The mftinc package*

Scott Pakin
pakin@uiuc.edu

2001/05/15

Abstract

The MFT program pretty-prints METAFONT source code into a TeX file. The mftinc package facilitates incorporating such files into a LaTeX 2ε document. In addition, mftinc provides routines for improved comment formatting and for typesetting font tables.

1 Introduction

METAFONT [1] is Donald Knuth's system for creating fonts—and entire families of fonts—by describing the characters mathematically in a specialized programming language. As with any programming language, it is important for a programmer to document his code, to make it easier to extend and modify in the future. MFT is a stand-alone utility that makes METAFONT programs more readable by typesetting different language constructs (keywords, variables, etc.) in different fonts and styles. For example, the following is the font program for Computer Modern Roman's plus-sign character (taken from `punct.mf`):

```
cmchar "Plus sign";
beginarithchar("+"); pickup rule.nib;
x1=x2=good.x .5w; top y1=h+eps; .5[y1,y2]=math_axis;
lft x3=hround u-eps; x4=w-x3; y3=y4=math_axis;
draw z1--z2;  % stem
draw z3--z4;  % crossbar
labels(1,2,3,4); endchar;
```

and this is how MFT formats it:

cmchar `"Plus sign"`;
beginarithchar(`"+"`); **pickup** *rule.nib*;
$x_1 = x_2 = \mathit{good.x}\ .5w;\ \ \mathit{top}\, y_1 = h + \mathit{eps};\ \ .5[y_1, y_2] = \mathit{math\_axis};$
$\mathit{lft}\, x_3 = \mathrm{hround}\ u - \mathit{eps};\ \ x_4 = w - x_3;\ \ y_3 = y_4 = \mathit{math\_axis};$
draw z_1 `--` z_2; % stem
draw z_3 `--` z_4; % crossbar
labels$(1, 2, 3, 4)$; **endchar**;

---

*This file has version number v1.00, last revised 2001/05/15.

The formatted version draws attention to language features. It shows keywords in bold, variables in italics, subscripts as subscripts, and comments right-justified. The problem, though, is that MFT produces Plain TeX documents, which can't readily be included into a LaTeX 2ε document. What's the advantage of including formatted font programs in LaTeX? The answer is that it lets you take advantage of LaTeX's formatting and structuring capabilities to produce clear font documentation with comparatively little effort. Because a METAFONT program is like any other program, good documentation is important, as it makes it easier to extend and modify the font in the future. Using LaTeX, you can, for instance, put majuscules in one chapter, miniscules in another, and punctuation in a third; you can include graphics produced by METAFONT's smoke or proof modes to show what the resulting glyphs should look like; and you could add hyperlinks, a table of contents, a bibliography, font samples, and anything else that can clarify how the various character programs operate.

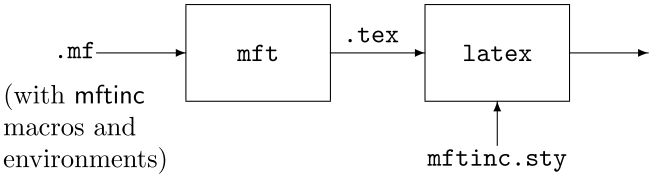
The mftinc packages's initial purpose was somewhat unambitious: simply include an MFT-produced `.tex` file within a LaTeX 2ε document. But it evolved from that to support the following additional features:

- Comments describing large, top-level blocks of code, such as the character programs themselves
- Stanza-level comments within a block of code
- Font tables, à la Knuth's `textfont.tex` utility

Figure 1 shows an example of how one might format Computer Modern Roman's plus-sign program, using mftinc's comment environments. Notice how mftinc makes the introductory paragraph stand out from the character program by placing it between a pair of horizontal rules and the final paragraph stand out from the surrounding code by prefixing each line with a percent sign (METAFONT's comment character).

2 Usage

I hope the previous section and Figure 1 piqued your interest in mftinc. We'll now look at how to use all of mftinc's features. Remember, the idea is that you use mftinc's commands and environments within a `.mf` file (generally on lines beginning with `%%`, as MFT passes such lines unmodified to the resulting `.tex` file). You then format the `.mf` file using MFT. And finally, you include the resulting `.tex` file into your main LaTeX document (which contains a `\usepackage{mftinc}` in its prologue to enable mftinc's features).



The following is the definition of the plus sign character ("+"). Admittedly, the glyph is so simple that it doesn't really need the depth of commentary that's I'm providing here. I mainly wanted to show how mftinc formats comments. Speaking of which, this is a block-level comment, created with mftinc's `explaincode` environment.

cmchar`"Plus sign"`;
beginarithchar(`"+"`);
% This is an ordinary MFT comment, entered with `%`. Notice how only the first line starts with a percent sign, and the text isn't even properly indented. Yuck!
pickup $rule_{nib}$;

% This is another ordinary MFT, `%`-prefixed comment. I had to
% break the lines manually and end each line with a `\]`. While
% that's okay for one-liners, it's an immense bother for longer
% comments (like this one).
$x_1 = x_2 = good.x\ .5w$;
$top\ y_1 = h + eps$;
$.5[y_1, y_2] = math\_axis$;
$lft\ x_3 =$ hround $u - eps$;
$x_4 = w - x_3$;
$y_3 = y_4 = math\_axis$;

% Ah, much better! This comment was entered within one of mftinc's
% `wrapcomment` environments. Notice how it's indented the correct
% amount, every line starts with a `%`, and the lines are fully justified—
% and none of this required any manual formatting. mftinc did all
% the work for us.
draw z_1 -- z_2; % stem
draw z_3 -- z_4; % crossbar
labels(1, 2, 3, 4);
endchar;

Figure 1: Computer Modern Roman's "+", formatted with mftinc

Note that mftinc's macros and environments do, in fact, work in the main LaTeX document. It's just that some of them aren't particularly interesting outside of a METAFONT file.

