

# Package **mathfont** v. 2.3 Implementation

Conrad Kosowsky

September 2023

`kosowsky.latex@gmail.com`

For easy, off-the-shelf use, type the following in your preamble and compile with  $\text{Xe}\text{L}\text{A}\text{T}\text{E}\text{X}$  or  $\text{L}\text{u}\text{a}\text{L}\text{A}\text{T}\text{E}\text{X}$ :

```
\usepackage[(font name)]{mathfont}
```

As of version 2.0, using  $\text{L}\text{u}\text{a}\text{L}\text{A}\text{T}\text{E}\text{X}$  is recommended.

## Overview

The **mathfont** package adapts unicode text fonts for math mode. The package allows the user to specify a default unicode font for different classes of math symbols, and it provides tools to change the font locally for math alphabet characters. When typesetting with  $\text{L}\text{u}\text{a}\text{T}\text{E}\text{X}$ , **mathfont** adds resizable delimiters, big operators, and a `MathConstants` table to text fonts.

---

This file documents the code for the **mathfont** package. It is not a user guide! If you are looking for instructions on how to use **mathfont** in your document, see `mathfont_user_guide.pdf`, which is included with the **mathfont** installation and is available on CTAN. See also the other pdf documentation files for **mathfont**. Section 1 of this document begins with the implementation basics, including package declaration and package options. Section 2 deals with errors and messaging, and section 3 provides package default settings. Section 4 contains fontloader, and section 5 contains the optional-argument parser for `\mathfont`. Section 6 documents the code for the `\mathfont` command itself. Section 7 contains the code for local font changes. Section 8 contains miscellaneous material. Sections 9–11 contain the Lua code to modify font objects at loading, and section 12 lists the unicode hex values used in symbol declaration. Version history and code index appear at the end of the document.

## 1 Implementation Basics

First and foremost, the package needs to declare itself.

```
1 \NeedsTeXFormat{LaTeX2e}
2 \ProvidesPackage{mathfont}[2023/09/09 v. 2.3 Package mathfont]
```

We specify conditionals and one count variable that we use later in handling options and setup.

---

Acknowledgements: Thanks to Lyric Bingham for her work checking my unicode hex values. Thanks to Shyam Sundar, Adrian Vollmer, Herbert Voss, and Andreas Zidak for pointing out bugs in previous versions of **mathfont**. Thanks to Jean-François Burnol for pointing out an error in the documentation in reference to his `mathastext` package.

```

3 \newif\ifM@XeTeXLuaTeX      % is engine one of xetex or luatex?
4 \newif\ifM@NoLuaotfload     % cannot find luaotfload.sty?
5 \newif\ifM@adjust@font     % should adjust fonts with lua script?
6 \newif\ifM@font@loaded     % load mathfont with font specified?
7 \newif\ifE@sterEggDecl@red % already did easter egg?
8 \newcount\M@loader         % specifies which font-loader to use

```

We disable the twenty user-level commands. If `mathfont` runs normally, it will overwrite these “bad” definitions later, but if it throws one of its two fatal errors, it will `\endinput` while the user-level commands are error messages. That way the commands don’t do anything in the user’s document, and the user gets information on why not. The bad definitions gobble their original arguments to avoid a “missing `\begin{document}`” error. To streamline the process, we metacode most of the error messages. We do so in three batches: those that `\@gobble` their argument, those that `\@gobbletwo` their argument, and those that accept an optional argument.

```

9 \long\def\@gobble@brackets[#1]{}
10 \def\M@NoMathfontError#1{\PackageError{mathfont}
11   {\MessageBreak Invalid command\MessageBreak
12   \string#1 on line \the\inputlineno}
13   {Your command was ignored. I couldn't\MessageBreak
14   load mathfont, so I never defined this\MessageBreak
15   control sequence.}}

```

First the commands that normally accept a single argument—the “bad” versions `\@gobble` the argument. To keep the syntax straightforward, we put the `\def` declaration inside `\@tempa` and the definition itself inside `\@tempb`. We need to do this because the macro name is stored in `\@i`, and otherwise, we would end up with a mess of `\expandafte`s to expand all instances of `\@i`.

```

16 \@tfor\@i:=\setfont
17   \RuleThicknessFactor
18   \IntegralItalicFactor
19   \SurdVerticalFactor
20   \SurdHorizontalFactor
21   \CharmLine
22   \CharmFile\do{%
23     \edef\@tempa{\protected\def\expandafter\noexpand\@i}%
24     \edef\@tempb{\noexpand\M@NoMathfontError
25       \expandafter\noexpand\@i
26       \noexpand\@gobble}%
27     \expandafter\@tempa\expandafter{\@tempb}}

```

Now for the macros that `\@gobbletwo` their argument. The code is essentially the same.

```

28 \@tfor\@i:=\newmathrm
29   \newmathit
30   \newmathbf
31   \newmathbfi
32   \newmathbold
33   \newmathboldi
34   \newmathsc

```

```

35 \newmathscit
36 \newmathbfsc
37 \newmathbfscit\do{%
38   \edef\@tempa{\protected\def\expandafter\noexpand\@i}%
39   \edef\@tempb{\noexpand\M@NoMathfontError
40     \expandafter\noexpand\@i
41     \noexpand\@gobbletwo}%
42   \expandafter\@tempa\expandafter{\@tempb}}

```

For the optional argument, we check if the following character is a [. If yes, we gobble first the brackets and then the mandatory argument. If not, we gobble the single mandatory argument.

```

43 \@tfor\@i:=\mathfont\mathconstantsfont\do{%
44   \edef\@tempa{\protected\def\expandafter\noexpand\@i}%
45   \edef\@tempb{\noexpand\M@NoMathfontError
46     \expandafter\noexpand\@i
47     \noexpand\@ifnextchar[{\noexpand\expandafter
48       \noexpand\@gobble\noexpand\@gobble@brackets}{\noexpand\@gobble}}
49   \expandafter\@tempa\expandafter{\@tempb}}

```

We code `\newmathfontcommand` by hand because it is the only command with four arguments.

```

50 \protected\def\newmathfontcommand{%
51   \M@NoMathfontError\newmathfontcommand\@gobblefour}

```

Check that the engine is  $X_{\text{T}}\text{T}_{\text{E}}\text{X}$  or  $\text{LuaT}_{\text{E}}\text{X}$ . If yes, set `\ifM@XeTeXLuaTeX` to true. (Otherwise the conditional will be false by default.)

```

52 \ifdefined\directlua
53   \M@XeTeXLuaTeXtrue
54 \fi
55 \ifdefined\XeTeXrevision
56   \M@XeTeXLuaTeXtrue
57 \fi

```

The package can raise two fatal errors: one if the engine is not  $X_{\text{T}}\text{T}_{\text{E}}\text{X}$  or  $\text{LuaT}_{\text{E}}\text{X}$  (and cannot load OpenType fonts) and one if  $\text{T}_{\text{E}}\text{X}$  cannot find the `luaotfload` package. In both cases, the package will stop loading, so we want a particularly conspicuous error message. For each message, we check the appropriate conditional to determine if we need to raise the error. If yes, we change space to catcode 12 inside a group. We define a `\GenericError` inside a macro and then call the macro for a cleaner error context line. The `\@gobbletwo` eats the extra period and return that  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  adds to the error message. Notice that we expand the error before the `\endgroup`—this is because we need to switch `\M@XeTeXLuaTeXError` with its replacement text while it is still defined before we leave the group. At the same time, we want `\AtBeginDocument` and `\endinput` outside the group. The second `\expandafter` means that we expand the final `\fi` before `\endinput`, which balances the original conditional.

```

58 \ifM@XeTeXLuaTeX\else
59 \begingroup
60 \catcode`\ =12\relax

```

```

61 \def\M@XeTeXLuaTeXError{\GenericError{}}%
62 {\MessageBreak\MessageBreak
63 Package mathfont error:%
64 \MessageBreak\MessageBreak
65 *****\MessageBreak
66 *                *\MessageBreak
67 *      UNABLE TO  *\MessageBreak
68 *      LOAD MATHFONT *\MessageBreak
69 *                *\MessageBreak
70 *      Missing XeTeX *\MessageBreak
71 *      or LuaTeX   *\MessageBreak
72 *                *\MessageBreak
73 *****\MessageBreak\@gobbletwo}%
74 {See the mathfont package documentation for explanation.}%
75 {I need XeTeX or LuaTeX to use mathfont. It\MessageBreak
76 looks like the current engine is something\MessageBreak
77 else, so I'm going to stop reading in the\MessageBreak
78 package file now. (You won't be able to use\MessageBreak
79 commands from mathfont in your document.) To\MessageBreak
80 load mathfont correctly, please retypeset your\MessageBreak
81 document with one of those two engines.^^J}}%
82 \expandafter\endgroup
83 \M@XeTeXLuaTeXError
84 \AtEndOfPackage{\typeout{:: mathfont :: Failed to load\on@line.}}
85 \expandafter\endinput % we should \endinput with a balanced conditional
86 \fi

