Baskerville

The Annals of the UK TEX Users' Group **Editor: Editor: Sebastian Rahtz** **Vol. 4 No. 2**

ISSN 1354–5930 **February 1998**

Articles may be submitted via electronic mail to `baskerville@tex.ac.uk`, or on MSDOS-compatible discs, to Sebastian Rahtz, Elsevier Science Ltd, The Boulevard, Langford Lane, Kidlington, Oxford OX5 1GB, to whom any correspondence concerning *Baskerville* should also be addressed.

This reprint of *Baskerville* is set in Times Roman, with Computer Modern Typewriter for literal text; the source is archived on CTAN in `usergrps/uktug`.

Back issues from the previous 12 months may be ordered from UKTUG for £2 each; earlier issues are archived on CTAN in `usergrps/uktug`.

Please send UKTUG subscriptions, and book or software orders, to Peter Abbott, 1 Eymore Close, Selly Oak, Birmingham B29 4LB. Fax/telephone: 0121 476 2159. Email enquiries about UKTUG to `uktug-enquiries@tex.ac.uk`.

Contents

I Editorial

1 *Baskerville* articles needed

We need material for *Baskerville*! Please send your interesting articles to the editor, and delight fellow TEX users. *Please note the following schedule of copy deadlines*:

Issue	*Submit material for publication*	*Submit last-minute notices*	*PostScript file sent to production team*
4.3	May 23	May 30	Jun 6
4.4	Aug 15	Aug 22	Aug 29
4.5	Oct 17	Oct 24	Oct 31

Each issue of *Baskerville* will have a special theme, although articles on any TEX-related subject are always welcome. Contributions on the themes for the remainder of 1994 are eagerly solicited:

☞ *Baskerville* 4.3 will concentrate on graphics;
☞ *Baskerville* 4.4 will be a back to basics special issue on mathematical and tabular typesetting;
☞ *Baskerville* 4.5 will try and go beyond TEX, to see what is on the horizon.

2 Apology!

Alan Jeffrey's article in *Baskerville* 4.1 was the victim of the very font-encoding problems which his *fontinst* software is designed to solve. Between the proof copies created on a machine in Geneva, and the final copy created at Aston, one of the font tables turned to garbage. It came about because the two TEX setups used the same *virtual* fonts, but differently-encoded *raw* fonts, so the virtual Times assumed that glyphs were at positions where in fact something very different lived. For the record, the *correct* table showing Times in TEX layout is as follows (answers on a postcard, please, explaining why the lslash and Lslash glyphs are also here):

Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	Φ	Ψ	Ω	ff	fi	fl	ffi	ffl
ı	▮	`	´	ˇ	˘	¯	˚	¸	ß	æ	œ	ø	Æ	Œ	Ø
■	!	”	#	$	%	&	’	(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	¡	=	¿	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[	“	]	ˆ	˙
‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	–	—	˝	˜	¨
										Ł					
										ł					

Readers who are fascinated by the whole subject of fonts and encodings will be delighted to find *another* practical article on the subject in this *Baskerville*; we hope that other people will grow to appreciate *Baskerville*'s continuing definitive saga on the subject that won't go away.

2.1 Colophon

This issue of *Baskerville* was to have dealt entirely with practicalities of LATEX 2ε, to go with the group's March meeting on this major new release of the package; we do include a detailed overview of 2ε, but some articles will not be ready until future issues, and in the meanwhile we find plenty to occupy the modern TEXie.

This issue of the journal was created entirely with the test distribution of LATEX 2ε, using Y&Y's implementation of TEX, and printed on a Hewlett Packard LaserJet 4.

Encoding-hounds will appreciate knowing that this *Baskerville* was processed using a slightly extended Windows ANSI encoding; this was achieved by using Y&Y's comprehensive font tools to reencode the fonts. This allowed direct preview using *dviwindo* and Adobe Type Manager. Having the base fonts in Windows ANSI then allowed me to build a full set of EC-encoded virtual fonts (using Jeffrey's *fontinst*) for when I want a portable source or access to the full range of EC characters. Sometimes you *can* have the best of all possible worlds. . .

Baskerville is set in ITC New Baskerville Roman and Gill Sans, with Computer Modern Typewriter for literal text. Production and distribution was undertaken in Cambridge by Robin Fairbairns and Jonathan Fine.

II Letters to the editor

1 TEX on the Atari — questions

I became aware of the power and style of TEX through reading *Digital Typography* by Richard Rubinstein a few months ago; consequently, I am a newcomer to both TEX and UKTUG. I was advised to stick to plain TEX to begin with and use the version of AtariTEX created by Christoph Strunk of Bochum based on TEX version 3.1, METAFONT version 2.7 and μEmacs version 3.09. Unfortunately, Strunk (like many another German computer guru) provides all his documentation in fairly difficult German and doesn't answer letters, so I am appealing to the *Baskerville* readership to see if there is anyone 'out there' who is familiar with—or even uses—AtariTEX and could help me with the following, relatively minor, problems.

1. How can I install a more up-to-date version of μEmacs? As a user of a mouse driven computer, I find it very hard that I am forced to edit text without my mouse and cannot make another version of μEmacs (say 3.10, being modest in my aspirations), let alone a totally different editor *with* a spell checker, integrate properly into the shell environment.
2. Strunk's laser printer driver, dvi_lj2, is designed for the Hewlett-Packard LaserJet II – is there one which can take advantage of the features of the LaserJet III? In particular, I would like to be able to print in landscape from time to time. It may well be that there is now one for the LaserJet 4 which will allow printing at 600 dpi.
3. My German is not strong, but am I right in thinking that Strunk has forgotten to include an option in dvi_lj2 which would allow double sided printing?

A paragraph for 'strangers' to the world of AtariTEX. The marketing policy of Atari Corporation does indeed tend to support an attitude of massive indifference to their products, but the TT computer is a powerful and user-friendly machine. (For the technical, it uses a 68030 processor and a 68882 floating point coprocessor running at 32 MHz in conjunction with the GEM interface to humans.) Strunk's version of AtariTEX appears to me to be convenient to use, and has many features which I have yet to explore thoroughly. On a lighter note, I have found two which have not been mentioned by users of TEX on other platforms – incorporating images is trivially easy, I normally use .IMG format, but others may be used; there is a PostScript printer driver and also one which produces files of each page in .IMG format. This driver provides a very cumbersome way of printing in landscape as well as its more important function of allowing the incorporation of TEX output into my word processor.

John Bowsher
Bonner's Field
Northbourne Road
Great Mongeham
Deal, Kent CT14 0LD

2 Typesetting the TEX logo properly?

. . . Working on a logo has made me realise that, in my opinion, *Baskerville* doesn't print the official TEX logo correctly. When cmr is used the E in the middle is joined to the serifs on the initial T and final X; in ITC New Baskerville Roman, the E is floating uncomfortably on its own. That cannot have been the intention of the original designer of the logo.

John Bowsher

[Editor's reply: You are quite right; the TEX logo could use a redefinition for each font family. Philip Taylor brought up the same issue when the TUG'93 proceedings were set in Lucida Bright. But I think we need clarification from Knuth — he might say, for instance, that we should always print the TEX logo in Computer Modern.]

III LaTeX 2ε, an overview

Michel Goossens
CN Division, CERN, CH1211 Gen
lqeve 23, Switzerland
goossens@cern.ch

Summary

This article gives an overview of the new or extended user commands available with LaTeX 2ε, 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 LaTeX 2ε?

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, 6, 7, 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 LaTeX 2ε, 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 LaTeX 2ε 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 LaTeX 2ε 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 LaTeX 2ε 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.

Initial commands

$\boxed{\texttt{\textbackslash NeedsTeXFormat\{}\textit{format-name}\texttt{\}[}\textit{release-date}\texttt{]}}$

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 LaTeX 2ε. 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. LaTeX 2ε provides the following syntax to facilitate this

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

When your document file is run through LaTeX 2ε 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.

Preamble commands

The next commands in the preamble are specifically designed to differentiate LaTeX 2ε documents from those needing LaTeX 2.09.