2.1 File inclusion

`\mftinput {⟨filename⟩}`

`\mftinput` `\mftinput` is used within the main LaTeX document to incorporate an MFT-produced `.tex` file. If a file extension is not supplied, `.tex` is assumed.

2.2 Improved comment handling

```
\begin{explaincode} [⟨options⟩]
⟨comment text⟩
\end{explaincode}
```

`explaincode` METAFONT programs define one character program for each glyph in the font. It's good style to start each of these—and other top-level blocks of code—with a comment describing the code and its particular nuances. The `explaincode` environment typesets such comments between horizontal rules, so that the comments are more easily distinguishable from the code they describe. `explaincode` also adds a little stretchable space above the first rule to separate the comment from whatever precedes it. For example, the following lines in a `.mf` file:

```
%% \begin{explaincode}
%%   This text is set within an \texttt{explaincode} environment.
%%   \texttt{explaincode} is intended to be used before a character
%%   program or other large block of code.
%% \end{explaincode}
```

will look like this when run through `latex`:

This text is set within an `explaincode` environment. `explaincode` is intended to be used before a character program or other large block of code.

The optional argument to `\begin{explaincode}` provides control over the thickness of the two rules. This is discussed in Section 2.4.

```
\begin{wrapcomment}
⟨comment text⟩
\end{wrapcomment}
```

`wrapcomment` The `wrapcomment` environment is used for comments that describe a stanza—

a logical chunk of code—within a character program or macro. The important features of comments that are typeset with `wrapcomment` are the following:

- They can be multiple lines long.
- They wrap text like any other piece of LaTeX code.
- Each line of output begins with a percent line.
- The comments use the same indentation as the block of METAFONT code they describe.

Here's an example of a `wrapcomment` that's indented within a METAFONT **for** loop and the way that mftinc tells LaTeX to format it:

```
for i = 0 upto length cpath:
  %% \begin{wrapcomment}
  %%   See how comments typeset within a \texttt{wrapcomment}
  %%   environment are indented?  They line up with the first
  %%   \verb+%%+ within the environment.  Just remember to use two
  %%    percent signs instead of one, or bad things will happen.
  %% \end{wrapcomment}
  draw z[i]--z.c;
endfor;
```

for $i = 0$ **upto** length *cpath*:
% See how comments typeset within a `wrapcomment` environment
% are indented? They line up with the first `%%` within the envi-
% ronment. Just remember to use two percent signs instead of
% one, or bad things will happen.
draw $z[i]$ -- z_c;
endfor;

`\mfcomment`

`\mfcomment` One advantage that MFT's `%` comments have over `%%` comments is that they format anything that's set between vertical bars as it if it were METAFONT code. For example, `|draw z1--z2|` is typeset as "**draw** z_1 -- z_2". The problem is that the `explaincode` and `wrapcomment` environments need to be typeset with `%%`, so their contents gets passed directly to LaTeX. To embed METAFONT code within `explaincode` or `wrapcomment`, one need only end the previous line with `\mfcomment` and put the METAFONT code on the next line, preceded by a `%`:

```
%% \begin{wrapcomment}
%%   The reason we set \mfcomment
%    |x4 = w - x3|
%%   below is to ensure that when we later \mfcomment
%    do a ``|draw x4{up}..x1..{down}x3|'', the character
```

```
%%   will have equal left and right sidebearings.
%% \end{wrapcomment}
```

% The reason we set $x_4 = w - x_3$ below is to ensure that when we % later do a “**draw** $x_4\{up\} \mathrel{..} x_1 \mathrel{..} \{down\}x_3$”, the character will have % equal left and right sidebearings.

2.3 Font tables

Most TEX distributions come with a program of Knuth's called `testfont.tex`, which can produce a variety of font samples. One of `testfont`'s more useful features is the ability to produce a table of all the characters in a given font:

```
% tex testfont
This is TeX, Version 3.14159 (Web2C 7.3.2)
(/usr/share/texmf/tex/plain/base/testfont.tex

Name of the font to test = cmr10.mf
Now type a test command (\help for help):)
*\table

*\bye
[1]
Output written on testfont.dvi (1 page, 5812 bytes).
Transcript written on testfont.log.
```

Table 1 depicts the table that this produces. Characters are numbered in both octal (*´000–´177*) and hexadecimal (″00–″7F). Empty rows—of which there aren't any in this example—are automatically omitted.

`\fonttable` The problem is that `testfont.tex` was designed to be used interactively and stand-alone. But wouldn't it be nice to be able to include a font table in the same document that contains the annotated font source code? With mftinc, you can do just that. mftinc includes a `\fonttable` command, based on the one in `testfont.tex`—in fact, much of `\fonttable`'s code was taken verbatim from `testfont.tex`—but extended to provide more features and to interact better with LATEX.

`\fonttable [`⟨*options*⟩`] {`⟨*font name*⟩`}`

The mandatory argument, ⟨*font name*⟩, is the name of the font to chart. Note that this must be the TEX, as opposed to LATEX 2$_\varepsilon$, font name. For example, to draw a font table of 11 pt. Computer Modern Typewriter Text, one would have to write “`\fonttable{cmtt10 at 11pt}`”, because there is no 11 pt. version of the font, only a 10 pt. version scaled up to 11 pt. The optional argument to `\fonttable`, ⟨*options*⟩, provides control over the width of the table and the range of characters included within it. This is discussed in Section 2.4.