```

Now do the same thing in checking for `luaotfload`. If the engine is `LuaTeX`, we tell `mathfont` to implement Lua-based font adjustments by default. The conditional `\ifM@NoLuaotfload` will keep track of whether `TeX` could find `luaotfload.sty`. If the engine is `XqTeX`, issue a warning.

```

87 \ifdefined\directlua
88 \M@adjust@fonttrue % if engine is LuaTeX, adjust font by default
89 \IfFileExists{luaotfload.sty}
90   {\M@NoLuaotfloadfalse\RequirePackage{luaotfload}}{\M@NoLuaotfloadtrue}
91 \else
92 \AtEndOfPackage{\PackageWarningNoLine{mathfont}{%
93   The current engine is XeTeX, but as\MessageBreak
94   of mathfont version 2.0, LuaTeX is\MessageBreak
95   recommended. Consider compiling with\MessageBreak
96   LuaLaTeX. Certain features will not\MessageBreak
97   work with XeTeX}}
98 \fi

```

If the engine is `LuaTeX`, we absolutely must have `luaotfload` because `LuaTeX` needs this package to load OpenType fonts. Before anything else, `TeX` should check whether it can find `luaotfload.sty` and stop reading in `mathfont` if it cannot. Same command structure as before. Newer `LATeX` versions load `luaotfload` as part of the format, but it never hurts to double check.

```

99 \ifM@Noluaotfload % false by default; true if LuaTeX AND no luaotfload.sty
100 \begingroup
101 \catcode`\ =12\relax
102 \def\M@NoluaotfloadError{\GenericError{}}%
103 {\MessageBreak\MessageBreak
104 Package mathfont error:%
105 \MessageBreak\MessageBreak
106 *****\MessageBreak
107 *                *\MessageBreak
108 *      UNABLE TO   *\MessageBreak
109 *      LOAD MATHFONT *\MessageBreak
110 *                *\MessageBreak
111 *      Cannot find the *\MessageBreak
112 *      file luaotfload.sty *\MessageBreak
113 *                *\MessageBreak
114 *****\MessageBreak@gobbletwo}%
115 {You are likely seeing this message because you haven't^^J%
116 installed luaotfload. Check your TeX distribution for a^^J%
117 list of the packages on your system.^^J^^J%
118 See the mathfont documentation for further explanation.}%
119 {You're in trouble here. It looks like the current\MessageBreak
120 engine is LuaTeX, so I need the luaotfload package\MessageBreak
121 to make mathfont work properly. However, I can't\MessageBreak
122 find luaotfload, which likely means something is\MessageBreak
123 wrong with your TeX installation. I'm going to stop\MessageBreak
124 reading in the mathfont package file. (You won't be\MessageBreak
125 able to use commands from mathfont in your document.)\MessageBreak
126 To load mathfont work correctly, make sure you have\MessageBreak
127 installed luaotfload.sty in a directory searchable\MessageBreak
128 by TeX or compile with XeLaTeX.^^J}}%
129 \expandafter\endgroup
130 \M@NoluaotfloadError
131 \AtEndOfPackage{\typeout{:: mathfont :: Failed to load\on@line.}}
132 \expandafter\endinput % we should \endinput with a balanced conditional
133 \fi

```

Some package options are now deprecated, specifically `packages`, `operators`, and `no-operators`. In the case of these options, the command `\M@Optiondeprecated` issues an error and tells the user the appropriate alternative. We check for `atveryend` to use with the easter egg.

```

134 \def\M@Optiondeprecated#1#2{\PackageError{mathfont}
135 {Option "#1" deprecated}
136 {Your option was ignored. Please\MessageBreak
137 use #2\MessageBreak
138 instead. For more information,\MessageBreak
139 see the mathfont documentation.}}

```

Now we code the package options. The deprecated options cause an error.

```

140 \DeclareOption{packages}{%

```

```

141 \M@Optiondeprecated{packages}
142 {the macro \string\restoremathinternals}}
143 \DeclareOption{operators}{%
144 \M@Optiondeprecated{operators}
145 {the bigops keyword with \string\mathfont}}
146 \DeclareOption{no-operators}{%
147 \M@Optiondeprecated{no-operators}
148 {the bigops keyword with \string\mathfont}}
Easter egg!
149 \DeclareOption{easter-egg}{%
150 \ifE@sterEggDecl@red\else
151 \E@sterEggDecl@redtrue
152 \def\EasterEggUpdate{\show\E@sterEggUpd@te}
153 \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
154 Okay, opening your Easter egg.^^J^^J}
155 \EasterEggUpdate
156 \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
157 Uh oh. It looks like^^J%
158 your Easter egg flew^^J%
159 out the window. I don't^^J%
160 suppose you know the^^J%
161 best kind of bait to^^J%
162 lure an egg?^^J^^J}
163 \EasterEggUpdate
164 \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
165 Still wrangling. Try back later.^^J^^J}
166 \AtBeginDocument{\bgroup
167 \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J:%
168 If we have zero eggs^^J%
169 and zero bunnies, how^^J%
170 many gnats does it take^^J%
171 to change a lightbulb??^^J^^J}
172 \EasterEggUpdate
173 \egroup}
174 \AtEndDocument{%
175 \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
176 Happy, happy day! Happy,^^J%
177 happy day! Clap your hands,^^J%
178 and be glad your hovercraft^^J%
179 isn't full of eels!^^J^^J}
180 \EasterEggUpdate
181 \let\E@sterEggUpd@te\relax
182 \let\EasterEggUpdate\relax}
183 \fi}% my easter egg :)

```

The five real package options. The default-loader and fontspec-loader tell mathfont what to use as a backend for loading fonts.

```
184 \DeclareOption{default-loader}{\M@loader\z@}
```

```
185 \DeclareOption{fontspec-loader}{\M@loader@one}
```

The options `adjust` and `no-adjust` overwrite `mathfont`'s default decision about whether to apply Lua-based font adjustments to all future fonts loaded.

```
186 \DeclareOption{adjust}{\M@adjust@fonttrue}
```

```
187 \DeclareOption{no-adjust}{\M@adjust@fontfalse}
```

Interpret an unknown option as a font name and save it for loading. In this case, the package sets `\ifM@font@loaded` to true and stores the font name in `\M@font@load`.

```
188 \DeclareOption*{\M@font@loadedtrue\edef\M@font@load{\CurrentOption}}
```

```
189 \ProcessOptions*
```

For the font-loader, we have a bit of processing to do. First print an informational message in the log file. The default loader is easy, but if the user requests `fontspec`, we have to make sure to load everything properly.

```
190 \ifcase\M@loader
```

```
191   \wlog{Package mathfont Info: Default font-loader was
```

```
192     requested for font loading.}
```

```
193 \or
```

```
194   \wlog{Package mathfont Info: Package fontspec was
```

```
195     requested for font loading.}
```

If `fontspec` was already loaded, check whether `\g__fontspec_math_bool` is true or not. If it is, change it to false.

```
196   \@ifpackageloaded{fontspec}
```

```
197     {\wlog{Package mathfont Info: Package fontspec detected.}
```

```
198       \csname bool_if:NTF\expandafter\endcsname
```

```
199       \csname g__fontspec_math_bool\endcsname
```

```
200       {\wlog{Package mathfont Info: Setting
```

```
201         \string\g__fontspec_math_bool to false.}
```

```
202       \csname bool_set_false:N\expandafter\endcsname
```

```
203       \csname g__fontspec_math_bool\endcsname}{\relax}}
```

If `fontspec` was not loaded, check that the package file exists.

```
204   {\wlog{Package mathfont Info: Package fontspec not detected.}
```

```
205     \IfFileExists{fontspec.sty}
```

```
206       {\wlog{Package mathfont Info: File fontspec.sty was found.}
```

```
207         \wlog{Package mathfont Info: Loading fontspec.}
```

```
208         \RequirePackage[no-math]{fontspec}}
```

```
209     {\PackageError{mathfont}{Missing package fontspec;\MessageBreak
210       using default font-loader instead}
```

```
211       {You requested fontspec as the font-loader for\MessageBreak
```

```
212       mathfont. However, I can't find the fontspec\MessageBreak
```

```
213       package file, so I'm going to use the default\MessageBreak
```

```
214       font-loader instead. (This likely means that\MessageBreak
```

```
215       something is wrong with your TeX installation.)\MessageBreak
```

```
216       Check your TeX distribution for a list of the\MessageBreak
```

```
217       packages installed on your system. To resolve\MessageBreak
```

```
218       this error, make sure fontspec is installed in\MessageBreak
```

```
219       a directory searchable by TeX or load mathfont\MessageBreak
```

```

220         with the default-loader option.^^J}
221         \M@loader\z@}}
222 \fi