$\boxed{\texttt{\textbackslash documentclass[}\textit{option-list}\texttt{]\{}\textit{class-name}\texttt{\}[}\textit{release-date}\texttt{]}}$

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.

$\boxed{\texttt{\textbackslash documentstyle[}\textit{option-list}\texttt{]\{}\textit{style-name}\texttt{\}[}\textit{release-date}\texttt{]}}$

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 LaTeX 2ε. Note, however, that in this mode, you *cannot* use any of the LaTeX 2ε extensions described in this article.

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

$\boxed{\texttt{\textbackslash usepackage[}\textit{option-list}\texttt{]\{}\textit{package-name}\texttt{\}[}\textit{release-date}\texttt{]}}$

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 LaTeX 2ε 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.

$\boxed{\texttt{\textbackslash listfiles}}$

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.

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 LaTeX 2ε 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.

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. If there is a second optional parameter to `\newcommand`, etc., then the command being defined has an optional parameter, with *default* value if not specified 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 LaTeX 2ε 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 $x_0,\ldots x_n,\quad z_0,\ldots z_n$

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 LaTeX 2ε 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 LaTeX 2ε, 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 18.32983pt and now 21.9958pt.
The height is 6.50987pt and now 7.81186pt.
The depth is 2.17491pt and now 2.6099pt.
“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*`}`

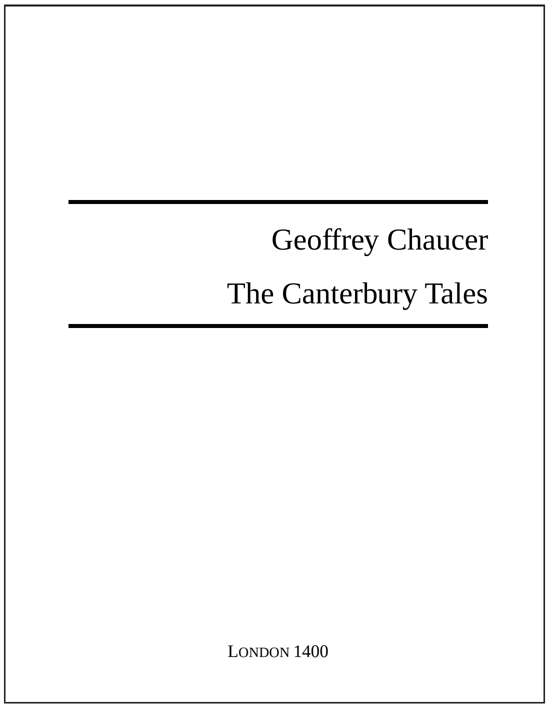
This is a more useful rubber length, since `\fill` is equivalent to `\stretch{1}`. More generally, `\stretch{`*dec_num*`}` has a stretchability of *dec_num* times `\fill`. It can be used to fine-tune the positioning of text horizontally or vertically.

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 $\boxed{\frac{1}{3}}$ 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