Table 1: Complete `cmr10` character set

	´0	´1	´2	´3	´4	´5	´6	´7	
´00x	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	″0x
´01x	Φ	Ψ	Ω	ﬀ	ﬁ	ﬂ	ﬃ	ﬄ	
´02x	ı	ȷ	`	´	ˇ	˘	¯	˚	″1x
´03x	¸	ß	æ	œ	ø	Æ	Œ	Ø	
´04x	˗	!	”	#	$	%	&	’	″2x
´05x	(	)	*	+	,	-	.	/	
´06x	0	1	2	3	4	5	6	7	″3x
´07x	8	9	:	;	¡	=	¿	?	
´10x	@	A	B	C	D	E	F	G	″4x
´11x	H	I	J	K	L	M	N	O	
´12x	P	Q	R	S	T	U	V	W	″5x
´13x	X	Y	Z	[	“	]	ˆ	˙	
´14x	‘	a	b	c	d	e	f	g	″6x
´15x	h	i	j	k	l	m	n	o	
´16x	p	q	r	s	t	u	v	w	″7x
´17x	x	y	z	–	—	˝	˜	¨	
	″8	″9	″A	″B	″C	″D	″E	″F	

2.4 Options

The `explaincode` environment and the `\fonttable` macro each take an optional argument containing zero or more comma-separated, ⟨*key*⟩=⟨*value*⟩ pairs. These allow for finer control over `mftinc`'s behavior.

`toprule=`⟨*dimen*⟩
`bottomrule=`⟨*dimen*⟩

The horizontal rules drawn above and below an `explaincode` comment are normally 1 pt. thick. The `toprule` and `bottomrule` options enable you to change this default. For example:

```
%% \begin{explaincode}[toprule=3mm,bottomrule=5pt]
%%   The rule above this sentence is 3\,mm.\ thick, and the rule
%%   below this sentence is 5\,pt.\ thick.
%% \end{explaincode}
```

The rule above this sentence is 3 mm. thick, and the rule below this sentence is 5 pt. thick.

`tablewidth=`⟨*dimen*⟩

The tables drawn by `\fonttable` normally expand to fill the width of the text. The `tablewidth` option lets you choose an arbitrary table width. For example:

```
\fonttable[tablewidth=0.5\linewidth]{logo10}
```

	′0	′1	′2	′3	′4	′5	′6	′7	
′10x		A				E	F		″4x
′11x						M	N	O	
′12x	P			S	T				″5x
′13x									
	″8	″9	″A	″B	″C	″D	″E	″F	

`charrange=`⟨*range*⟩

Knuth's original table code shows every character in a given font, and that's what `\fonttable` does by default. However, the `charrange` option lets you limit the range of characters that are output to a subset of the characters available in the font. ⟨*range*⟩ is the range of character codes to output, specified as "⟨*first*⟩`-`⟨*last*⟩". ⟨*first*⟩ and ⟨*last*⟩ are both inclusive and can be specified in any number format that TEX accepts—decimal (`123`), hexadecimal (`"7B`), or octal (`'173`). If ⟨*first*⟩ is omitted (`-123`), it defaults to the first character in the font. If ⟨*last*⟩ is omitted (`123-`), it defaults to the last character in the font. Single numbers (`123`) are acceptable, as are comma-separated ranges of numbers (`65-96,123-127`). In the last case, the ranges must be specified within curly braces so that mftinc knows they are all part of `charrange`'s argument, and not the argument to a subsequent option.

`charrange` is useful when typesetting font documentation, because a section can begin by showing a table of all the glyphs that will be defined in that section. For example `punct.mf` defines the following subset of the Computer Modern fonts, according to Knuth's comments at the top of that file:

```
\fonttable[tablewidth=0.75\linewidth,
           charrange={'41,'43,'45,'47-'54,'56-'57,'72-'73,'75,
                      '100,'133,'135,'140}]{cmss10 at 11pt}
```

Table 2 shows the result of that `\fonttable` invocation. Note how only the specified characters are shown, and empty rows (more precisely, empty double-rows) are omitted from the table. Hence, the table ranges from hexadecimal `"20`–`"6F` instead of from `"00`–`"7F`.

Table 2: `cmss10` characters defined in `punct.mf`

	´0	*´1*	*´2*	*´3*	*´4*	*´5*	*´6*	*´7*	
´04x		!		#		%		'	″2x
´05x	(	)	*	+	,		.	/	
´06x									″3x
´07x			:	;		=			
´10x	@								″4x
´11x									
´12x									″5x
´13x				[		]			
´14x	‘								″6x
´15x									
	″8	″9	″A	″B	″C	″D	″E	″F	

`\setmftdefaults {⟨options⟩}`

`\setmftdefaults` It can be cumbersome to repeatedly pass the same arguments to `charexplain` or `\fonttable`. Hence, mftinc exports a `\setmftdefaults` macro. `\setmftdefaults` takes the same ⟨*key*⟩=⟨*value*⟩ pairs as `charexplain` and `\fonttable`, but uses them to change the default value of each option for all future invocations of `charexplain` and `\fonttable`:

```
\setmftdefaults{charrange=65-67,toprule=3pt,bottomrule=3pt}
\begin{explaincode}
  \textsf{mftinc}'s default parameters have been altered.
  However, it's still possible to override those defaults
  on a case-by-case basis.
  \begin{center}
    \fonttable{cmsy10 at 17pt}
    \fonttable[charrange=68-70]{cmsy10 at 17pt}
    \fonttable{cmsy10 at 17pt}
  \end{center}
\end{explaincode}
```

The result is shown in Figure 2.

mftinc's default parameters have been altered. However, it's still possible to override those defaults on a case-by-case basis.