```

We print an informational message specifying the font-loader in use. We store default OpenType features in `\M@otf@features`. The contents depend on the font-loader because we use `XYTeX/luatload` syntax versus `fontspec` syntax. By default, `mathfont` loads fonts with Latin script, default language, `TeX` and common ligatures, and lining numbers.

```

223 \ifcase\M@loader
224   \wlog{Package mathfont Info: Using default font-loader.}
225   \AtEndOfPackage{\typeout{:: mathfont :: Using default font-loader.}}
226   \def\M@otf@features{script=latin;language=DFLT;+tlig;+liga;+lnum}
227 \or
228   \wlog{Package mathfont Info: Using fontspec as font-loader.}
229   \AtEndOfPackage{\typeout{:: mathfont :: Using fontspec as font-loader.}}
230   \def\M@otf@features{Script=Latin,
231     Language=Default,
232     Ligatures={TeX,Common},
233     Numbers=Lining}
234 \fi

```

We print an informational message depending on whether the user enabled Lua-based font adjustments. If `\directlua` is defined, that means we are using `LuaTeX`, so we print a message depending on `\ifM@adjust@font`.

```

235 \ifdefined\directlua
236   \ifM@adjust@font
237     \wlog{Package mathfont Info: Enabling Lua-based font adjustments.}
238     \AtEndOfPackage{%
239       \typeout{:: mathfont :: Lua-based font adjustments enabled.}}
240   \else
241     \wlog{Package mathfont Info: Disabling Lua-based font adjustments.}
242     \AtEndOfPackage{%
243       \typeout{:: mathfont :: Lua-based font adjustments disabled.}}
244   \fi
245 \else

```

If `\directlua` is undefined, we make sure Lua-based font adjustments are disabled, and we issue an error if the user tried to manually enable them.

```

246   \ifM@adjust@font
247     \PackageError{mathfont}{Option^^J"adjust" ignored with XeTeX}
248     {Your package option "adjust" was ignored.\MessageBreak
249     This option works only with LuaTeX, and it\MessageBreak
250     looks like the current engine is XeTeX. To\MessageBreak
251     enable Lua-based font adjustments, typeset\MessageBreak
252     with LuaLaTeX.^^J}
253     \M@adjust@fontfalse
254   \fi
255   \wlog{Package mathfont Info: Disabling Lua-based font adjustments.}
256   \AtEndOfPackage{%

```



```

257 \typeout{:: mathfont :: Lua-based font adjustments disabled.}}
258 \fi

```

## 2 Errors and Messaging

Some error and informational messages. Table 1 lists all macros defined in this section along with a brief description of their use. We begin with general informational messages.

```

259 \def\M@SymbolFontInfo#1#2#3#4{\wlog{^^JPackage mathfont Info:
260 Declaring new symbol font from #1!^^J%
261 NFSS Family Name: \space#2^^J%
262 Series/Shape Info: #3^^J%
263 Symbol Font Name: \space#4^^J}}
264 \def\M@FontChangeInfo#1#2{\wlog{Package mathfont Info:

```

**Table 1: Various Messages and Errors and Their Uses**

Command	Use
<code>\M@FontChangeInfo</code>	Use a symbol font inside <code>\mathfont</code>
<code>\M@NewFontCommandInfo</code>	Declare new alphanumeric font-change command
<code>\M@SymbolFontInfo</code>	Declare new symbol font
<code>\M@CharsSetWarning</code>	Warning when calling <code>\mathfont</code> multiple times for same keyword
<code>\M@InternalsRestoredError</code>	User called <code>\mathfont</code> after restoring kernel
<code>\M@MissingNFSSShapesWarning</code>	Warning if font is missing shapes
<code>\M@NoBaseModeDetectedWarning</code>	Warning if no base-mode version of a font
<code>\M@InvalidOptionError</code>	Bad option for <code>\mathfont</code>
<code>\M@InvalidSupoptionError</code>	Bad suboption for <code>\mathfont</code>
<code>\M@MissingOptionError</code>	Missing an option for <code>\mathfont</code>
<code>\M@MissingSuboptionError</code>	Missing suboption for <code>\mathfont</code>
<code>\M@BadMathConstantsFontError</code>	Argument not previously fed to <code>\mathfont</code>
<code>\M@BadMathConstantsFontTypeError</code>	Argument not “upright” or “italic”
<code>\M@LuaTeXOnlyWarning</code>	User called <code>\mathcontsantsfont</code> in $\TeX$
<code>\M@DeprecatedWarning</code>	Warning for certain deprecated macros
<code>\M@DoubleArgError</code>	Gave multiple tokens to be the font-change macro
<code>\M@HModeError</code>	Font-change command used outside math mode
<code>\M@MissingControlSequenceError</code>	No macro provided to be font-change command
<code>\M@BadIntegerError</code>	Font metric adjustment value was not an integer
<code>\M@ForbiddenCharmFile</code>	Charm file contains a bad character
<code>\M@ForbiddenCharmLine</code>	Charm line contains a bad character
<code>\M@NoFontAdjustError</code>	Command called when Lua-based font adjustment was disabled

```

265 Setting #1 chars to #2!}}
266 \def\M@NewFontCommandInfo#1#2#3#4#5{\wlog{^^JPackage mathfont Info:
267 Creating \string#1 using #2!^^J%
268 NFSS Family Name: \space#3^^J%
269 Series/Shape Info: #4/#5^^J}}
270 \def\M@CharsSetWarning#1{\PackageWarning{mathfont}
271 {I already set the font for\MessageBreak
272 #1 chars, so I'm ignoring\MessageBreak
273 this option for \string\mathfont\space
274 on line \the\inputlineno\@gobble}}
275 \def\M@MissingNFSSShapesWarning#1#2{\PackageWarning{mathfont}
276 {The nfss family "#1"\MessageBreak
277 from line \the\inputlineno\space is missing shapes. You\MessageBreak
278 may see some substitutions or errors.\MessageBreak
279 Missing shape(s):#2\@gobble}}
280 \def\M@NoBaseModeDetectedWarning#1{\PackageWarning{mathfont}
281 {I couldn't find a base-mode version of\MessageBreak
282 the nfss family "#1"\MessageBreak
283 from line \the\inputlineno, so I'm using the family\MessageBreak
284 you specified for the math font. Some\MessageBreak
285 font features may not work in math\MessageBreak
286 mode\@gobble}}

```

Warnings for the `\mathbb`, etc. commands. Warning for deprecated commands.

```

287 \def\M@DeprecatedWarning#1#2{\PackageWarning{mathfont}
288 {Your \string#1\space command on\MessageBreak
289 line \the\inputlineno\space is deprecated, and I\MessageBreak
290 replaced it with \string#2\@gobble}}

```

Error messages associated with `\mathfont`.

```

291 \def\M@InvalidOptionError#1{\PackageError{mathfont}
292 {Invalid^^Joption "#1" for \string\mathfont\on@line}
293 {Hm. You used a keyword that isn't actually an optional\MessageBreak
294 argument for \string\mathfont. Check
295 that you spelled the keyword\MessageBreak
296 correctly. Otherwise, I'm not sure what's wrong. Is this\MessageBreak
297 option listed in the package documentation? In any event,\MessageBreak
298 I'm going to ignore it.^^J}}
299 \def\M@InvalidSuboptionError#1{\PackageError{mathfont}
300 {Invalid^^Jsuboption "#1" for \string\mathfont\on@line}
301 {Hm. You used a keyword that isn't actually a suboption\MessageBreak
302 for \string\mathfont. Check that you
303 spelled the keyword correctly.\MessageBreak
304 Otherwise, I'm not sure what's wrong. Is this suboption\MessageBreak
305 listed in the package documentation? In any event, I'm\MessageBreak
306 going to ignore it.^^J}}
307 \def\M@MissingOptionError{\PackageError{mathfont}
308 {Missing^^Joption for \string\mathfont\on@line}
309 {It looks like you included a , or = in\MessageBreak