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]`). LaTeX 2ε 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, LaTeX 2ε 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}}
```

$\boxed{\text{May}}$ $\boxed{\text{May}}$

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.

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

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.

Generalized parboxes and minipages

Sometimes it is helpful to predefine the vertical dimension of a paragraph box. For this LaTeX 2ε 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

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 LaTeX 2ε. *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.

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 LaTeX 2ε 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 honoured;

- some font 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.

LaTeX 2ε 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 LaTeX 2ε warns you about the substitution.

In the area of math fonts, LaTeX 2ε provides the following new commands:

`\mathnormal{`*This is normal math italic*`}`
`\mathcal{`$\mathcal{MATH\ CALLIGRAPHIC}$`}`
`\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 LucidaBright 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 LaTeX 2ε, 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 LaTeX 2ε

This section discusses the files that come with the LaTeX 2ε distribution and lists some of the packages which are already adapted to LaTeX 2ε.

Files associated to LaTeX 2ε 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 LaTeX 2ε 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 LaTeX 2ε;
- 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 LaTeX 2ε 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 LaTeX 2ε 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 LaTeX 2ε 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 LaTeX 2ε 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.

Bibliography

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.

IV Using virtual fonts with Textures

Laurent Siebenmann
lcs@matups.matups.fr

1 Introduction

Virtual font support was first provided in mid 1993 in two popular implementations of TEX for the Macintosh, namely Textures and OzTEX.

For Textures users, virtual fonts offer the first oportunity to exploit the high performance of the scalable CM/PS fonts, which Textures provides, in a way that permits perfect linebreaking and kerning for accented west European languages.

This article[1] is very much user-oriented in the sense that the main point is to deliver performance we have we have been dreaming of since Knuth's virtual font standard was unveiled in 1989, using just the equipment and software we have today. In particular its sections are called 'tutorials'.

Users of other TEX installations may find some 'window shopper' interest in the following pages. Textures is one of the first implementations of TEX to have made a living reality of 'electronic paper' in the sense that through most of the TEX production cycle it is more pleasant to use preview than printed copy. This results from a number of features:

– Smooth linkage of editor and preview and TEX. Typically just a couple of seconds separates typing from preview.
– Graphics preview (albeit bitmapped).
– Scalable fonts of high quality. The ATM (Adobe Type Manager) system for screen rendition of 'hinted' Adobe Type 1 fonts is the secret here along with the CM/PS fonts.
– Fast preview. The user passes from one A4 preview page to the next in about .5 sec on a Mac with 33mhz, 68040 microprocessor. Compared with other previewers, Textures has a speed advantage factor of 2 or more (processor power considered). That can be expressed as an advantage of 4 years progress since average computer speed doubles in about that period.

Unfortunately this delightful environment has not been well adapted to use with European languages using accents, quite simply because there have been no (ready-made) accented characters attached to the CM/PS fonts — until finally virtual fonts have made this possible as the tutorials will explain.

The issue of preview speed is troublesome to developers of virtual font previewers. The problem is very evident with Textures where a slowing factor of two is currently observed. It was recently mentioned by Berthold Horn [*Baskerville* 4.1, page 10] as a possible argument against virtual fonts. Since it tends to spoil the 'electronic paper' feature of Textures that I value so highly, I was very concerned.

After having had a closer look I am reassured. The devices presented in this tutorial will suffice to skirt this difficlty, *and*/*or* a frontal attack will surely be launched to recover lost preview speed by reprogramming. Textures users should take heart and employ virtual fonts forthwith.

2 Tutorial 1: Textures virtual font resources

A 'virtual font' is in essence a 'collage' of character images or 'glyphs' taken from other fonts (virtual or ordinary). Its shapes are composites rather than distinctly new.

For example, the virtual font dmr10 contains a composite glyph `Eacute` made up of an acute accent `acute` from cmr10 combined with `E` also from cmr10, each precisely positioned in a way specified by the virtual font. One might object that the same result on paper can always be produced by TEX without virtual fonts. True, but to produce

[1] Peter Galko (`galko@trix.genie.uottawa.ca`) spurred this didactic effort by requesting consumer oriented information on virtual fonts for Textures. Special thanks are due to him for reading and criticizing. Thanks also to Peter and Nicolas Jungers (`nj@fusl.ac.be`) for sharing the work to package and post the DM virtual fonts for Textures.

the result without can be painful. Particular painful would be the hyphenation of words containing `Eacute` and the kerning (or ligature) of `Eacute` with other letters.[2]

Virtual fonts are ideal for efficiently building fonts with accented characters for French and other European languages. The first were perhaps the DM fonts of Nicolas Brouard and Wayne Sullivan, created for the 1990 GUTenberg PC TEX distribution; they extend the CM fonts by adding accented characters in the positions specified by the layout standard referred to as 'ISOLatin1', essentially the top quarter (i.e., the 192–255 positions) of the 'Cork' European norm (alias the DC font layout).

In standard TEX implementations, virtual fonts are defined by two files: a `.tfm` binary file which TEX uses in the typesetting phase to determine character sizes, kerning and ligatures; and a `.vf` binary file which the print driver uses to find out how to render a character in the virtual font.

The virtual font specification adopted by Textures is non-standard. But for good reasons: on current hard discs the minimal file size is 5K octets, which is usually significantly larger than a `.vf` and `.tfm` file combined, and the number of files the operating system can handle efficiently is often 'too small'. This causes much time and disk space to be wasted. Imagine one wants to introduce 200 virtual fonts. With standard TEX this costs 400 files and say 2M octets (M for million) of disk space plus time costs for handling the files.

What about Textures? Textures puts the information contained in the `.vf` and `.tfm` files into a `*TFM` resource which is then made part of a font suitcase file. Thus to include these 200 virtual fonts, you add *one* font suitcase file containing 200 `*TFM` resources, each of which combines .tfm and .vf information. Each tends to be 1–5K octet so the suitcase file will be less than 1M octet. Very neat, and worth the change.

3 Tutorial 2: Textures 'Type 1' virtual TEX fonts

Adobe Type 1 fonts are scalable PostScript fonts that contain 'hints' for optimal rasterization on screen via ATM, and on paper via PostScript printers. The quality of a rasterized images that ATM produces, operating at interactive speeds, is comparable to that produced by METAFONT. Furthermore the speed of screen display via ATM is essentially the same as for bitmapped fonts — after a moderate startup time, even though ATM never stores bitmaps permanently.

Users should beware however that the good speed performance quoted is only realized when sufficient memory has been allocated to ATM. A typical TEX user needs about 700Kb. The sign that you need to allocate more RAM to ATM is continual hard disk access when you are scrolling back over material you have already previewed.

Since Textures switched to using Adobe Type 1 versions of CM fonts, users of ATM have benefitted from the possibility of previewing these fonts at any size. Unfortunately, these fonts do not include any characters with accents, which is a significant impediment to Europeans (the use of TEX's `\accent` primitive leaves much to be desired where hyphenation, kerning, ligatures and beauty are concerned.)

Although there is still no extended set of CM-face fonts with accented characters in Adobe Type 1 format, ordinary extensions of the CM fonts are available in bitmapped form (notably the DC fonts of Schwartz that use the Cork encoding, and the TM fonts of Taupin that are fully compatible with CM—both freely available). *However*, bitmapped screen fonts for prose are unreadable at most sizes, so they are disqualified in the ATM world that Blue Sky research has so wholeheartedly embraced by distributing its excellent CM/PS fonts as a standard part of Textures.

A few extended CM fonts in the form of virtual fonts have been in circulation since 1990, notably the DM fonts of Brouard-Sullivan. As implemented by Blue Sky research in Textures 1.6 of 1993 and later, the virtual font notion marries smoothly with Adobe Type 1 fonts and ATM—to the point where there is no obvious way for the user to see the difference between virtual and non-virtual Type 1 font. Thus a virtual font is for TEX's purposes 'of Adobe Type 1' as soon as the regular fonts on which it is based are themselves Adobe Type 1.

Often one understands by *extended CM font* any font that is encoded (essentially) like a CM font in the segment 0–127 and (essentially) according to the Cork norm in the range 128–255. One has potentially an enormous class of such virtual extensions for Adobe Type 1 prose fonts. As matters stand these can be generated by Jiří Zlatuška's `accents` package available on the CTAN servers. Unfortunately `accents` has to my knowlege been compiled only on PC's (no other binary has been posted). To my knowlege no virtual font produced by `accents` has ever been posted. This should be remedied.

What about Cork norm fonts? They disagree with CM fonts in the range 0–127. Consequently, they are not currently used in conjunction with (extended) CM encoding. (They could be — as explained in my article in the 1992 EuroTEX

[2] Experts know that virtual font glyphs can contain a bit more than the DM fonts do : rules and specials — the same sort of extras one finds in TEX's `dvi` output files. For more details, see Knuth's article in TUGboat.

proceedings — but there has not been sufficient demand.) I know of three Cork encoded virtual font series for Adobe Type 1 fonts, and all can be adapted to Textures using EdMetrics:

1. a series posted by Haralambous on `ftp.dmi.ens.fr` and derived from his Vulcano utility. This covers only the range 192–255.
2. a series posted by Nicolas Jungers on `matups.matups.fr`. It includes a sprinkling of characters in the range 128–191, notably `section` and `pounds` plus letters with 'hacek' accent.
3. a series generated using `fontinst` by Sebastian Rahtz on CTAN (in `fonts/metrics`)

All these postings cover a selection of commonly used fonts including the basic LaserWriter series.

4 Tutorial 3: Textures DM virtual fonts

Apart from some sample files, only one virtual font was distributed with Textures in 1993, namely MTMI, the MathTime math italic that builds on Times Italic and a 'raw' math font RMTMI. Obviously it would be a healthy thing for more virtual fonts to become available in Textures format. Blue Sky Research, the makers of Textures provide a tool called EdMetrics that (with version 0.7 or later) is capable of converting a virtual font specified in its human readable text form as a '.vpl' file into a Textures `*TFM` resource format and placing the resource in a suitcase. The '.vpl' file is, in TEX's standard font administration, used to construct both the .vf and .tfm files for the font using a utility 'vptovf'. EdMetrics includes something like 'pltovf'. Recall the etymologies: '.tfm'='TeX font metric'; '.vf'= 'virtual font' and '.vpl'='virtual property list'.

To introduce Textures users to virtual fonts, we have made available the original series of DM virtual fonts in Textures format. There are 55 of them. We provide them in a Textures font suitcase called 'DM metrics' occupying 210Kb. They are packaged as a binhexed self-extracting Macintosh archive file whose name begins DM-VFs (see 'References' below).

Install these VF's by simply pushing the suitcase into the 'TeX Metrics' or 'TeX Fonts' folder. More on exploiting them in later tutorials.

If you examine one of these `*TFM`'s with EdMetrics you will see that nothing tricky is involved (it seems); we have done nothing original—such as create a new virtual font. For that you would be well advised to join the Metafont email list.

Even without further coaching, you will manage to produce more VFs for Textures starting from generic virtual fonts posted on the CTAN TEX archives. It's worth trying. EdMetrics may cause crashes — but be assured that Blue Sky Research is working on it. Soon the whole process will be dispatched with a few mouse clicks.

5 Tutorial 4: Trying out extended CM fonts for Textures

The following tutorial will let you try out the Textures DM virtual fonts presented in the previous tutorial. It is framed for Plain TEX and its direct extensions. You will probably find that it works in any environment that: (a) is able use virtual fonts, (b) uses Computer modern fonts, and (c) uses the font nomenclature of Plain TEX. In particular, NFSS users are advised to revert to Plain TeX for the trials.

Try the following instructive exercises. A later tutorial will propose an efficient working setup (valid for NFSS).

1. Install these VF's by simply pushing the suitcase into the 'TeX Metrics' or 'TeX Fonts' folder
2. Put these fonts into service in place of the similar CM fonts by the command `\DMtext` defined as follows:

```
\def\gobble#1{}
\def\repeatfont{\edef\temp{%
 \expandafter\string\the\font}%
 \csname\expandafter\gobble\temp\endcsname}

\def\DMtext{%
  \font\tenrm=dmr10
  \font\ninerm=dmr9
  \font\tenbx=dmbx10
  \font\ninebx=dmbx9
  \font\tenti=dmti10
  \font\nineti=dmti9
  \font\tensl=dmsl10
  \font\ninesl=dmsl9
  \repeatfont}
```

and return by using `\CMtext`:

```
\def\CMtext{%
  \font\tenrm=cmr10
  \font\ninerm=cmr9
  \font\tenbx=cmbx10
  \font\ninebx=cmbx9
  \font\tenti=cmti10
  \font\nineti=cmti9
  \font\tensl=cmsl10
  \font\ninesl=cmsl9
  \repeatfont}
```

The greatest potential weakness of these provisional macros is their dependence on Plain TEX's nomenclature. In other formats `\tenrm` (etc.) may become another control sequence. The above DM fonts are the main prose fonts of the first DM distribution for Textures. If you have more DM fonts installed you can extend the list at the cost of slowing the macros. By using the command `\showthe\font` you can see whether the new fonts are being used.

TEX user: Where do I put these macro definitions, and how do I use them?
The Lion: Your style file for your current article would be a natural place.

3. To see the characters available in any font (say dmr10) compile the file `FontTable256Hack.tex` after decommenting the line