	´0	´1	´2	´3	´4	´5	´6	´7	
´10x		$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$					″4x
´11x									
	″8	″9	″A	″B	″C	″D	″E	″F	

	´0	´1	´2	´3	´4	´5	´6	´7	
´10x					$\mathcal{D}$	$\mathcal{E}$	$\mathcal{F}$		″4x
´11x									
	″8	″9	″A	″B	″C	″D	″E	″F	

	´0	´1	´2	´3	´4	´5	´6	´7	
´10x		$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$					″4x
´11x									
	″8	″9	″A	″B	″C	″D	″E	″F	

Figure 2: Example of changing and overriding mftinc's defaults

3 Other information

This section contains miscellaneous commentary on mftinc, MFT, and other things that don't fit into any of the other sections.

3.1 mftinc copyright and license

3.2 Package dependencies

mftinc requires the rawfonts and keyval packages, both of which are included with virtually every LaTeX 2ε distribution. The `wrapcomment` environment additionally requires chngpage and lineno, which are nonstandard but freely available from CTAN (`http://www.ctan.org/`). If mftinc can't find chngpage or lineno, it will issue a *warning* message, which turns into an error message at the first `\begin{wrapcomment}`. Hence, if you merely want to include MFT output, font tables, and character-level comments and are willing to sacrifice stanza-level comments, you can avoid the bother of downloading and installing two additional packages.

3.3 Including proof and smoke images

Knuth's *Computer Modern Typefaces* [2] shows proof-mode versions of each character next to the corresponding character program. One way to do this yourself for your own fonts is to use MetaPost, which can produce an Encapsulated PostScript (EPS) image of each character in a font. The exact details may differ slightly from system to system, but here's the basic approach: First, assuming you don't already have it, you have to produce a `mfplain.mem` file. The command to do this on a Unix-based system is usually:

```
mpost -ini '\input mfplain; dump'
```

On Windows, you'll probably need to use double quotes instead of single quotes. On other systems, you're on your own.

The `mfplain.mem` files enables MetaPost to accept (most) METAFONT commands. The next step is to use MetaPost plus `mfplain.mem` to produce a proof-mode version of your font:

```
mpost -mem mfplain '\mode:=proof; prologues:=2; input ⟨filename⟩'
```

. . . or a smoke-mode version:

```
mpost -mem mfplain '\mode:=smoke; prologues:=2; input ⟨filename⟩'
```

In either case, MetaPost will produce a separate EPS file for each character in the font. These will be named ⟨*filename*⟩`.`⟨*character code*⟩. For example, the EPS file for `cmr10.mf`'s letter "A" will be called `cmr10.65`, because "A" is at position 65 in that font. You may want to give these files a `.eps` extension, so that LaTeX and other programs realize that the files are EPS. The good news is that even pdfLaTeX, which can't read arbitrary EPS files, can read MetaPost's EPS output. (By default, pdfLaTeX expects the files to have a `.mps` extension, however.)

3.4 Known bugs

The first `%%` after a `\begin{wrapcomment}` must be indented at least one space. Otherwise, LaTeX will abort with "`! LaTeX Error: \begin{wrapcomment} on input line` ⟨*line*⟩ `ended by \end{linenumbers}`".

3.5 A brief MFT reference

MFT processes comments beginning with one to four percent signs in different ways, as shown in Table 3. The MFT documentation says that comments starting with more than four percent signs are verboten. mftinc is normally used within double-percent comments, because those are passed directly to LaTeX with no additional processing on MFT's part.

Table 3: MFT comment types

Type	Description
`%`	Format a comment using TeX (or with mftinc, LaTeX), with the addition that text within vertical bars is formatted as if it were outside of the comment (i.e., as if it were METAFONT code). Single-`%` comments are output right-justified with a leading percent sign. Ending a line with `\]` makes it left-justified, though.
`%%`	Format a comment using TeX (or with mftinc, LaTeX), with none of single-`%`'s bells and whistles—no leading percent sign, no right-justification, and no support for embedded METAFONT code. mftinc's `explaincode` and `wrapcomment` environments belong within double-`%` comments.
`%%%`	Given a list of space-separated METAFONT tokens, make MFT format all of them like the first one in the list. Hence "`%%% addto mymacro`" says to format the token `mymacro` just like METAFONT's `addto` primitive.
`%%%%`	MFT discards lines beginning with quadruple-`%` comments.

`mftmac.tex`, which is `\input` by every `.tex` file that MFT produces, defines a number of macros for typesetting METAFONT (Table 4). These may be used within a `%%` comment when doing so is more convenient than mftinc's `\mfcomment` macro (e.g., if only a single symbol need be typeset). The following are the important things to note about these macros:

- They're defined to be used in math mode, so be sure to use them within `$...$`.
- The different boldfaced operators have different surrounding spacing (not always obvious from Table 4). To select the right operator, I usually look at the `.tex` file to see how it formats the operator in the font program listing.
- `\\` doesn't mean "line break", as it normally does in LaTeX; use `\newline` instead.