```

```

310 the optional argument of \string\mathfont\space but\MessageBreak
311 didn't put anything before it.^^J}}
312 \def\M@MissingSuboptionError{\PackageError{mathfont}
313 {Missing^^Jsuboption for \string\mathfont\on@line}
314 {It looks like you included an = somewhere\MessageBreak
315 in the optional argument of \string\mathfont\space but\MessageBreak
316 didn't put the suboption after it. Either\MessageBreak
317 that or you typed == instead of =.^^J}}
318 \def\M@InternalsRestoredError{\PackageError{mathfont}
319 {Internal^^Jcommands restored}
320 {This package slightly changes two LaTeX\MessageBreak
321 internal commands, and you really shouldn't\MessageBreak
322 be loading new math fonts without those\MessageBreak
323 adjustments. What happened here is that you\MessageBreak
324 used \string\mathfont\space in a situation where those\MessageBreak
325 two commands retain their original defini-\MessageBreak
326 tions. Presumably you used \string\mathfont\space after\MessageBreak
327 calling the \string\restoremathinternals\space command.\MessageBreak
328 I'm going to ignore this call to \string\mathfont.\MessageBreak
329 Try typesetting this document with all\MessageBreak
330 \string\mathfont\space commands placed before you call\MessageBreak
331 \string\restoremathinternals.^^J}}

```

Error messages for \mathconstantsfont.

```

332 \def\M@BadMathConstantsFontError#1{\PackageError{mathfont}
333 {Invalid font specifier\MessageBreak
334 for \string\mathconstantsfont:\MessageBreak
335 "#1"}
336 {Your command was ignored--I can't parse your argument.\MessageBreak
337 Please make sure to use text that you have previously\MessageBreak
338 fed to \string\mathfont\space for the argument of
339 \string\mathconstantsfont.^^J}}
340 \def\M@BadMathConstantsFontTypeError#1{\PackageError{mathfont}
341 {Invalid\MessageBreak font specifier for
342 \string\mathconstantsfont:\MessageBreak"#1"}
343 {The optional argument of \string\mathconstantsfont\MessageBreak
344 should be "upright" or "italic." Right now,\MessageBreak
345 it's "#1."^^J}}
346 \def\M@MathConstantsNoFontAdjustWarning{\PackageWarning{mathfont}
347 {Your \string\mathconstantsfont\space
348 on line \the\inputlineno\MessageBreak
349 is mainly for use in LuaTeX with font\MessageBreak
350 adjustments enabled. In the current\MessageBreak
351 situation, it is probably not doing\MessageBreak
352 anything\@gobble}}

```

Error messages for the \newmathrm, etc. commands.

```

353 \def\M@MissingControlSequenceError#1#2{\PackageError{mathfont}
354 {Missing control sequence\MessageBreak

```

```

355 for\string#1\MessageBreak on input line \the\inputlineno}
356 {Your command was ignored. Right now the\MessageBreak
357 first argument of \string#1\space is "#2."\MessageBreak
358 Please use a control sequence instead.^^J}}
359 \def\M@DoubleArgError#1#2{\PackageError{mathfont}
360 {Multiple characters in\MessageBreak
361 first argument of \string#1\MessageBreak
362 on input line \the\inputlineno}
363 {Your command was ignored. Right now the\MessageBreak
364 first argument of \string#1\space is "#2,"\MessageBreak
365 which is multiple characters. Please use\MessageBreak
366 a single character instead.^^J}}
367 \def\M@HModeError#1{\PackageError{mathfont}
368 {Missing \string$ inserted\MessageBreak
369 on input line line \the\inputlineno}
370 {I generated an error because
371 you used \string#1\space outside of\MessageBreak
372 math mode. I inserted a \string$
373 before your \string#1, so we\MessageBreak
374 should be all good now.^^J}}

```

We need error messages related to Lua-based font adjustments.

```

375 \def\M@ForbiddenCharmLine#1{\PackageError{mathfont}
376 {Forbidden charm info contains #1}
377 {The argument of your \string\CharmLine\space
378 macro on line \the\inputlineno\MessageBreak
379 contains the character #1, which will mess me up\MessageBreak
380 if I try to read it, so I'm ignoring this call\MessageBreak
381 to \string\CharmLine. To resolve this error, make sure\MessageBreak
382 your charm information contains only integers,\MessageBreak
383 floats, asterisks, commas, and spaces.^^J}}
384 \def\M@ForbiddenCharmFile#1{\PackageError{mathfont}
385 {Forbidden charm info contains #1}
386 {One of the lines in your \string\CharmFile\space
387 from line \the\inputlineno\MessageBreak
388 contains the character #1, which will mess me up\MessageBreak
389 if I try to read it, so I'm ignoring this line\MessageBreak
390 from your file. To resolve this error, make sure\MessageBreak
391 your charm information contains only integers,\MessageBreak
392 floats, asterisks, commas, and spaces.^^J}}
393 \def\M@NoFontAdjustError#1{\PackageError{mathfont}
394 {Your command \MessageBreak\string#1 is invalid\MessageBreak
395 without Lua-based font adjustments}
396 {You haven't enabled Lua-based font adjustments,\MessageBreak
397 but the macro you called won't do anything without\MessageBreak
398 them. I'm going to ignore your command for now. To\MessageBreak
399 resolve this error, load mathfont with the package\MessageBreak
400 option "adjust" or compile with LuaLaTeX.^^J}}

```

```

401 \def\M@BadIntegerError#1#2{\PackageError{mathfont}
402 {Bad argument for\MessageBreak\string#1}
403 {Your command was ignored. Please make sure\MessageBreak
404 that your argument of \string#1\space\MessageBreak
405 is a nonnegative integer. Right now it's\MessageBreak
406 "#2".^^J}}

```

### 3 Default Settings

We save four macros from the  $\LaTeX$  kernel so we can change their definitions. To adapt the symbol declaration macros for use with unicode fonts, we reverse the conversion to hexadecimal in `\count0` and change the `\math` primitive to `\Umath`. Whereas the traditional primitives accept hexadecimal input, `\Umath` primitives accept decimal input with a `+` sign.

```

407 \let\@@set@mathchar\set@mathchar
408 \let\@@set@mathsymbol\set@mathsymbol
409 \let\@@set@mathaccent\set@mathaccent
410 \let\@@DeclareSymbolFont\DeclareSymbolFont
411 \@onlypreamble\@@set@mathchar
412 \@onlypreamble\@@set@mathsymbol
413 \@onlypreamble\@@set@mathaccent
414 \@onlypreamble\@@DeclareSymbolFont
415 \wlog{Package mathfont Info: Adapting \noexpand\set@mathchar for unicode.}
416 \wlog{Package mathfont Info: Adapting \noexpand\set@mathsymbol for unicode.}
417 \wlog{Package mathfont Info: Adapting \noexpand\set@mathaccent for unicode.}
418 \wlog{Package mathfont Info: Increasing upper bound on
419 \noexpand\DeclareSymbolFont to 256.}

```

Kernel command to set math characters from keystrokes.

```

420 \def\set@mathchar#1#2#3#4{%
421 \multiply\count\z@ by 16\relax
422 \advance\count\z@\count\tw@
423 \global\Umathcode`#2=\mathchar@type#3+#1+\count\z@\relax}

```

Kernel command to set math characters from control sequences.

```

424 \def\set@mathsymbol#1#2#3#4{%
425 \multiply\count\z@ by 16\relax
426 \advance\count\z@\count\tw@
427 \global\Umathchardef#2=\mathchar@type#3+#1+\count\z@\relax}

```

Kernel command to set accents.

```

428 \def\set@mathaccent#1#2#3#4{%
429 \multiply\count\z@ by 16\relax
430 \advance\count\z@\count\tw@
431 \protected\xdef#2{%
432 \Umathaccent\mathchar@type#3+\number#1+\the\count\z@\relax}}

```

We increase the upper bound on the number of symbol fonts to be 256.  $\text{Lua}\TeX$  and  $\text{Xe}\TeX$  allow up to 256 math families, but the  $\text{L}\TeX$  kernel keeps the old upper bound of 16 symbol fonts under these two engines. We patch `\DeclareSymbolFont` to change the `\count18<15`

to `\count18<\e@mathgroup@top`, where `\e@mathgroup@top` is the number of math families and is 256 in  $X_{\square}\text{T}_{\text{E}}\text{X}$  and  $\text{LuaT}_{\text{E}}\text{X}$ . Because macro patching is complicated, the next few lines may seem somewhat esoteric. Our approach is to get a sanitized definition with `\meaning` and `\strip@prefix`, implement the patch by expanding `\M@p@tch@decl@re`, and retokenize the whole thing. A simpler approach, such as calling `\M@p@tch@decl@re` directly on the expansion of `\DeclareSymbolFont`, won't work because of how  $\text{T}_{\text{E}}\text{X}$  stores and expands parameter symbols inside macros.