```
 \def\fontname{dmr10 scaled 1720}
```

Just one line of this sort should be active. Preview and print to learn what characters the DM fonts offer.

4. To be able to access the new accented characters using Knuth's syntax `\'e` for `e-acute` etc. put `\input caesarcm` at the head of your typescript.

TEX user: But that syntax already works *without* CaesarCM!
The Lion: Yes, but it has been giving you *composite* characters in TEX and they slightly damage kerning, ligatures, and hyphenation. To see a modest objective difference, typeset

```
    \CMtext \'ex\'ex\'ex\'ex\'ex\'ex\'ex\'ex\par
    \DMtext \'ex\'ex\'ex\'ex\'ex\'ex\'ex\'ex\par
```

`\DMtext` provides extra kerning; this is possible since `\'e` yields a true letter in the DM fonts and this letter appears in kerning tables of dmr10 etc; but not in those of cmr10 etc.

TEX user: I am pleased enough to have these fine adjustments which dmr10, etc., offer. But I don't want to have pay a heavy price. In particular I do not want to type `\'e`; that is a big pain. For quite some time a macro file called `option_keys` has let me instead type the regular Macintosh e-acute character (call it `Mac-e-acute`.
The Lion: In just a moment, you will be able to type either way! The service Caesar *alone* provides is to assure that *for the output* the best form of `e-acute` the font has available is used, and it makes that `e-acute` behave as best one can. All *automatically*.

5. To be able to directly type Mac 8-bit characters put `\input Active8MacCM.tex` at the head of your typescript, following `\input caesarcm`.

TEX user: But isn't `\input option_keys` recommended instead by Blue Sky Research.
The Lion: The `option_keys` file of the current Textures distribution (maybe the name is slightly different) is a valid alternative (in the presence of CaesarCM) provided you are willing and able to avoid the TEX primitives `\lowercase` and `\uppercase`, which misbehave outrageously under `option_keys`. `Active8MacCM.tex` does not really maintain good behaviour of the TEX primitives `\lowercase` and `\uppercase`, but it provides good replacement macros instead: `\Lowercase` or `\eLowercase` and `\Uppercase` or `\eUppercase`.
TEX user: But please, even if `Active8MacCM.tex` does better than `option_keys` for alphabetical material, I would like to keep the performance of `option_keys` for non-alphabetical stuff.
The Lion: `\input option_keys` followed by (the order matters!) `\input Active8MacCM.tex` should do the trick.

6. To enjoy (say) French hyphenation, rename the French 8-bit hyphenation file (say `fr8hyph.ck` — see references) as hyphen.tex and then recompile plain.tex and when TEX gives you its * prompt, type `\dump` and `return` to get (say) fPlain format. This requires initex for generic implementations. For Textures (version 1.6 or later, please), one uses the format 'Virtex'.

TEX user: I have never used virtex or initex or any of that jazz. Maybe this is where I get `\dump`'ed!
The Lion: Wait a minute. This is where, in reality, the fattest dividend comes: hyphenation for good linebreaking in just about any European language. Learning to understand how TEX works, will pay enormous dividends in the long run; indeed TEX is truly an open system whose secrets can be had for the asking. But OK, so many things get mixed up in the matter of format building that we all need help. Check your local documentation on initex/virtex. Some day there will be powerful assistants available for format building; for a prototype (concentrating on French and English) see 'format-dumper-cm'.

7. To see why a virtual font deserves the epithet 'virtual', activate the 'Save as. . . ' menu choice while the font table's preview window is active, and save in Illustrator88 format. The result is a PostScript file that Adobe Illustrator can open directly. If you peek inside this file with a text editor after changing its type to TEXT you will find that the font dmr10 is *absent*; in its stead you find cmr10 listed in the font roster. If you are fortunate enough to have a recent version of Adobe Illustrator (issued after the appearance of ATM), you will see something much more dramatic by simply opening the Illustrator file. When you zoom in on the letter `eacute`, its composite nature becomes visible. It is made up of two text records both from the font cmr10: one for the `e` and one for the acute accent. Each can be independently edited within Illustrator. Blue Sky has thus concealed in the Textures' 'Save as. . . ' option a 'dvi-to-postscript' converter of a particularly useful sort: the output can be reprocessed! Professionals prize this feature for interactively inserting labels composed by TEX into figures produced by Illustrator.

TEX user: I have glimpsed tempting features but I am worried about the effort required to put the whole act together in a form suitable for everyday use with Textures.

The Lion: For that, a different sort of tutorial is needed. Come back tomorrow.

And the Lion stretched out to take a nap in the shade of the hyphenation trie.

6 Tutorial 5: Tutorial on setting up DM fonts for Textures 1.6.X

The aim of this tutorial is to discuss every-day use of the virtual DM fonts to improve on the hyphenation and kerning offered by the CM/PS fonts distributed with Textures. The main task is to set up a bilingual format under Textures, letting one take advantage of the accented characters in the DM fonts. The term 'bilingual' should initially be read here as 'French and English'. With the help of the Babel package, it will be possible progressively to deal with any small collection of European languages; it is a matter of ironing out isolated difficulties — and that has already been done fairly thoroughly for French and English.

1. Use the format-dumper-cm directory (see 'References') to compile one of the basic formats Plain, LATEX, *AMS*TEXor *LAMS*TEX in bilingual form. To start the process, it suffices to launch the file 'initex.me' under Textures's version of initex, which has the form of a TEX format called 'virtex'. (This initex presentation is non-standard but *very* convenient.) As accent administration package, choose CaesarCM rather than HyAccCM. Choose CM fonts if a choice is offered; this will not prevent the use of DM fonts. Other choices are up to you.
   Name the resulting format and place it in the folder (=directory) `TeX Formats`. When you want to enjoy DM fonts, remember to select this format.
2. To permit Macintosh 8-bit typing of accented characters optionally add Active8MacCM.tex to the above format. Do this by compiling the following tiny file with the above format:

   ```
   %\input option.keys %% optional
   \input Active8MacCM.tex
   \dump
   ```

   and have the new format replace the above format.
3. Place the suitcase of Textures DM fonts into your TEX Fonts folder. To prevent confusion and duplications, you should have only one such suitcase for DM fonts.
4. Load the macro file `dmview.sty` in the usual way.
   This file contains slightly modified definitions of the macro commands `\DMtext` and `\CMtext` discussed in the last tutorial. Recall that the command `\DMtext` switches in the basic DM text fonts hopefully in a way making them replace the corresponding CM fonts. `\CMtext` switches back.
   There is additionally a convenient and powerful command `\DMpar`, roughly equivalent to `\DMtext` now, followed by `\CMtext` at the next end of paragraph. Whereas these two fail if the fonts are changed by non-standard commands like `\smallroman`, `\DMpar` deals optimally with whichever font is current. Hence `\DMpar` should be used almost exclusively.
   The accompanying file `dmview.tst` will let you test these macros with the format you have compiled.

TEX user: Why not switch to DM fonts from the outset and never return?

The Lion: Some day that will surely be the thing to do; indeed DM fonts would then be preloaded in place of CM fonts. At present however, it is a simple problem of preview speed that argues for using CM fonts most of the time. In February 1994, the virtual DM fonts consume *twice* the time for previewing; and that is a big time sacrifice that few users are happy to make all the time.

TEX user: How unfortunate. I use a Macintosh SI that requires about 2 seconds to preview an A4 page. I would like to see the preview move faster not slower! I recall that Adobe Type 1 fonts such as CM/PS were a joy from the day I got them because previewing remained just as fast as for non-scalable bitmapped fonts. Admittedly, there is a time penalty of few seconds when the previewing begins, but thereafter nothing noticeable. Are you suggesting I can use the DM fonts effectively while avoiding time wastage?

The Lion: Indeed you can, and `dmview.sty` is there to assist you. To begin with, so long as you print rather than preview, you can use the DM fonts as much as you like without paying a time penalty. This is because the printer is then the bottleneck, not the virtual fonts.
Where previewing is concerned, a flexible strategy is required to avoid losing half your preview speed. Basically, you use CM fonts until the preview reveals inadequate hyphenation and then insert `\DMpar` at the beginning of any paragraph needing better hyphenation. Since the use of DM fonts is thus restricted to the few paragraphs with an added `\DMpar`, the slowing of preview should be slight, typically 5% to 10%.

TEX user: Inserting `\DMpar` may not be too difficult in itself; but no one will be happy to learn a new typesetting procedure.

The Lion: Ah, but cleaning up bad linebreaks is is a quite inevitable part of typesetting! The bad breaks entailed in starting out with CM/PS rather than DM will normally be a modest fraction of the total — from mathematics, from accidents of page design, from neologisms, and (surprisingly important) from words spelled with an explicit hyphen. Better learn to concentrate this inevitable job at a single late stage of typesetting, say by initially using plenty of `\emergencystretch`. Then do the job efficiently, say with instant preview (via LinoTEX) of difficulty paragraphs. There is more on this in the last tutorial.

TEX user: I will have to see what works best. Long ago, I used preview very little; so I know I can get the best possible results without preview, provided, in proofing, I splurge and use a good deal of extra paper. Since the coming of the CM/PS fonts and ATM, I always insist on previewing even when I have a printer beside me. In previewing, is a big time disadvantage of virtual fonts inevitable? I mean for users who do not resort to tricks like `\DMpar`.

The Lion: I should hope not; the future of TEX's font system depends on it! One TEX previewer has cut the time disadvantage of virtual fonts to under 20 percent.

TEX user: I would bet on Textures taking up the challenge. Textures has the best record for speed among TEX implementations on the Macintosh, and speed is one of the best features of Textures.

The Lion: With substantial parts of Textures in assembly language, Textures has a headstart in speed analysis. But these things take time, even when there is a good idea available such as one sees in OzTEX 1.6x. Better not to hold your breath while waiting.

TEX user: Overall, this DM setup sounds neat, I grant you, at least if I manage to learn the tricks you mention. But I cannot help thinking how much simpler it would be to use new extended Macintosh encoded CM/PS fonts that mesh perfectly with the recently distributed Macintosh encoded Adobe Times, etc. Thanks anyhow.

The Lion: Wishful thinking! No Macintosh encoded extension of the CM/PS fonts has been announced as of February 1994. Nor acknowledged as a project. Further, use of Macintosh encoding (really Textures' own idiosyncratic version of Mac encoding) requires a good deal of special parametering of existing TEX software (CaesarCM, NFSS, all hyphenation packages, etc.). As things stand, you the TEX user will have to put your shoulder to the wheel to make Textures work smoothly with the Macintosh encoding.

TEX user: OK, but to be perfectly frank, I am still uneasy about using a system that is out of line with Times, Helvetica, etc. as currently distributed with Textures.

The Lion: Only a demanding user will notice that anything whatever is out of line! That is because CaesarCM already does a remarkably good job using just the segment 0–127 where the Macintosh and CM encodings essentially agree. Better, it is possible to make the 8-bit Type 1 fonts distributed by Adobe mesh perfectly with the DM fonts, indeed without any sacrifice of preview speed; see the 'xAdobeTextures' font posting on the CTAN servers.

7 Tutorial 6 Maintaining preview speed with Textures' virtual fonts

The following indicates in more detail how to start with CM/PS fonts and land up with irreproachable linebreaking and kerning using DM fonts while nevertheless avoiding substantial slowing of Textures' preview (version 1.6.x).

There are a number of settings that will permit you to delay the cleanup of line breaking problems (involving introduction of DM fonts) until the article's content is complete and stable:

```
\emergencystretch=25pt %% Plain default is 0pt
    %% suppresses underfull line complaints
\tolerance=10000 %% default is 100
\overfullrule=0pt %% default is 10pt
    %% makes line overruns less visible
\hfuzz=\hsize %% default is 0pt
    %% suppresses overfull line complaints
```

The comments here are mere hints about complex behaviour; see the TEXbook for an accurate account. The complexity can however be handled empirically (athletically!) by varying the parameters.

TEX user: How does this square with the uncompromising quality TEX boasts of?

The Lion: Everything in its due time! Demanding perfection at an early stage can be counterproductive in many typesetting environments.

When the time for cleanup of linebreaking comes, return toward the default values. Perhaps not all the way if the lines are short. And perhaps by stages since many complaints in the log confuse the mind and slow TEX down.

Experience suggests that, in French for example, most of your line-breaking problems will have nothing to do with the failings of CM/PS fonts as administered by CaesarCM. To scan efficiently for these problems, the maximum Textures preview speed is a blessing not to be abandoned lightly. Further, use of some sort of instant preview of modified paragraphs is highly desirable because several successive corrections are often required. This is provided by Textures' 'Lightning' mechanism if the typescript has just a few pages. But in general one should seriously consider using LinoTeX (see 'References' below) to get recomposition of single paragraphs in just a couple of seconds (LinoTEX serves Textures and OzTEX.).

Every five pages or so, you can expect a line breaking problem of the sort that will be fixed by using DM fonts rather than CM/PS. Not more often because CaesarCM incorporated Desarmenien's device for allowing hyphenation far from accents with CM fonts. Recall that one inserts the command `\DMpar` at the beginning of each troublesome paragraph to bring DM fonts unto play for just that paragraph. If your font system is exotic, you may also have to insert `\DMpar` after font switches within the paragraph. In this way you should be able to get linebreaking as good as you please at an average cost of something like a 5% or 10% slowing of the preview.

TEX user: Instead, why not simply hand hyphenate the the word causing trouble, using `\-`. That is what I have been doing for years!

The Lion: Not too bad a solution if you really know the rules of hyphenation of the language. Remember though that you should then always take the time to provide the complete hyphenation; otherwise you risk causing bad breaks later on. Indeed a word containing `\-` is not further hyphenated by TEX. Popping in `\DMpar` is a solution unrivaled for simplicity. Also it produces statistically better results. Indeed many new line break points become available throughout the paragraph; and TEX can often use these to overcome the difficulty without breaking the word you would guess!

After linebreaking has been perfected, comes the time to perfect pagebreaking — and not before since the former significantly influences the latter. This pagebreaking again calls for the fast previewing you hopefully will have preserved.

Perfectionists will want to switch globally to DM fonts before final printing — not so much to further improve linebreaking as to assure perfect kerning between an accented character and an immediately preceding character. Such kerning is not available with the unextended CM fonts. If high resolution is to be used for final printing, such a switch perhaps justifies the extra time required to recheck pagebreaks.

With Textures there is a convincing trick that lets one see many small differences that the improved kerning of DM fonts will bring. Begin with a chapter with many accents prepared as above; put `\DMtext` in the header and recomposes; but immediately upon launching composition, click on the preview window of page 2 (say) of the first version which will still be on view; shortly, the the change to the DM font version will be clearly visible; more likely than not there will be no change of linebreaks and pagebreaks; but you are almost sure to see slight lateral motion of many characters in response to new kerning around accented characters.

TEX user: So far so good. But I know from experience there is bad trouble ahead, at least in the case of a big job like a book. Final checking of pagebreaks and even linebreaks is absolutely obligatory after switching fonts; otherwise the insistance on typographic quality may backfire badly. But the Textures preview is now twice as slow, just when one wishes it were twice as fast. Reverting to printed copy at this point is a dreadful waste of paper, and not particularly efficient at that. The time loss here is not worrysome for a small job because it is moderate compared with other delays, notably various startup times. But for a big job I wonder whether I would be willing to attempt the full switch to DM fonts!

The Lion: A lion's pride demands uncompromising quality. Your scenario for big jobs seems to call for a *deus ex machina*!

The idea is to pass the Textures .dvi file(s) through P. Breitenlohner's DVIcopy utility; that will replace each virtual DM character by the corresponding array of non-virtual CM characters. The resulting .dvi file is without virtual fonts and will preview at top speed in Textures (Textures version 1.6.4 and later, please; earlier versions have a bug from faulty optimization causing displacement of many accents to the right.). OzTEX, in versions 1.6x, includes in its driver a fast and reliable version of this utility, serving as the basis of OzTEX's use of virtual fonts. It can conveniently be conscripted for the task, provided generic DM virtual fonts have been installed for OzTEX. Thus the *deus ex machina* for big jobs — is OzTEX!

This detour via OzTEX brings an extra benefit: the resulting CM based `dvi` files are highly portable because the CM fonts are Knuth's standard fonts for TEX. Users are fortunate that OzTEX and Textures have adopted different approaches to virtual fonts!

Perhaps some day Textures too will have the 'DVIcopy' feature; that would make life simpler for Textures users. In the interim this tutorial should help users get the best possible results from DM fonts — at the moderate price of being well informed!

8 References

The software described in this article, Textures excepted, has its current master posting on `matups.matups.fr` and most of it is mirrored on the CTAN anonymous ftp servers. The current locations on `matups.matups.fr` are `/pub/TeX/` followed by:

```
Fonts.dir
      FontTable256*,  xAdobe-Textures*, K-PS.dir
Fonts.dir/DM.dir
      dm-table.txt, dmview*
Fonts.dir/DM.dir/TexturesDMs.dir
      DM-VFs*, TexturesDM-VFs.readme
format-dumper-cm.dir
      caesarcm.tex, fr8hyph.ck, initexme.tex
LinoTeX.dir
oztex.dir (version 1.6) shareware at $30
TypingTeX.dir
      Active8MacCM.tex
```

For Textures utilities such as EdMetrics consult the anonymous ftp site `ftp.bluesky.com`. There is also an email help address `help@bluesky.com`.

V TeX and SGML — Friend or Foe?

Jonathan Fine
J.Fine@uk.ac.cam.pmms

At the last UKTUG committee meeting there was an interesting discussion about holding a meeting (London, November this year perhaps) on TeX and SGML. It became clear that for such a meeting to be successful, particularly for developing and promoting TeX as a typesetting system, that the purpose, focus, agenda, speakers and audience were matters that required careful thought and further discussion. What follows are some personal observations and opinions on the subject, with which the rest of the committee may or may not agree. It is my intention to open communication and begin a debate that will continue through to the proposed meeting this winter and beyond. My primary sources are *The TeXbook*, and *The SGML Handbook* (Charles Goldfarb, OUP 1990), which will be cited as [T] and [S] respectively.

First, some words about standards. An old joke has someone saying "Yes, we believe in standards. That's why we have so many of them." The joke, of course, is that standards should create or make manifest uniformity amongst similar objects. Eli Phalet gave an early demonstration of the effectiveness of standards to President Lincoln, early in the US Civil War. He dismantled several rifles, mixed the parts up in a heap, and the reassembled the rifles, thereby demonstrating the interchangability of the parts. (This won him a large Union munitions contract.) Because the parts had been manufactured to carefully specified tolerances, this could be done. Then it was surprising. Now, it is perhaps surprising that it was once surprising. We take it for granted. Another meaning for 'standard' is as a flag which leads an army into battle. Such standards are economic realities in the commercial world.

The word 'document' is overworked. Instead, I will use the word 'compuscript' (or script for short), to refer to a structured file containing text and tags or processing commands. It is convenient to think of a script as being an ASCII file meeting (formal or informal) syntax conditions. Thus presented, many files are scripts. TeX and LaTeX files satisfy an informal syntax. The same is true of macro files. Other examples are the content of a database, expressed in any one of a number of formats, program source files for any of the many programming languages, and document files for the various word processors and other typesetting systems. The ISO standard defines a document to be "A collection of information that is processed as a unit. A document is classified as being of a particular document type." [S, p124,263]

This may seem rather pedestrian and pedantic, but we are not yet able to repeat for scripts Remington's rifle trick, which is of course based on boring and pedantic precise specifications for the parts. Incidentally, if you look up *Boring* in the Yellow Pages, it will say *See Civil Engineers*.

The same compuscript may be processed in several ways. It may be edited, typeset, formatted for online display, compiled (if a program source file), or have its spelling and grammar checked. Portions may be extracted to form a secondary compuscript, such as an abstracts or citation journal. We can now see the complementary rôles of SGML and TeX. The first is a standard for the specification of compuscripts. It is [S, p7–8] "based on two novel postulates

a) Markup should describe a document's structure and other attributes rather than specify processing to be performed [. . .]
b) Markup should be rigorous so that the techniques available for processing rigorously-defined objects like programs and databases can be used for processing documents as well."

while TeX is "a new typesetting system intended for the creation of beautiful books—and especially for books that contain a lot of mathematics" [T, page v]. Thus, SGML is a specification language for compuscripts while TeX is a typesetting system which will process suitable compuscripts.

So far as I can tell, both TeX and SGML are sound in their basic design. Given this—although some may disagree—one would expect them to work well together, like nuts and bolts. However, they do not, and it is worth understanding why and how. Here I must admit to having a trumpet to blow. It is my belief that a TeX format can be written, that will parse and typeset suitable SGML compuscripts, and that such a format is the way to go. The following remarks are focussed on the existing TeX and LaTeX formats.

TeX has no inbuilt concept of markup or of parsing. This is probably as it should be, and I suggest that the reader reflect on why. My opinion is that such is—in terms of Knuth's goal of creating beautiful books—a bell or whistle. A diversion. For similar reasons, I believe, Knuth saw no need to write a text file editor. He did however produce the WEB programming tools. He did supply TeX and a couple of thousand lines of macros. Since then TeX macro packages have mixed parsing in with processing in a manner which prohibits rigorous markup—a hallmark of SGML. One symptom of this is the recurrent problems of verbatim text within a macro argument, such as a section title.

Because users can define new commands, the syntax of a TEX compuscript is always subject to change. It may be harmless to write

```
\def\beq{\begin{equation}}
\def\eeq{\end{equation}}
```

in the preamble to a LATEX compuscript, but

```
\beq ax^2 + bxy + cy^2 \eeq
```

will now cause a spell checker programmed to skip mathematics to trip up. Moreover, to set up such a checker to find the error in

