proc, and ltxdoc. Below, we say a few words about each one of them.

article, report, book
- they behave like the old LaTeX 2.09 styles;
- twocolumn and openbib are now internal options;
- a set of new internal options was added: a4paper, a5paper, b5paper, letterpaper, legalpaper, executivepaper, landscape.

letter
- it behaves like the old style;
- it has a set of new internal options: a4paper, a5paper, b5paper, letterpaper, legalpaper, executivepaper.

slide
- it behaves like the old style, but used with LaTeX2ε;
- it supports local font configuration by looking for `sfonts.cfg`;
- it has a set of new internal options: a4paper, a5paper, b5paper, letterpaper, legalpaper, executivepaper, landscape;
- the option twocolumn is not supported.

proc
- it is no longer an option but a *document class*;
- it is built on the article class;
- it disallows options a5paper, b5paper, onecolumn, titlepage.

ltxdoc
- it is used to format the LaTeX2ε source code;
- it is built on article and requires the doc package;
- it looks for the configuration file `ltxdoc.cfg`;
- it defines the commands `\DocInclude` and `\GetFileInfo`;
- it disallows the option a5paper.

Presently the following packages are available:

- ifthen, for building control structures. It provides on top of `\ifthenelse` and `\whiledo` available previously with LaTeX 2.09, the new commands `\newboolean`, `\setboolean`, and `\boolean`.
- makeidx, showidx, to help you make indexes.
- doc, shortvrb, for generating class and package file documentation.
- oldlfont, newlfont, for compatibility with LaTeX 2.09 and version 1 of the NFSS.
- latexsym LaTeX2ε no longer loads the `lasy` fonts by default; if needed they become available by loading this package. Note that they are not necessary when either amsfonts or amssymb is used.
- exscale allows for different math extension fonts.
- eufrak and euscript give access to the Euler fraktur and script alphabets, oldgerm to Haralambous' beautiful old German fonts, while pandora allows you to

use Billawala's Pandora font family.

- syntonly will make LaTeX only check the syntax of your document, while tracefnt, with its various options errorshow, warningshow, infoshow, and debugshow, allows you to trace NFSS as LaTeX processes your document.
- varioref provides a way to automatically adapt the text of a reference, depending on the position of the `\label`.

Many other packages on CTAN already work with LaTeX2$_\varepsilon$ or will soon be converted. In the first category one finds a4, epsfig, exams, labels, layout, the NTG document class family artikel1, rapport3, etc., subeqnarray, psnfss, textfit, while the latter contains the babel collection, changebar, ltugboat and friends, the 'Mainz' packages array, ftnright, multicol, theorem, verbatim, and supertabular.

8 Miscellaneous goodies

This section describes some features which are perhaps not used every day, but which can come in handy for solving certain practical document preparation problems.

8.1 Controlling page breaks

Sometimes, when preparing the *final* version of your document, you might need to help LaTeX break the pages in a suitable way. LaTeX 2.09 had commands like `\clearpage`, `\samepage`, etc., while LaTeX2$_\varepsilon$ provides, in addition, commands which increase or decrease the height of the *current* page from its 'natural' height `\textheight` by an amount *size*.

```
\enlargethispage{size}
\enlargethispage*{size}
```

For example, `\enlargethispage{-\baselineskip}}` decreases the length of the current page by one line, while `\enlargethispage*{2\baselineskip}}` makes it two lines longer than usual.

The starred form also shrinks any vertical white space on the page as much as possible, so as to fit the maximum amount of text onto the page.

8.2 Floats

A new command and a new 'float specifier' will allow you more control over LaTeX's float placement algorithm.

```
\suppressfloats[placement]
```

This command stops any further floating environments from being placed on the current page. The optional argument *placement* can be either `t` or `b` (not both), and in this case the restriction applies only to putting further floats at the top or at the bottom.

Extra float placement specifier: `!`

This can be used, along with at least one of `h`, `t`, `b` and `p`, in the float placement optional argument.

If a `!` is present then, just for this particular float, whenever it is processed by the float mechanism the following are ignored:

- all restrictions on the number of floats which can appear;
- all explicit restrictions on the amount of space on a text page which may be occupied by floats or must be occupied by text.

The mechanism will, however, still attempt to ensure that pages are not overfull and that floats of the same type are printed in the correct order.

Note that its presence has no effect on the production of float pages.

A `!` placement specifier overrides the effect of any `\suppressfloats` command for this particular float.

References

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

[2] Leslie Lamport. *LaTeX—A Document Preparation System. Second edition* Addison-Wesley, Reading, USA, 1994.

[3] Frank Mittelbach and Rainer Schöpf. Towards LaTeX 2.10. *TUGBoat*, 10(3):400–401, November 1990.

[4] Frank Mittelbach. E-TeX: Guidelines for future TeX extensions. *TUGBoat*, 11(3):337–345, September 1990.

[5] Frank Mittelbach and Chris Rowley", LaTeX 2.09 $\hookrightarrow$ LaTeX3. *TUGBoat*, 13(1):96-101, April 1992.

[6] Chris Rowley. LaTeX3 update. *TUGBoat*, 13(3):390-391, October 1992.

[7] Frank Mittelbach and Chris Rowley and Michael Downes. Volunteer work for the LaTeX3 project. *TUGBoat*, 13(4):510-515, December 1992.

[8] Frank Mittelbach and Chris Rowley. Volunteer work for the LaTeX3 project. *TeX and TUG NEWS*, 3(1):7-11, January 1994.