As of November 2022, the  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  kernel team the name of `\DeclareSymbolFont` to `\DeclareSymbolFont@m@dropped`, and now `\DeclareSymbolFont` is a wrapper around the renamed macro. This was done for error checking purposes to remove extra `m`'s from certain NFSS family names. This means that if `\DeclareSymbolFont@m@dropped` is defined, we should patch that macro, and otherwise, we should patch `\DeclareSymbolFont`.

```

433 \ifx\DeclareSymbolFont@m@dropped\@undefined
434   \edef\@tempa{\expandafter\strip@prefix\meaning\DeclareSymbolFont}
435   \def\@tempb{\def\DeclareSymbolFont##1##2##3##4##5}
436 \else
437   \edef\@tempa{\expandafter\strip@prefix\meaning\DeclareSymbolFont@m@dropped}
438   \def\@tempb{\def\DeclareSymbolFont@m@dropped##1##2##3##4##5}
439 \fi
440 \def\M@p@tch@decl@re#1<15#2\@nil{#1<\e@mathgroup@top#2}
441 \edef\M@DecSymDef{\expandafter\M@p@tch@decl@re\@tempa\@nil}

```

Now `\M@DecSymDef` contains the patched text of our new `\DeclareSymbolFont`, all with catcode 12. In order to make it useable, we have to retokenize it. We use `\scantextokens` in  $\text{LuaT}_{\text{E}}\text{X}$  and a safe version of `\scantokens` in  $X_{\square}\text{T}_{\text{E}}\text{X}$ . We store the `\def\DeclareSymbolFont` and parameter declaration in a separate macro `\@tempa` to make it easy to expand around them when we redefine `\DeclareSymbolFont`.

```

442 \ifdefined\directlua
443   \expandafter\@tempb\expandafter{\scantextokens\expandafter{\M@DecSymDef}}

```

Unfortunately, while `\scantextokens` is straightforward, `\scantokens` is a menace. The problem is that when it expands, the primitive inserts an end-of-file token (because `\scantokens` mimics writing to a file and `\inputing` what it just wrote) after the retokenized code, and this is why `\scantokens` can produce an end-of-file error. The easiest way to make the command useable is to put a `\noexpand` before the end-of-file token with `\everyeof`, and at the same time, this needs to happen inside an `\edef` so that  $\text{T}_{\text{E}}\text{X}$  handles the `\noexpand` as it is first seeing the end-of-file token. In order to prevent the `\edef` from also expanding our retokenized definition of `\DeclareSymbolFont`, we put the definition inside an `\unexpanded`.

```

444 \else
445   \begingroup
446   \everyeof{\noexpand}
447   \endlinechar\m@ne

```

The first `\edef` expands `\M@DecSymDef` and defines `\M@retokenize` to be `\scantokens{\unexpanded{\langle new definition \rangle}}`, and the second `\edef` carries out the retokenization. Once we have stored the patched definition in `\M@retokenize`, we expand `\M@retokenize` after the `\endgroup` and redefine `\DeclareSymbolFont` by calling `\@tempa`.

```

448 \edef\M@retokenize{\noexpand\scantokens{\noexpand\unexpanded{\M@DecSymDef}}}
449 \edef\M@retokenize{\M@retokenize}
450 \expandafter\endgroup
451 \expandafter\@tempb\expandafter{\M@retokenize}
452 \fi

```

We need to keep track of the number of times we have loaded fonts, and `\M@count` fulfills this role. The `\M@toks` object will record a message that displays in the log file when the user calls `\mathfont`. The `\newread` is for Lua-based font adjustments.

```

453 \newbox\surdbox
454 \newcount\M@count
455 \newcount\M@RuleThicknessFactor
456 \newcount\M@IntegralItalicFactor
457 \newcount\M@SurdVerticalFactor
458 \newcount\M@SurdHorizontalFactor
459 \newmuskip\radicandoffset
460 \newread\M@Charm
461 \newtoks\M@toks
462 \M@count\z@
463 \M@RuleThicknessFactor\@m
464 \M@IntegralItalicFactor=400\relax
465 \M@SurdHorizontalFactor\@m
466 \M@SurdVerticalFactor\@m
467 \radicandoffset=3mu\relax

```

Necessary booleans and default math font shapes.

```

468 \newif\ifM@upper
469 \newif\ifM@lower
470 \newif\ifM@diacritics
471 \newif\ifM@greekupper
472 \newif\ifM@greeklower
473 \newif\ifM@agreekupper
474 \newif\ifM@agreeklower
475 \newif\ifM@cyrillicupper
476 \newif\ifM@cyrilliclower
477 \newif\ifM@hebrew
478 \newif\ifM@digits
479 \newif\ifM@operator
480 \newif\ifM@symbols
481 \newif\ifM@extsymbols
482 \newif\ifM@delimiters
483 \newif\ifM@radical
484 \newif\ifM@arrows
485 \newif\ifM@bigops
486 \newif\ifM@extbigops
487 \newif\ifM@bb
488 \newif\ifM@cal
489 \newif\ifM@frak
490 \newif\ifM@bcal

```

```

491 \newif\ifM@bfrac
492 \newif\ifM@optionpresent
493 \newif\ifM@suboptionpresent
494 \newif\ifM@arg@good
495 \newif\ifM@Decl@reF@mily
496 \newif\ifM@fromCharmFile

```

Default shapes.

```

497 \def\M@uppershape{italic}           % latin upper
498 \def\M@lowershape{italic}          % latin lower
499 \def\M@diacriticsshape{upright}    % diacritics
500 \def\M@greekuppershape{upright}    % greek upper
501 \def\M@greeklowershape{italic}     % greek lower
502 \def\M@agreekuppershape{upright}   % ancient greek upper
503 \def\M@agreeklowershape{italic}    % ancient greek lower
504 \def\M@cyrillicuppershape{upright} % cyrillic upper
505 \def\M@cyrilliclowershape{italic}  % cyrillic lower
506 \def\M@hebrewshape{upright}        % hebrew
507 \def\M@digitsshape{upright}        % numerals
508 \def\M@operatorshape{upright}      % operator font
509 \def\M@delimitersshape{upright}   % delimiters
510 \def\M@radicalshape{upright}       % surd
511 \def\M@bigopssshape{upright}       % big operators
512 \def\M@extbigopssshape{upright}    % extended big operators
513 \def\M@symbolsshape{upright}       % basic symbols
514 \def\M@extsymbolsshape{upright}    % extended symbols
515 \def\M@arrowsshape{upright}        % arrows
516 \def\M@bbshape{upright}            % blackboard bold
517 \def\M@calshape{upright}           % caligraphic
518 \def\M@frakshape{upright}          % fraktur
519 \def\M@bcalshape{upright}           % bold caligraphic
520 \def\M@bfrakshape{upright}         % bold fraktur

```

The `\M@keys` list stores all the possible keyword options, and `\M@defaultkeys` stores the character classes that `\mathfont` acts on by default.

```

521 \def\M@keys{upper,lower,diacritics,greekupper,%
522  greeklower,agreekupper,agreeklower,cyrillicupper,%
523  cyrilliclower,hebrew,digits,operator,delimiters,%
524  radical,bigops,extbigops,symbols,extsymbols,arrows,%
525  bb,cal,frak,bcal,bfrak}
526 \def\M@defaultkeys{upper,lower,diacritics,greekupper,%
527  greeklower,digits,operator,symbols}

```

If the user enabled Lua-based font adjustments, the `\M@defaultkeys` list also includes delimiters, surd, and big operator symbols.

```

528 \ifM@adjust@font
529  \edef\M@defaultkeys{\M@defaultkeys,delimiters,radical,bigops}
530 \fi

```



## 4 Fontloader

We come to the fontloader. The main font-loading macro is `\M@newfont`. It expects a font name and OpenType feature information in its argument, and it parses the information, adds fonts to the NFSS if necessary, and stores NFSS font family names in `\M@fontn@me` and `\M@fontn@me@base`. (Advanced users: please do not call `\M@newfont` directly because its implementation may change. Instead call `\mathfont` with the `empty` keyword and extract the NFSS family name from `\M@fontn@me` or `\M@fontn@me@base`.) Our general approach is to feed the mandatory argument of `\mathfont` to `\M@newfont`, then proceed in one of three ways: (1) if the argument of `\mathfont` corresponds to a font family already in the NFSS, including one that `mathfont` created through the built-in font-loader, check what shapes are present and issue a warning if we appear to be missing any important ones; (2) if the argument is not a family in the NFSS and `\M@loader` is 0, we use the default font-loader, which is essentially a wrapper around code that we would expect to find in a typical `fd` file; or (3) otherwise, we feed the font name and feature information directly to `fontspec` and save the corresponding family name for reference later.

We use two control sequences for the family names because in Lua $\TeX$ , we load the font twice. The first time uses a default or user-specified renderer (typically node mode), and the second time uses base mode. Node mode is better for text, but it has more limited capabilities in math mode. Accordingly, we use a node-mode version of the font for text and the base-mode version for math, and advanced users who want to manually switch to a given `\fontfamily` at some point in their document are probably better off using the family that `mathfont` loaded with the default renderer. The package stores the NFSS family name for this version of the font in `\M@fontn@me`, and `\M@fontn@me@base` contains the NFSS family name of the base-mode version. In X $\TeX$ , these control sequences will be identical. Given a font family name from `\M@fontn@me`, it is possible to find the corresponding base-mode family name by calling `\csname\M@fontn@me-base\endcsname`. With `mathfont`'s built-in font-loader, the family name will be the mandatory argument of `\mathfont` with spaces removed, and the family name from using `fontspec` will be different.