Table 4: Additional MFT macros

Macro	Example
`\\{`⟨*identifier*⟩`}`	*i*, *eps*
`\1{`⟨*operator*⟩`}`	length, hround
`\2{`⟨*operator*⟩`}`	**beginchar**, **for**
`\3{`⟨*closing operator*⟩`}`	**fi**, **endgroup**
`\4{`⟨*binary operator*⟩`}`	**step**, **at**
`\5{`⟨*constant*⟩`}`	**true**, **nullpicture**
`\6{`⟨*binary operator*⟩`}`	++, scaled
`\7"`⟨*string*⟩`"`	`"Hello, world!"`
`\8{`⟨*relation*⟩`}`	. ., `--`
`\?{`⟨*relation*⟩`}`	:: , ||:
`\PS`	+−+
`\SH`	#
`\frac{`⟨*num*⟩`}/{`⟨*den*⟩`}`	$^{17}/_{23}$

4 Implementation

Most users can stop reading at this point. The Implementation section contains the annotated source code for the mftinc package itself, which is useful only to people who want a detailed and precise explanation of how mftinc works. If you're planning on extending or customizing (or debugging!) the package, this is the section for you. (Note that mftinc is released under the LaTeX Project Public License, which gives your the right to make whatever modifications you want, provided you don't call the result "mftinc".)

```
⟨*package⟩
```

4.1 Including MFT-formatted files

The following code provides the minimal amount of functionality that mftinc needs to be useful: the ability to include an MFT-produced TeX file in a LaTeX 2ε document. Because mftmac uses TeX (and LaTeX 2.09) font names, such as `\tenbf`, we have to load the rawfonts compatibility package to make it work. In addition, mftmac assumes that the `\bffam` and `\itfam` font families are predefined, which they aren't in LaTeX 2_ε, so we have to define those, too.

```
\RequirePackage{rawfonts}
\newfam\bffam
\newfam\itfam
```

`\mftinput` Fortunately, most of mftmac's screwy macro definitions are defined in the local scope (i.e., with `\def` instead of `\gdef`). Hence, we can simply `\input` an MFT-formatted file within a group, and most things will go back to normal at the `\endgroup`.

```
\DeclareRobustCommand{\mftinput}[1]{\begingroup\input #1\endgroup}
```

4.2 Argument processing

The `explainchar` environment and the `\fonttable` command each take a few optional arguments. We use the keyval package to help process these arguments. Table 5 lists the arguments that are currently supported.

Table 5: Options supported by mftinc's macros and environments

Key	Applies to	Affects	Meaning
`toprule`	`explainchar`	`\mft@top@rule`	Width of the rule above `explainchar` comments
`bottomrule`	`explainchar`	`\mft@bot@rule`	Width of the rule below `explainchar` comments
`tablewidth`	`\fonttable`	`\mft@table@width`	Width of the font table
`charrange`	`\fonttable`	`\mft@ranges` and `\mft@expanded@ranges`	Comma-delimited, hyphenated ranges of characters to include in the font table

```
\RequirePackage{keyval}
\define@key{mft}{toprule}{\setlength{\mft@top@rule}{#1}}
\define@key{mft}{bottomrule}{\setlength{\mft@bot@rule}{#1}}
\define@key{mft}{tablewidth}{\setlength{\mft@table@width}{#1}}%
\define@key{mft}{charrange}{%
  \def\mft@ranges{}%
  \def\mft@expanded@ranges{}%
  \mft@parse@ranges#1,,%
  {\let\@elt=\mft@expand@range\mft@ranges}%
}
```

`\setmftdefaults` Rather than repeatedly specify the same optional arguments, one can use `\setmftdefaults` to specify default values for all mftinc macros and environments that take optional arguments. `\setmftdefaults` takes one mandatory argument, which has the same effect globally as the various macros' and environments' optional arguments have locally.

```
\DeclareRobustCommand{\setmftdefaults}[1]{\setkeys{mft}{#1}}
```

4.3 Improved comment formatting

There are three main places a font designer might want to insert code comments:

1. Before a character program or macro,
2. Before a stanza of a code within a character program or macro, and
3. On the same line as some METAFONT code.

MFT has weak support for the first two of those. While MFT passes lines starting with "`%%`" directly to TeX (or, when mftinc is used, LaTeX), text formatted this way doesn't sufficiently stand stand out from the formatted METAFONT code, in my opinion. Comments starting with `%` are normally right-justified and work well when used for brief phrases that share a line with METAFONT code, but they are cumbersome to use for longer, non-right-justified, stanza-level comments. In order to make each output line start with a `%` (to make it clear that the text is a comment and not code), the author must manually break lines and, in addition, end each line with `\]` to inhibit right-justification.

The macros that will be introduced in this section solve all of these problems.

4.3.1 Character-level comments

To clearly separate commentary from the program text that follows, we define a simple, `explaincode` environment that draws a horizontal rule above and below the contained text.

`\mft@top@rule` Specify the thickness of the rule above the `explaincode` text.

```
\newlength{\mft@top@rule}
\setlength{\mft@top@rule}{1pt}
```

`\mft@bot@rule` Specify the thickness of the rule below the `explaincode` text.

```
\newlength{\mft@bot@rule}
\setlength{\mft@bot@rule}{1pt}
```

`explaincode` Draw a rule above and below any text contained within `\begin{explaincode}`… `\end{explaincode}`. For æsethetics, we add a little stretchable glue above the first rule and a little shrinkable glue below the bottom rule. We also prohibit page breaks between the rules and the text.

```
\newenvironment{explaincode}[1][]{%
  \setkeys{mft}{#1}%
  \par\vskip 4ex \@plus 2ex
  \hrule\@height\mft@top@rule
  \nobreak\medskip\nobreak\noindent\ignorespaces
}{%
  \nobreak\medskip\nobreak
  \hrule\@height\mft@bot@rule
  \vskip 2ex \@minus 1ex
}
```

4.3.2 Stanza-level comments

We define a new environment for formatting stanza-level comments that honors the following properties:

- The comments can be multiple lines long.
- They wrap text like an ordinary block of LaTeX code.
- Each line of output begins with a percent line.
- The comments use the same indentation as the block of METAFONT code they describe.