```
\begin{equation}
e = mc^2 \qquad\hbox{Eintsien}
\end{equation}
```

will not be easy.

A more substantial problem is the special and contingent typesetting instructions, that are required to achieve quality typesetting. The simplest examples are the space adjustments `\>` and so forth used with mathematics. The breaking and spacing of long equations and formulae, when setting to a narrow measure, presents more difficulties, if one is to typeset from a compuscript satisfying a rigorous syntax. The same applies to tables. Typically, one might expect a skilled compositor (either human or robotic) to 'annotate' the author's compuscript for, say, a scholarly journal with commands to control or adjust page breaks, the size and placement of floating items—in a word, page make up. SGML recognizes [S, p139,277] that one sometimes needs "processing instructions," which are "markup consisting of system specific data that controls how a document is to be processed." Here, the system might be TEX-based typesetting, or typesetting to a particular design, or some other application. "As war is to diplomacy," writes Goldfarb [S, p139], so this is "the last resort of descriptive markup."

The key to success for SGML is that it provides standards for compuscripts, or more exactly provides tools for the expression of such standards. This allows diverse programs to process the same compuscript in various ways, for different purposes. Yuri Rubinsky, in his preface [S, page x] wrote

> Over the next five years, computer users will be invited to anbandon their worst habits: They will no longer have to work at every computer task as if it had no need to share data with all their other computer tasks; they will not have to act as if the computer is simply a complicated, slightly-more-lively replacement for paper; [. . .]; not have to appease software programs that seem to be at war with one another.

but perhaps he is too optimistic—he was writing in October 1990.

There appear to be two main situations where TEX can contribute to SGML based document processing. The first is the high quality typesetting of SGML compuscripts, such as the content of a database. The second is more subtle. The tagging process adds information to the compuscript, and thereby makes it more valuable. For example, in this document the names of our two author, Knuth and Goldfarb, are set in the main body font, and so require no additional markup. But for a hypertext retrieval engine, we will want these names linked to an index of persons. Mechanical processes may help, but because many people share the same family name, a certain amount of author assistance is required, particularly for the more common names, family names that are also place names, and so forth.

This is only one example of how the author is uniquely qualified to provide data tagging, as we may call it. Employees can be told to tag data, but this strategy is unlikely to work for the authors of scholarly publications. Instead they must be equipped with tools and incentives. In particular, a document processing system which returns benefits (such as copious indices and cross-references) to the author as a consequence of data tagging will provide an incentive perhaps stronger than coercion. TEX is freely and widely available. It deserves to be part of such a system.

VI Tips for installing LaTeX 2$_\varepsilon$ for OzTeX

Peter Abbott

Having collected the files from the `macros/latex2e/base` directory in the CTAN archives I decided to install it on a Mac IIfx. This is what occured and may assist others in preventing wasted time. I used OzTeX version 1.6 for this exercise.

1. Edit the default config to include a LaTeX 2$_\varepsilon$ directory in the `TeX-inputs` path and put all the LaTeX 2$_\varepsilon$ files in there. I created a folder `LaTeX2e` and added it to `TeX-inputs`.
2. Edit `hash-size` from 2500 to 4000 and change `hash_prime` accordingly.
3. Change the two circle tfm files to `lcircle tfm` (Andrew will correct this one in version 1.7)
4. Add `LaTeX-docs` to `TeX-inputs`.
5. Run initex on `unpack2e.ins` and then on `latex2e.ltx`
6. Go and read a book or do something useful. The elapsed time on my IIfx was 11871.7secs, yes over 3 hours!

VII Malcolm's Gleanings

Malcolm Clark
m.clark@warwick.ac.uk

0.1 Rest in peace

For a number of years, the 'maverick' (so described by Adrian 'no relation' Clark) occasional publication *TEXline* made its infrequent appearance. The last issue, number 14, came out in February 1992. As the erstwhile editor I now reluctantly admit that it has ceased publication. I had hoped that it might survive to a ripe old age (like issue 20), but a variety of reasons compel me to abandon it. *TEXline* was always a free publication, so at least I don't feel that there is a vast subscription paying audience out there which will feel aggrieved at not receiving the next edition. After all, part of the deal was supposed to be that if you wrote for it you were then on the subscription list. To the list of contributors, sorry, your subscriptions have lapsed. To the rest of you, well, you know you get what you pay for.

There are now plenty of other ways to disseminate information about TEX and family. Estimable publications like *Baskerville*, *Lettres GUTenberg* (no, not *Lettres français*), TTN and the Dutch group's MAPS more than adequately fill any holes that might have been left by *TEXline*, and have the great virtue of being related to various groups. I hope that this encourages people to belong to those groups.

Of course there are other ways to spread the word. The electronic distribution of information is an obvious alternative. Why, for example, don't we have all these journals and newsletters (or annals) up on WWW (World Wide Web)? TTN is already available in electronic form, but I think I can see the day when it is available in (say) POSTSCRIPT form so that I can just call it up on the screen in front of me. The problem here lies in one of the recurrent problems of material available on the Internet – who would bother subscribing to TUG or the UK group if one of the key benefits of membership, the publications, was already available electronically? I still prefer bits of paper (I can easily take them home, read them in the bath, make notes on them – and the batteries rarely run flat), but paper may be a luxury we can't really afford. I'm moderately hopeful that gadgets like the Sony DiscMan, together with emerging technologies like Adobe's Acrobat may make portable electronic books a realistic alternative.

0.2 Carousel?

A couple of years ago, Adobe Systems started to introduce their 'Multiple Masters' technology. At the time it was not wholly clear what their agenda really was. Multiple Masters allow you to retain the 'look and feel' of a printed (or screen) document when you do not have access to the 'correct' fonts. Imagine you had been sent *Baskerville* electronically, but without the embedded Baskerville fonts which are part of its corporate identity. Exactly what would happen when you tried to view it or print it depends on lots of things: at the worst you might not be able to view it at all; alternatively you might have some substitution which retained character widths, or character positions. The effect is that you would end up with something which was frankly unpleasant. The content would still be the *Baskerville* you have come to love, but its appearance would be well below the standard which we have come to expect.

Adobe's solution was to develop a scheme where two font families were introduced which would attempt to emulate the characteristics of the 'missing' font(s). All this means is that they would have the same character widths and heights (slants, weights or whatever) so that the colour of the 'page' would be similar. Of course, there was no intention that this would be an exact reproduction – just that it would preserve characteristics like line breaks and the other physical attributes of the page. The two base fonts introduced were Minion and Myriad (now renamed Adobe Sans and Adobe Serif, I think). These were placed in the public domain.

Originally Adobe's marketing people (probably to put us off the true scent) were suggesting that this was the start of a wide range of Multiple Master fonts. In passing, this is where Yannis Haralambous in the last edition of *Baskerville* was misled. He states that – 'These (MM) are extremely complex and memory consuming, but still much poorer than METAFONT created fonts': this may be true, but it is totally irrelevant and misses the point entirely; this is not what was going on at all. It was all a prelude to Acrobat, or, as it was originally code-named, Carousel. Multiple Masters were simply an enabling technology which had to be in place before Acrobat could be released. I had originally supposed that part of the reason behind MM was font licensing. The copyright of fonts depends on the country in which you sit. See, for example, Chuck Bigelow's article in *TUGboat* ('Notes on typeface protection', vol.7(3), 1986).

It is by no means certain that you have the right to transmit the details of any arbitrary font electronically: by doing so you may be in breach of copyright (there is no use pleading that Bodoni has been dead for more than 50 years, or Baskerville for that matter – they never held the copyright anyway). Another reason not to do this is that encoding or embedding the fonts takes up an enormous amount of space. If you are transmitting a book sized document, the extra for fonts is no big issue: if you are transmitting something relatively short, it could be the biggest item. There are therefore legal and practical reasons for this route.

Once we have an Acrobat document, it can be transmitted electronically without difficulty, and can be read by anyone with an Acrobat reader. Such readers seem destined to be made available with every machine, virtually as part of the operating system. It no longer matters what software was used to prepare the original – you will still be able to see what the author intended, without having that software yourself. There are other little advantages: the one I like most is the post-it notes that you can attach to the document. For example, this could mean that as an author I send my Acrobat document (which of course started life as TEX or LATEX) to my publisher, whose editors would annotate via the post-it notes. I then revise the returned document. It all starts to sound very plausible. And yes, it does work.

Another pleasing feature is the ability to search the Acrobat documents. This works reasonably well, although, as you might expect, ligatures (and I dare say, diacriticals) don't yet work properly. Of course, years and years ago you could search a dvi file for text strings with ArborText's previewer, with just the same sorts of restrictions. I think it is always good when the innovations introduced through TEX and its associated products becomes incorporated into mainstream technologies like this. It shows how TEX leads the way.

0.3 Knuth, the video

Donald Knuth is now available on video: University Video Communications, based at Stanford, now have a tape of Knuth's 'Computer musings' in which he discusses the One-Way Associative Law. Sounds just the thing for a stocking filler next Christmas.

0.4 Sheep stealing in Barnet

From time to time a chestnut re-appears. A half remembered quotation from Frederic Goudy is trotted out: ''anyone who would letterspace lower case would steal sheep''. There are many who concur with the spirit of this statement, but unfortunately it isn't what Fred said. Matters have been confused recently by the publication of a book by Erik Spiekermann & E M Ginger entitled 'Stop Stealing Sheep', which does contain the erroneous quotation. Fortunately they also print the correct version. What Goudy actually said was ''anyone who would letterspace black letter would steal sheep''. Where does that leave us? In *Baskerville* 3(2) I noted that Eric Gill suggests letterspacing lower case as a substitute for italics. While I know of no rumours that Gill actually *stole* sheep, that might have been the least of their worries.

0.5 Public domain does not mean user unfriendly

In another surprising outburst (one by-product of the remarkable mud-wrestling competition held with Berthold Horn, judged by most observers to result in a draw), Yannis Haralambous suggests in a footnote that 'public domain software is never as user-friendly as commercial': I beg to differ. I have the doubtful pleasure of a Windows machine on my desk. One of its few really excellent pieces of software is Pegasus, a Windows-based electronic mailer. It is simple, straightforward, and almost a pleasure to use. A true pleasure to use is Eudora, another mailer, on the Macintosh. Both of these are public domain, and I challenge anyone to carp significantly about their user interface.

In fact, I'm rather surprised by Yannis' assertion that 'important innovations (in TEX etc) have always appeared first in public domain software': I have only to think back to the release of TEX on the pc: this was not in the public domain but was the result of significant and independent work by Lance Carnes and David Fuchs – maybe it was not an important innovation, but to me, freeing TEX from the restrictions of academic mainframe computers and releasing it to the personal computer level seems a quantum leap forward. Even Yannis' beloved virtual fonts were released first by ArborText, many years before they became commonplace. There are countless other examples around – think of Blue Sky's innovative Lightning Textures, or Michael Vulis' VTEX. This continued denigration of the importance which commercial vendors have had in the adoption and spread of TEX smacks of a re-writing of easily verifiable history. They were there: they supported us all for many years until public domain versions of the program finally became robust and reliable. I believe they still have a place in the development of TEX and TEX tools. I just hope they don't get so frustrated by the lack of understanding and respect they receive that they abandon the game entirely.

0.6 Hints & Tips

At the first meeting that this group held at RHBNC one of the 'events' was a LATEX tables workshop. One of the problems that always comes up with tables is the incredible contortions that you have to go through to make the floating tables appear just where it is that you want. It would be naive to assume that this is going to be one of the problems that will disappear with LaTeX3. In the first place it is extremely difficult to come up with a sensible and coherent set of rules which describe the variety of conditions and exceptions to figure and table placement; in the second place, the LaTeX defaults are perhaps not the best chosen. At least we can do something about that, although there are rather too many parameters to be able to come up with a definitive optimal set of values. However, at the RHBNC meeting, Geeti Granger of John Wiley & Sons provided a set of values which she suggests had been used with more success at John Wiley than the defaults. She suggested the values in the following table:

parameter	default	suggested
topnumber	2	2
bottomnumber	1	2
totalnumber	3	4 or 2
dbltopnumber	2	2
`\topfraction`	0.7	0.9
`\bottomfraction`	0.3	0.5
`\textfraction`	0.2	0.1
`\floatpagefraction`	0.5	0.8
`\dbltopfraction`	0.7	0.9
`\dblfloatpagefraction`	0.5	0.8

VIII Backslash—About `\def` and `\let`

Jonathan Fine
J.Fine@uk.ac.cam.pmms

Beginners to programming TEX sometimes wonder as to the difference between `\let` and `\def`. More exactly, how do the commands