We begin with commands to add series and shape information to the NFSS for a given font family. The `\M@declare@shape` macro takes several arguments. It checks whether the series/shape pair exists in the NFSS, and if not, it adds it using `\DeclareFontShape`. The argument structure is

- #1—NFSS font family name
- #2—optional /B or /I (or /BI) suffix on the font name
- #3—a list of (default) OpenType feature tags
- #4—a list of (the user's) OpenType feature tags
- #5—NFSS series identifier
- #6—NFSS shape identifier

We assume that the font file reference has already been stored in `\@tempbase`.

```

531 \def\M@declare@shape#1#2#3#4#5#6{%
532   \ifcsname TU/#1/#5/#6\endcsname
533   \else
534     \DeclareFontShape{TU}{#1}{#5}{#6}{<->"\@tempbase#2:#3;#4"}{}
```

535 `\fi}`

The `\M@fill@nfss@shapes` command does the work of populating the NFSS with the correct shape information. The argument structure is:

- #1—NFSS font family name
- #2—a list of (default) OpenType feature tags
- #3—a list of (the user’s) OpenType feature tags

We call `\M@declare@shape` for each combination of medium/bold series and upright/italic shape, and the result is an entry in the NFSS for each combination. We have separate declarations for regular and small caps because they have different shape identifiers in the NFSS. We manually set `smcp` to be `true` or `false` accordingly.

```
536 \def\M@fill@nfss@shapes#1#2#3{%
537   \for\@i:=#1}{#2;-smcp}{#3}{\mddefault}{\shapedefault},%
538   {#1}{/I}{#2;-smcp}{#3}{\mddefault}{\itdefault},%
539   {#1}{/B}{#2;-smcp}{#3}{\bfdefault}{\shapedefault},%
540   {#1}{/BI}{#2;-smcp}{#3}{\bfdefault}{\itdefault},%
```

And do small caps. If a small caps font face is separate from the main font file,  $\TeX$  won’t be able to find it automatically. In that case, you will have to write your own `fd` file or `\DeclareFontShape` commands.

```
541   {#1}{#2;+smcp}{#3}{\mddefault}{\scdefault},%
542   {#1}{/I}{#2;+smcp}{#3}{\mddefault}{\scdefault\itdefault},%
543   {#1}{/B}{#2;+smcp}{#3}{\bfdefault}{\shapedefault},%
544   {#1}{/BI}{#2;+smcp}{#3}{\bfdefault}{\scdefault\itdefault}%
545   \do{\expandafter\M@declare@shape\@i}}
```

The `\M@check@nfss@shapes` macro checks if a font family has shapes declared in upright, italic, bold, and bold italic. If any of those shapes are missing, we issue a warning. We store the missing series/shape pairs in `\@tempb` to print them as part of the warning message.

```
546 \def\M@check@nfss@shapes#1{%
547   \let\@tempb\@empty
548   \let\@tempwarning\@gobble
549   \for\@i:=\mddefault/\shapedefault,%
550     \mddefault/\itdefault,%
551     \bfdefault/\shapedefault,%
552     \bfdefault/\itdefault\do{%
553     \expandafter\ifx\cename TU/#1/\@i\endcename\relax
554     \def\@tempwarning{\M@MissingNFSSShapesWarning{#1}}
555     \edef\@tempb{\@tempb, \@i}
556   \fi}
```

We use a small hack to get everything to print correctly. If all shapes are present, then `\@tempwarning` is `\@gobble`, and the argument disappears. Otherwise, the argument becomes part of the warning message. The `\@gobble` eats the (unnecessary) first comma inside `\@tempb`.

```
557   \@tempwarning{\expandafter\@gobble\@tempb}}
```

We use `\M@split@colon` and `\M@strip@colon` for parsing the argument of `\mathfont`. If the user calls `\mathfont{<name>:<features>}`, we store the name in `\@tempbase` and the

features in `\@tempfeatures`. If the user specifies a name only, then `\@tempfeatures` will be empty. Syntactically, we use `\M@strip@colon` to remove a final `:` the same way we removed a final `=` when we parsed the optional argument in the `previousnext` section.

```
558 \def\M@split@colon#1:#2\@nil{\def\@tempbase{#1}
559 \def\@tempfeatures{#2}}
560 \def\M@strip@colon#1:{#1}
```

The main font-loading macro. The command takes a single argument, which should have the form `\NFSS family` or `\font name`:`\optional features`, and it begins by parsing the argument. It splits the argument at a `:` and stores each portion in `\@tempbase` and `\@tempfeatures`. If `\@tempfeatures` is not empty, it has an extra colon at the end. We remove it. Previous versions of `mathfont` allowed users to say `\mathfont{fontspec}`, but this functionality is no longer supported.

```
561 \def\M@newfont#1{%
562 \edef\@tempa{#1}
563 \expandafter\M@split@colon\@tempa:\@nil
564 \ifx\@tempfeatures\@empty\else
565 \edef\@tempfeatures{\expandafter\M@strip@colon\@tempfeatures}
566 \fi
```

First we check if the argument is an entry in the NFSS. If yes, we check if the shapes are present using `\M@check@nfss@shapes`. We assume that if the user specifies a font family identifier here, they know what they are doing, and the font-loader makes no attempt to fill in missing shapes.

```
567 \ifcsname TU+\@tempbase\endcsname % is font family already in nfss?
568 \let\M@fontn@me\@tempbase
569 \M@check@nfss@shapes\M@fontn@me
570 \ifx\directlua\@undefined % if XeTeX
571 \expandafter\edef\csname\M@fontn@me-base\endcsname{\M@fontn@me}
572 \let\M@fontn@meb@se\M@fontn@me
573 \else % if LuaTeX
```

With `LuaTeX`, we would like to have a proper base-mode version of the font. If the font declaration happened outside of `mathfont` and the engine is `LuaTeX`, then `mathfont` expects to find another font family whose NFSS identifier is stored in `\font family-base`, and we assume this second font was loaded with `mode=base`. If that information exists, we use it for the base-mode version. Otherwise, we issue a warning if the engine is `LuaTeX`.

```
574 \ifcsname\M@fontn@me-base\endcsname % if base-mode version is known
575 \edef\M@fontn@meb@se{\csname\M@fontn@me-base\endcsname}
576 \M@check@nfss@shapes\M@fontn@meb@se
577 \else
578 \M@NoBaseModeDetectedWarning{\M@fontn@me}
579 \edef\M@fontn@meb@se{\M@fontn@me}
580 \expandafter\edef\csname\M@fontn@me-base\endcsname{\M@fontn@me}
581 \fi
582 \fi
```

Now save the font families for reference later.

```
583 \expandafter\edef\csname M@fontfamily@\@tempbase\endcsname
```

```

584     {\M@f@ntn@me}
585     \expandafter\edef\csname M@fontfamily@base@\@tempbase\endcsname
586     {\M@f@ntn@meb@se}
587     \else

```

If the argument is not a font family identifier, then we check if `mathfont` has previously loaded a font using this argument. If yes, it will be stored in `\M@fontfamily@<arg>`, and we use the font family name from the previous call to `\M@newfont`. We remove the spaces before using the argument because in general, we do not want `mathfont` to use space characters internally to distinguish font names.

```

588     \edef@nospace\@tempa{\@tempa}
589     \ifcsname M@fontfamily@\@tempa\endcsname
590     \edef\M@f@ntn@me{\csname M@fontfamily@\@tempa\endcsname}
591     \edef\M@f@ntn@meb@se{\csname M@fontfamily@base@\@tempa\endcsname}
592     \else

```

If `\M@newfont` doesn't find anything previously that matches #1, we load the font. Under the built-in font-loader, the name for the font family will be #1 with spaces removed, which we store in `\M@f@ntn@me`. (A properly declared NFSS font family does not have spaces in its name because `LATEX` ignores spaces when scanning a font family declaration.) Then we call `\M@fill@nfss@shapes` to declare all the shapes.

```

593     \edef@nospace\@tempa{\@tempa}
594     \ifcase\M@loader % are we using default font-loader?
595     \let\M@f@ntn@me\@tempa
596     \wlog{Package mathfont Info: Adding \@tempbase\space to the nfss!}
597     \wlog{Family name: \M@f@ntn@me}
598     \DeclareFontFamily{TU}{\M@f@ntn@me}{}
599     \M@fill@nfss@shapes{\M@f@ntn@me}{\M@otf@features}{\@tempfeatures}

```

If the engine is `LuaTEX`, we load a separate version of the font with `mode=base`. Then we link the base-mode and regular versions.

```

600     \ifdefined\directlua
601     \edef\M@f@ntn@meb@se{\M@f@ntn@me-base}
602     \wlog{Package mathfont Info: Adding \@tempbase with mode=base
603     to the nfss!}
604     \wlog{Family name: \M@f@ntn@meb@se}
605     \DeclareFontFamily{TU}{\M@f@ntn@meb@se}{}
606     \M@fill@nfss@shapes
607     {\M@f@ntn@meb@se}{\M@otf@features}{\@tempfeatures;mode=base}
608     \else
609     \edef\M@f@ntn@meb@se{\M@f@ntn@me}
610     \fi
611     \or % are we using fontspec as font-loader?