`\mft@wc@indent` This ⟨*dimen*⟩ stores the indentation of a `wrapcomment` environment, excluding the space occupied by the initial percent signs.

```
\newlength{\mft@wc@indent}
```

`\mft@eat@quads` To figure out the correct indentation for the entire comment block, we (tail-recursively) count the number of `\quad`s in the first line, adding `1em` of space to `\mft@wc@indent` for each one encountered and discarding the `\quad` as we go. At the end, we make `\quad` a no-op, to prevent `\quad`s on subsequent lines from contributing unwanted space, indent by `\mft@wc@indent` plus the width of a percent sign, and use lineno to "number" the lines using percent signs.

```
\def\mft@eat@quads#1{%
  \ifx#1\quad
    \global\addtolength{\mft@wc@indent}{1em}%
    \expandafter\mft@eat@quads
  \else
    \def\quad{}%
    \settowidth{\@tempdima}{\%~}%
    \advance\@tempdima by \mft@wc@indent
    \vspace{-2ex}%
    \begin{adjustwidth}{\@tempdima}{}%
```

```
      \begin{linenumbers}%
        \internallinenumbers
        \renewcommand{\makeLineNumber}{%
          \rlap{\hspace*{\mft@wc@indent}\%}}%
        \expandafter#1%
  \fi
}
```

`wrapcomment`

Display a block of text that is indented to the same position as the first text after the `\begin{wrapcomment}`. `\mft@eat@quads` does most of the work. `wrapcomment` merely resets the indentation counter and makes the first `\quad` consume the rest (via `\mft@eat@quads`). The `\end{wrapcomment}` closes the `linenumbers` and `adjustwidth` environments opened by `\mft@eat@quads`.

```
\newenvironment{wrapcomment}{%
  \global\setlength{\mft@wc@indent}{0pt}%
  \def\quad{%
    \global\addtolength{\mft@wc@indent}{1em}%
    \mft@eat@quads
  }%
}{%
      \end{linenumbers}%
    \end{adjustwidth}%
}
```

`\mft@missing`

If we can't load one or both of the chngpage and lineno packages, disable the `wrapcomment` environment and issue a warning message. This is a little more user-friendly than forcing the user to download and install two packages if all he wants is to include an MFT-formatted file in a LaTeX document and has no interest in ever using the `wrapcomment` environment.

```
\def\mft@missing#1{%
  \PackageWarning{mftinc}{%
    Disabling the wrapcomment environment\MessageBreak
    (can't find #1.sty)%
  }
  \renewenvironment{wrapcomment}{%
    \PackageError{mftinc}{The wrapcomment environment is disabled}{%
      Your LaTeX installation is lacking #1.sty.\space\space
      The\MessageBreak mftinc package relies on both the chngpage
      package and\MessageBreak the lineno package in order to
      implement the wrapcomment \MessageBreak environment.\space\space
      Either install those packages, or refrain\MessageBreak from
      using wrapcomment in code that is formatted with
      mft\MessageBreak and included into LaTeX.
    }%
  }{}%
  \def\mft@missing##1{}%
}
\IfFileExists{chngpage.sty}{\RequirePackage{chngpage}}{\mft@missing{chngpage}}
\IfFileExists{lineno.sty}{\RequirePackage{lineno}}{\mft@missing{lineno}}
```

4.3.3 Other comment-related macros

`\mfcomment`

One advantage that MFT's `%` comments have over `%%` comments is that they format anything that's set between vertical bars as it if it were METAFONT code. For example, `|draw z1--z2|` is typeset as "**draw** z_1 -- z_2". The problem is that the `explaincode` and `wrapcomment` environments need to be typeset with `%%`, so their contents gets passed directly to LaTeX. To embed METAFONT code within one of those environments, one need only end the previous line with `\mfcomment` and put the METAFONT code alone on the next line, preceded by a `%`.

```
\long\def\mfcomment#1\9#2\par{\unskip#2 }
```

4.4 Font tables

TeX comes with a `testfont.tex` file that, among other things, outputs a table of all the characters in a given font. This table can be a useful addition to pretty-printed font documentation. However, `testfont.tex` is intended to be run stand-alone. The code in this section produces an identical-looking table to `testfont.tex`'s, but it can be included easily in a LaTeX document. The core of `\fontable` was taken almost verbatim from `testfont.tex`. I made the following key changes, however:

- I put everything within a `minipage`, to make it easy to move the table around and scale its width.
- I renamed all the global variables, so as to avoid potential conflicts with other packages or the main document;
- I added argument parsing to set the table width and to limit the character ranges.

4.4.1 Range processing

`\fonttable` normally shows only nonempty rows of characters. The macros in this section impose an additional limit: Only characters within certain ranges are output; the rest are treated as if they don't exist.