```
\def\cs{a}
\let\cs=a
```

differ from each other.

To understand this, we must know that the TEX operates by processing *tokens*. It is convenient here to divide tokens into three types:

– character token
– active character
– control symbol or sequence

of which the last two have the special property, that they may be assigned a meaning (and so are called *assignable* or *control* tokens), while character tokens (excluding the active characters of course) have a fixed and unchanging meaning. These then are the tokens. Assignable tokens can be given a meaning, while character tokens have a fixed meaning.

The other side of the coin is of course that tokens have meanings. The possible meanings for a token are more diverse. The list

– a letter
– an 'other' character
– a 'something' character, where 'something' is *begin group*, *end group*, *math shift*, *alignment tab*, *superscript*, *subscript* or *space*. The above, together with active characters (which are assignable, and have no intrinsic meaning of their own) give all the character tokens which can appear in the stomach of TEX. For more details see [37] (this means page 37 of *The TEXbook*), and [39], Exercise 7.3.
– primitive commands (starred in the index to the *TEXbook*)

gives some of the possibilities. There are others. The commands `\font`, `\chardef`, `\mathchardef`, `\countdef` and so forth assign a meaning to an (assignable) token, which does not appear in the above list. Finally, there are macros. Macros have a parameter text and a replacement text. Typically, they are created by using the `\def`, `\gdef`, `\edef` and `\xdef` commands. The difference between the various `\def` commands, which all produce a macro, is explained on pages [206] and [215–6]. Note that the macro does not carry as part of its meaning the type of command which was used to create it. Finally, so that every token has a meaning, there is a final possibility, which is that the meaning is `undefined`.

It is now possible to explain, at least in theoretical terms, the difference between `\def` and `\let`. The execution of a `\def` command will assign a macro meaning to a control sequence. Thus,

```
\def\cs{a}
```

will produce a zero parameter macro `\cs`, whose parameter text is empty and whose replacement text is the single letter `a`. This can be seen by now applying the `\show` command.

```
*\show\cs
> \cs=macro:
->a.
```

The `\let` command, however, is used to transfer (or more exactly make a copy or reference to) an existing meaning. Thus,

```
\let\cs=a
```

will produce a control sequence whose meaning is

```
*\show\cs
> \cs=the letter a.
```

the letter `a`. *This is not the same as the macro whose replacement text is the letter* `a`.

So how do `\def` and `\let` differ? Well, `\def` will always create a macro, while `\let` will transfer a meaning. If we have already performed

```
\def\aaa{aaaaaaaaaa}
```

then

```
\def\xyz{aaaaaaaaaa}
```

will create a new macro `\xyz` whose meaning happens to be the same as that of `\aaa`, while

```
\let\xyz\aaa
```

will give to `\xyz` the current meaning of `\aaa`, which happens to be

```
*\show\aaa
> \aaa=macro:
->aaaaaaaaaa.
```

but could have been otherwise. This use of `\let` does not change the meaning of `\aaa`. Here `\aaa` was a macro, but it could have had any meaning, perhaps even `undefined`.

If one finds oneself making definitions such as

```
\def\cs{\token}
```

where `\token` is a single control sequence or character token, perhaps one should instead be using

```
\let\cs\token
```

which will have the same effect, but more efficiently, in most contexts. However, there is a significant difference. When `\let` is used, `\cs` gets *the current meaning* of `\token` (which may not yet be defined). When `\def` is used, `\cs` is a macro which expands to `\token` and so it is the meaning of `\token` *at the time of the use of* `\cs` which is relevant. See [206–7] and especially Exercise 20.8, and also the definition of `\obeylines` [352]. This nuance does not apply when `\token` is given a fixed meaning, once and for all time.

Finally, the answers to the exercises in the last issue, and a new exercise.

Solution 1. *Use* `\aftergroup` *to export the token* `\xyz` *out of the group, and discuss merits*.

```
\beinggroup
  \aftergroup\typeshow
  \expandafter
  \aftergroup\csname xyz\endcsname
\endgroup
```

This code is slower and bulkier than use of multiple `\expandafter`s (as in the last issue) but will be quicker and more powerful if one wishes to assemble longer sequences of unusual tokens.

Solution 2. *Write a macro which tests as to whether (the meaning of) a token is expandable*. According to the recommended pages [212–215], macros and 'certain special primitives like `\number` and `\if`' are expanded. Reading on to the foot of [213] we find that `\noexpand` applied to a token has expansion 'the token itself; but that token is interpreted as if its meaning were '`\relax`' if it is a control sequence that would ordinarily be expanded by TeX's expansion rules.' This change of interpretation is purely temporary, and holds only during the execution or expansion of the next command. The meaning then reverts to the original value. (I have not found a reference for this in the *TeXbook*).

Here then is the solution. It will not do to compare `\noexpand #1` with `\relax`, for this fails when `#1` is `\relax`. Instead, we test to see if applying `\noexpand` to the parameter `#1` *causes its meaning to be changed*, for this is what `\noexpand` does to tokens whose meaning is expandable. (Note that `\noexpand` can change the meaning only of the instance of the token to which it applies, and then only for the next execution or expansion).

```
\long\def\isexpandable #1%
{%
  \immediate\write 16
  {%
    The token "\string #1" is
    \expandafter\ifx\noexpand #1#1%
    un\fi
    expandable
  }%
}
```