```

If the user requested `fontspec` as the font-loader, we pass the font name and features to `\fontspec_set_family:Nnn` for loading and store the NFSS family name in `\M@f@ntn@me`. In `LuaTEX`, we request a separate base-mode version by specifying `Renderer=Base`.

```

612     \wlog{Package mathfont Info: Passing \@tempbase\space to fontspec
613     for handling!}

```

```

614     \csname fontspec_set_family:Nnn\endcsname\M@f@ntn@me
615     {\M@otf@features,\@tempfeatures}{\@tempbase}
616     \ifdefined\directlua
617         \wlog{Package mathfont Info: Passing \@tempbase\space with
618             Renderer=Base to fontspec for handling!}
619     \csname fontspec_set_family:Nnn\endcsname\M@f@ntn@meb@se
620     {\M@otf@features,\@tempfeatures,Renderer=Base}{\@tempbase}
621     \else
622         \edef\M@f@ntn@meb@se{\M@f@ntn@me}
623     \fi
624 \fi

```

Now link the base-mode family name and store the family names for future reference.

```

625     \expandafter\edef\csname\M@f@ntn@me-base\endcsname{\M@f@ntn@meb@se}
626     \expandafter\edef\csname M@fontfamily@\@tempa\endcsname
627     {\M@f@ntn@me}
628     \expandafter\edef\csname M@fontfamily@base@\@tempa\endcsname
629     {\M@f@ntn@meb@se}
630 \fi
631 \fi}

```

Finally, the font-loading commands should appear only in the preamble.

```

632 \@onlypreamble\M@declare@shape
633 \@onlypreamble\M@fill@nfss@shapes
634 \@onlypreamble\M@newfont

```

At this point, the font information is stored in the NFSS, but nothing has been loaded. For text fonts, that happens during a call to `\selectfont`, and for math fonts, that happens the first time entering math mode. I've considered forcing some fonts to load now, but I'm hesitant to change L<sup>A</sup>T<sub>E</sub>X's standard font-loading behavior. On this issue, I plan to leave `mathfont` as is for the foreseeable future.

## 5 Parse Input

This section provides the macros to parse the optional argument of `\mathfont`. We have two parts to this section: error checking and parsing. For parsing, we extract option and suboption information, and for error checking, we make sure that both are valid. The command `\M@check@option@valid` accepts a macro containing (what is hopefully) the text of a keyword-option. The macro defines `\@temperror` to be an invalid option error and loops through all possible options. If the argument matches one of the correct possibilities, `mathfont` changes `\@temperror` to `\relax`. The macro ends by calling `\@temperror` and issuing an error if and only if the argument is invalid. If `\M@check@option@valid` finds a valid keyword-option, it changes `\if@optionpresent` to true.

```

635 \def\M@check@option@valid#1{%
636     \let\@temperror\M@InvalidOptionError % error by default
637     \@for\@j:=\M@keys\do{%
638         \ifx\@j#1
639             \let\@temperror\@gobble % eliminate error

```

```

640     \@optionpresenttrue      % set switch to true
641   \fi}
642 \def\@j{empty} % if option is "empty," we do nothing
643 \ifx\@j#1
644   \let\@temperror\@gobble
645   \@optionpresentfalse
646 \fi
647 \@temperror{#1}}

```

Do the same thing for the suboption.

```

648 \def\M@check@suboption@valid#1{%
649   \let\@temperror\M@InvalidSuboptionError % error by default
650   \@for\@j:=roman,upright,italic\do{%
651     \ifx\@j#1
652       \let\@temperror\@gobble % eliminate error
653       \@suboptionpresenttrue % set switch to true
654     \fi}
655   \@temperror{#1}}

```

Now we have to actually parse the optional argument. We want to allow the user to specify options using an `xkeyval`-type syntax. However, we do not need the full package; a slim 30 lines of code will suffice. When `\mathfont` reads one segment of *text* from its optional argument, it calls `\M@parse@option<text>=\@nil`. The `\M@parse@option` macro splits the option and suboption by looking for the first `=`. It puts its `#1` argument (hopefully the keyword-option) in `\@temp@opt` and puts `#2` (hopefully the keyword-suboption) in `\@temp@sub`. If the user specifies a suboption, `\@temp@sub` contains `<suboption>=`, and we use `\M@strip@equals` to get rid of the extra `=`. If the user does not specify a suboption, `\@temp@sub` will be empty.

```

656 \def\M@strip@equals#1=#2\@nil{%
657 \def\M@parse@option#1=#2\@nil{%
658   \@optionpresentfalse % set switch to false by default
659   \@suboptionpresentfalse % set switch to false by default
660   \def\@temp@opt{#1} % store option
661   \def\@temp@sub{#2} % store suboption

```

After storing the option in `\@temp@opt` and suboption in `\@temp@sub`, check for errors. We check for errors by determining if (1) `\@temp@opt` is empty, meaning the user did not specify an option; or (2) `\@temp@sub` is `=`, meaning the user typed `=` but did not follow it with a suboption.

```

662   \ifx\@temp@opt\@empty
663     \M@MissingOptionError
664   \else

```

Check that the user specified a valid option. We redefine `\@temp@opt` inside a group to keep the change local, and we end the group as quickly as possible after the comparison, which means separate `\egroups` in both branches of `\ifx`.

```

665     \M@check@option@valid\@temp@opt
666     \bgroup\def\@temp@opt{=}
667     \ifx\@temp@sub\@temp@opt
668       \egroup % first branch \egroup
669     \M@MissingSuboptionError

```

```

670     \else
671     \egroup % second branch \egroup
If \@temp@sub is nonempty, strip the final = and check that it contains a valid suboption.
672     \ifx\@temp@sub\@empty
673     \else
674     \edef\@temp@sub{\expandafter\M@strip@equals\@temp@sub}
675     \M@check@suboption@valid\@temp@sub % check that suboption is valid
676     \fi
677     \fi

```

If the user specified suboption `roman`, we accept it for backwards compatibility and convert it to `upright`. Again, we redefine `\@tempa` inside a group to keep the change local.

```

678     \bgroup\def\@tempa{roman}
679     \ifx\@temp@sub\@tempa
680     \egroup % first branch \egroup
681     \def\@temp@sub{upright}
682     \else
683     \egroup % second branch \egroup
684     \fi
685     \fi}

```

We code a general-purpose definition macro that defines its first argument to be the second argument fully expanded and with spaces removed.

```

686 \long\def\edef@nospace#1#2{%
687   \edef#1{#2}%
688   \edef#1{\expandafter\zap@space#1 \@empty}}

```

Perhaps something that sets spaces to `\catcode9` and then retokenizes `#2` would be better, but I don't think it matters very much.

## 6 Default Font Changes

This section documents default math font changes. The user-level font-changing command is `\mathfont`, and it feeds the font information to `\@mathfont`, the internal command that does the actual font changing. This macro is basically a wrapper around `\DeclareSymbolFont` and a bunch of calls to `\DeclareMathSymbol`, and when the user calls `\mathfont`, the command declares the user's font in the NFSS with `\M@newfont` and loops through the optional argument. On each iteration, `\@mathfont` validates the option and suboption, calls `\DeclareSymbolFont` if necessary, and sets the math codes with `\M@<keyword>@set`.

```

689 \protected\def\mathfont{\ifnextchar[{\@mathfont}{\@mathfont[\M@defaultkeys]}}

```

The internal font-changing command.

```

690 \def\@mathfont[#1]#2{%
691   \ifx\set@mathchar\@set@mathchar
692     \M@InternalsRestoredError

```

If the kernel commands have not been reset, we can do fun stuff. As of version 2.0, I'm removing the documentation for `\restoremathinternals` in the user guide, but the code will stay in for backwards compatibility.

```
693 \else
694 \M@toks{}
```