`\mft@ranges`
`\mft@parse@ranges`

`\mft@parse@ranges` is the top-level range-parsing function. It splits its argument into comma-separated ranges and uses `\@cons` to store these ranges in `\mft@ranges` in the form "`\@elt` $\langle \mathit{range}_1\rangle$`-!-!-!!` `\@elt` $\langle \mathit{range}_2\rangle$`-!-!-!!` ...". (The exclamation marks are needed by `\mft@expand@range` to parse the range into its components.)

```
\def\mft@ranges{}
\def\mft@parse@ranges#1,{%
  \def\mft@arg@i{#1}%
  \ifx\mft@arg@i\empty
  \else
    \@cons\mft@ranges{#1-!-!-!!}%
    \expandafter\mft@parse@ranges
  \fi
}
```

`\mft@expanded@ranges`
`\mft@gobble@range`
`\mft@expand@range`

Once `\mft@parse@ranges` has split comma-separated ranges into elements in `\mft@ranges`, the next step is to canonicalize each range, to simplify later range processing. That's what `\mft@expand@range` does. It converts each range in `\mft@ranges` to the form "`\@elt` ⟨*first*⟩`|`⟨*last*⟩`|`", in which neither ⟨*first*⟩ nor ⟨*last*⟩ is empty. Canonicalization works in the following manner:

⟨*first*⟩`-`⟨*last*⟩ ↦ ⟨*first*⟩`|`⟨*last*⟩`|`
⟨*first*⟩`-` ↦ ⟨*first*⟩`|65535|`
`-`⟨*last*⟩ ↦ `-1|`⟨*last*⟩`|`
⟨*only*⟩ ↦ ⟨*only*⟩`|`⟨*only*⟩`|`

The resulting canonicalized list is stored in `\mft@expanded@ranges`. The `\mft@expand@range` macro expects the input range to terminate with "`-!-!-!!`". This is how it distinguishes missing components from the end of the range. `\mft@gobble@range` discards any exclamation marks that remain after processing.

```
\def\mft@expanded@ranges{}
\def\mft@gobble@range#1!!{}
\def\mft@expand@range#1-#2-{%
  \def\mft@arg@i{#1}%
  \def\mft@arg@ii{#2}%
  \ifx\mft@arg@i\empty
    \def\mft@arg@i{-1}%
  \fi
  \ifx\mft@arg@ii\empty
    \def\mft@arg@ii{65535}%
  \fi
  \if\mft@arg@ii!%
    \def\mft@arg@ii{#1}%
  \fi
  \if\mft@arg@i!%
  \else
    \@cons\mft@expanded@ranges{\mft@arg@i|\mft@arg@ii|}%
  \fi
  \mft@gobble@range
}
```

4.4.2 Range checking

Once we know the set of ranges to output, we need to determine whether any characters in the current row lie within any of the ranges (`\mft@check@char`) and whether a character in a nonempty row lies within any of the ranges (`\mft@char`). These macros actually belong within `\fonttable`, but the macro nesting depth was starting to get too large—I was getting lost amid long sequences of `#`s.

`\mft@check@char`

Given an octal digit, form a number by appending it to a sequence `\mft@h` of octal digits. If the number lies within any of the ranges listed in `\mft@expanded@ranges`, output the corresponding character. Otherwise, output nothing.

```
\def\mft@check@char#1{%
  \begingroup
  \def\@elt##1|##2|{%
    \ifnum"\mft@h#1<##1
    \else
      \ifnum"\mft@h#1>##2
      \else
        \char"\mft@h#1
      \fi
    \fi
  }%
  \mft@expanded@ranges
  \endgroup
}
```

`\mft@char` If a given number lies within any of the ranges listed in `\mft@expanded@ranges`, output the corresponding character. Otherwise, output nothing.

```
\def\mft@char#1{%
  \begingroup
  \def\@elt##1|##2|{%
    \ifnum#1<##1
    \else
      \ifnum#1>##2
      \else
        \char#1
      \fi
    \fi
  }%
  \mft@expanded@ranges
  \endgroup
}
```

4.4.3 Table composition

Now that we've defined macros to parse `\fonttable`'s optional argument, to process ranges of character codes, and to check for numbers within ranges, we can finally proceed with defining `\fonttable`, the macro that actually composes the font table.

`\mft@table@width` `\mft@table@width` stores the width of the font table. Columns will expand automatically to fill that width. If the specified width is negative, `\fonttable` will instead use whatever column width is in effect when `\fonttable` is invoked.

```
\newlength{\mft@table@width}
\setlength{\mft@table@width}{-1pt}
```

`\mft@expanded@ranges` `\mft@expanded@ranges` stores a comma-separated list of hyphenated ranges. The default is a single range, `0-65535`, which encompasses all character positions.

```
\def\mft@expanded@ranges{\@elt 0|65535|}
```

`\fonttable`
`\mft@old@ranges`
`\mft@old@expanded@ranges`

Display all the characters in a given font. The first (optional) argument is a set of ⟨*key*⟩=⟨*value*⟩ pairs to specify the table width and range of characters to output. The second (mandatory) argument is the "raw" name of the font to use, e.g., `cmr10`.

```
\DeclareRobustCommand{\fonttable}[2][]{%
\begingroup
\let\mft@old@ranges=\mft@ranges
\let\mft@old@expanded@ranges=\mft@expanded@ranges
\setkeys{mft}{#1}%
\ifdim\mft@table@width<0pt
  \begin{minipage}{\linewidth}%
\else
  \begin{minipage}{\mft@table@width}%
\fi
  \font\testfont=#2\testfont
```

`\mft@m`
`\mft@n`
`\mft@p`
`\dim`

The first three of these were called `m`, `n`, and `p` in Knuth's code.

```
  \newcount\mft@m
  \newcount\mft@n
  \newcount\mft@p
  \newdimen\dim
```

`\oct`

Format an octal constant.

```
  \def\oct##1{\hbox{\rm\'{}\kern-.2em\it##1\/\kern.05em}}%
```

`\hex`

Format a hexadecimal constant.

```
  \def\hex##1{\hbox{\rm\H{}\tt##1}}%
```

`\setdigs`
`\mft@h`
`\mft@zero`
`\mft@one`

`\mft@h` is the hex prefix. `\mft@zero\mft@one` is the corresponding octal prefix. These were called `\h`, `\0`, and `\1` in Knuth's code.

```
  \def\setdigs##1"##2{\gdef\mft@h{##2}%
   \mft@m=\mft@n \divide\mft@m by 64 \xdef\mft@zero{\the\mft@m}%
   \multiply\mft@m by-64
   \advance\mft@m by\mft@n
   \divide\mft@m by 8
   \xdef\mft@one{\the\mft@m}}%
```

`\testrow`

Determine if a row is empty. `\mft@p`=1 if none of the characters exist. Note that I modified the definition of `\\` to make use of `\mft@check@char`.

```
\def\testrow{\setbox0=\hbox{\penalty 1\let\\=\mft@check@char
 \\0\\1\\2\\3\\4\\5\\6\\7\\8\\9\\A\\B\\C\\D\\E\\F%
 \global\mft@p=\lastpenalty}} % \mft@p=1 if none of the characters exist
```

`\oddline`

Draw an odd-numbered line.

```
  \def\oddline{\cr
    \noalign{\nointerlineskip}%
    \multispan{19}\hrulefill&
    \setbox0=\hbox{\lower 2.3pt\hbox{\hex{\mft@h x}}}\smash{\box0}\cr
    \noalign{\nointerlineskip}}%
```

`\ifskipping` Are we skipping empty rows?

```
\newif\ifskipping
```

`\evenline` Draw an even-numbered line.

```
\def\evenline{\loop\skippingfalse
 \ifnum\mft@n<256 \mft@m=\mft@n \divide\mft@m 16 \chardef\next=\mft@m
 \expandafter\setdigs\meaning\next \testrow
 \ifnum\mft@p=1 \skippingtrue \fi\fi
 \ifskipping \global\advance\mft@n 16 \repeat
 \ifnum\mft@n=256 \let\next=\endchart\else\let\next=\morechart\fi
 \next}%
```

`\morechart`
`\chartline`
`\chartstrut` Define a few more helper routines.

```
\def\morechart{\cr\noalign{\hrule\penalty5000}%
 \chartline \oddline \mft@m=\mft@one \advance\mft@m 1
 \xdef\mft@one{\the\mft@m}%
 \chartline \evenline}%
\def\chartline{&\oct{\mft@zero\mft@one x}%
 &&\:&&\:&&\:&&\:&&\:&&\:&&\:&&\:&&}%
\def\chartstrut{\lower4.5pt\vbox to14pt{}}%
```

`\table` Draw the entire table. In `testfont.tex`, this was one of the commands that a user would invoke at the TeX prompt.

```
\def\table{$$\global\mft@n=0
  \halign to\hsize\bgroup
    \chartstrut####\tabskip0pt plus10pt&
    &\hfil####\hfil&\vrule####\cr
    \lower6.5pt\null
    &&&\oct0&&\oct1&&\oct2&&\oct3&&\oct4&&\oct5&&\oct6&&\oct7&\evenline}%
```

`\endchart` Draw the last line of the table.

```
\def\endchart{\cr\noalign{\hrule}%
  \raise11.5pt\null&&&\hex 8&&\hex 9&&\hex A&&\hex B&
  &\hex C&&\hex D&&\hex E&&\hex F&\cr\egroup$$\par}%
\def\:{\setbox0=\hbox{\mft@char\mft@n}%
  \ifdim\ht0>7.5pt\reposition
  \else\ifdim\dp0>2.5pt\reposition\fi\fi
  \box0\global\advance\mft@n 1 }%
```

`\reposition`
`\centerlargechars` Define a few more helper routines.

```
\def\reposition{\setbox0=\vbox{\kern2pt\box0}\dim=\dp0
  \advance\dim 2pt \dp0=\dim}%
\def\centerlargechars{
  \def\reposition{\setbox0=\hbox{$\vcenter{\kern2pt\box0\kern2pt}$}}}%
```

Finally, we compose the table, finish off our `minipage`, and restore the previous values of `\mft@ranges` and `\mft@expanded@ranges` (which we had to save at the top of `\fonttable`, because `\@cons` contains an `\xdef`). This concludes the definition of `\fonttable`.

```
203   \table
204 \end{minipage}%
205 \global\let\mft@ranges=\mft@old@ranges
206 \global\let\mft@expanded@ranges=\mft@old@expanded@ranges
207 \endgroup
208 }
209 ⟨/package⟩
```

References

[1] Donald E. Knuth. *The METAFONTbook*, volume C of *Computers and Typesetting*. Addison-Wesley, Reading, Massachusetts, 1986.

[2] Donald E. Knuth. *Computer Modern Typefaces*, volume E of *Computers and Typesetting*. Addison-Wesley, Reading, Massachusetts, 1986.

Index

Numbers written in italic refer to the page where the corresponding entry is described, the ones underlined to the code line of the definition, the rest to the code lines where the entry is used.