We call `\M@newfont` on the mandatory argument of `\mathfont`, which stores the two NFSS family names (one for default renderer and one for base-mode renderer if using Lua $\TeX$ ) in `\M@font@ntn@me` and `\M@font@ntn@meb@se`. If we need a new value of `\M@count`, we store it in `\M@fontid@<NFSS family name>`. We will not need a new value of `\M@count` if the user asks for the same NFSS font family twice. Throughout the definition of `\mathfont`, `\@tempa` stores the value of `\M@count` that corresponds to the current font.

```
695 \M@newfont{#2}
696 \ifcsname M@fontid@M@font@ntn@me\endcsname\else % need new \M@count value?
697 \expandafter\edef\csname M@fontid@M@font@ntn@me\endcsname{\the
698 \M@count}
699 \expandafter\edef\csname M@fontid@M@font@ntn@meb@se\endcsname{\the
700 \M@count}
701 \advance\M@count\@ne
702 \fi
703 \edef\@tempa{\csname M@fontid@M@font@ntn@me\endcsname}
```

Expand, zap spaces from, and store the optional argument in `\@tempa`, and then perform the loop. We store the current keyword-suboption pair in `\@i` and then feed it to `\M@parse@option`. We need two `\edefs` here because `\zap@space` appears before `\@tempa` in `\M@eat@spaces`. We expand the argument with the first `\edef` and remove the spaces with the second.

```
704 \edef@nospace\@tempb{#1}
705 \for\@i:=\@tempb\do{\expandafter\M@parse@option\@i=\@nil
706 \if@optionpresent
```

If the user calls `\mathfont` and tries multiple times to set the font for a certain class of characters, `mathfont` will issue a warning, and the package will not adjust the font for those characters. Notice the particularly awkward syntax with the `\csname-\endcsname` pairs. Without this construct,  $\TeX$  won't realize that `\csname if@\@tempa\endcsname` matches the eventual `\fi`, and the `\for` loop will break. ( $\TeX$  does not have a smart if-parser!)

```
707 \expandafter\ifx % next line is two cs to be compared
708 \csname ifM@\@temp@opt\expandafter\endcsname\csname iftrue\endcsname
709 \M@CharsSetWarning{\@temp@opt}
710 \else
```

The case where the current option has not had its math font set. We add the keyword-option to the `\toks`.

```
711 \edef\@tempc{\the\M@toks^^J\@temp@opt}
712 \M@toks\expandafter{\@tempc}
```

If it's present, store the suboption in `\@<option>shape` and overwrite the default definition from earlier. Then add the shape information to the `\toks` and store it in `\@tempc`. When it actually sets the font by calling `\M@<keyword>@set`, `mathfont` will determine shape information for the current character class by calling the same `\@<option>shape` macro that we store in `\@tempc`.

```
713 \if@suboptionpresent
```



```

714         \expandafter\edef\csname M@\@temp@opt shape\endcsname{\@temp@sub}
715     \fi
716     \edef\@tempc{\the\M@toks\space
717         (\csname M@\@temp@opt shape\endcsname)}
718     \M@toks\expandafter{\@tempc}
719     \edef\@tempc{\csname M@\@temp@opt shape\endcsname}

```

We store the font shape information in \@tempb, specifically \@tempb will be the default NFSS shape code corresponding to the current suboption. At this point, \@tempc is either “upright” or “italic,” so we temporarily let \@tempb be the string “upright” and check if it equals \@tempc. We redefine \@tempb depending on the results.

```

720     \def\@tempb{upright}
721     \ifx\@tempb\@tempc
722         \let\@tempb\shapedefault
723     \else
724         \let\@tempb\itdefault
725     \fi

```

At this point we have the information we need to declare the symbol font: the NFSS family (\M@fontname), series (\mddefault), and shape (\@tempb) information. The symbol font name will be M<suboption><value of \M@count>. We check if the symbol font we need for the current set of characters is defined, and if not, we define it using this information.

```

726     \ifcsname symM@\@tempc\@tempa\endcsname\else
727         \M@SymbolFontInfo{\@tempbase}{\M@fontname@se}
728         {\mddefault/\@tempb}{M@\@tempc\@tempa}
729         \DeclareSymbolFont
730         {M@\@tempc\@tempa}{TU}{\M@fontname@se}{\mddefault}{\@tempb}
731     \fi

```

We store the new font information so we can write it to the log file \AtBeginDocument and send an informational message to the user.

```

732     \expandafter
733     \edef\csname M@\@temp@opt @fontinfo\endcsname{\@tempbase}
734     \M@FontChangeInfo{\@temp@opt}{\@tempbase}

```

And now the magic happens!

```

735     \csname M@\@temp@opt @set\endcsname % set default font
736     \csname M@\@temp@opt true\endcsname % set switch to true
737     \fi
738     \fi}

```

Display concluding messages for the user.

```

739     \edef\@tempa{\the\M@toks}
740     \ifx\@tempa\@empty
741         \wlog{The \string\mathfont\space command on line
742             \the\inputlineno\space did not change the font for any characters!}
743     \else
744         \wlog{}
745         \typeout{:: mathfont :: Using font \@tempbase\space
746             on line \the\inputlineno.}

```

```

747     \wlog{Character classes changed:\the\M@toks}
748   \fi
749   \fi}
750 \@onlypreamble\mathfont
751 \@onlypreamble\m@thf@nt
752 \@onlypreamble\@mathfont

```

The `\setfont` command will call `\mathfont` and set the text font.

```

753 \protected\def\setfont#1{%
754   \mathfont{#1}
755   \mathconstantsfont{#1}
756   \setmathfontcommands{#1}
757   \let\rmdefault\M@font@ntn@me}
758 \@onlypreamble\setfont

```

The macro `\mathconstantsfont` chooses a font for setting math parameters. It is intended for Lua $\TeX$  when `mathfont` can adjust text fonts and add a `MathConstants` table. It issues a warning if called without font adjustments enabled. First, it checks if the argument was previously fed to `\mathfont` by seeing whether `\M@fontfamily@<#1>` is equal to `\relax`. If yes, `#1` was never an argument of `\mathfont`, and we raise an error.

```

759 \let\M@SetMathConstants\relax
760 \protected\def\mathconstantsfont{%
761   \ifnextchar[{\@mathconstantsfont}{\@mathconstantsfont[upright]}}
762 \def\@mathconstantsfont[#1]#2{%
763   \edef\nospace\@tempa{#2}
764   \edef\@tempa{\csname M@fontfamily@base@\@tempa\endcsname}
765   \expandafter\ifx\@tempa\relax
766     \M@BadMathConstantsFontError{#2}
767   \else

```

Some error checking. If `#1` isn't "upright" or "italic," we should raise an error. If the `\@tempa` font doesn't correspond to a symbol font, we declare it. Before defining `\M@SetMathConstants` if necessary, we store the NFSS family name in `\m@th@const@nts@font`.

```

768   \def\@tempb{#1}
769   \def\@tempc{upright}
770   \ifx\@tempb\@tempc
771     \let\m@th@const@nts@font@sh@pe\shapedefault
772   \else
773     \def\@tempc{italic}
774     \ifx\@tempb\@tempc
775       \let\m@th@const@nts@font@sh@pe\itdefault
776     \else
777       \M@BadMathConstantsFontTypeError{#1}
778     \fi
779   \fi
780   \ifcsname symM#1\csname M@fontid@\@tempa\endcsname\endcsname\else
781     \DeclareSymbolFont{M#1\csname M@fontid@\@tempa\endcsname}
782     {TU}{\@tempa}{\mddefault}{\m@th@const@nts@font@sh@pe}
783   \fi

```

```
784 \let\math@const@nts@font\@tempa
```

We come to the tricky problem of making sure to use the correct MathConstants table. LuaTeX automatically initializes all math parameters based on the most recent `\textfont`, etc. assignment, so we want to tell L<sup>A</sup>T<sub>E</sub>X to reassign whatever default font we’re using to the correct math family whenever we load new math fonts. This is possible, but the implementation is super hacky. When L<sup>A</sup>T<sub>E</sub>X enters math mode, it checks whether it needs to redo any math family assignments, typically because of a change in font size, and if so, it calls `\getanddefine@fonts` repeatedly to append `\textfont`, etc. assignments onto the macro `\math@fonts`. Usually `\math@fonts` is empty because this process always happens inside a group, so we can hook into the code by defining `\math@font` to be `\aftergroup<extra code>`. In this case, the *extra code* will be another call to `\getanddefine@fonts`.

We initialize `\M@SetMathConstants` to be `\relax`, so we define it the first time the user calls `\mathconstantsfont`. The command calls `\getanddefine@fonts` inside a group and uses as arguments the upright face of the font corresponding to #1. Then we call `\math@fonts`, and to avoid an infinite loop, we gobble the `\aftergroup\M@SetMathConstants` macros that `mathfont` has inserted at the start of `\math@fonts`. Setting `\globaldefs` to 1 makes the `\textfont`, etc. assignments from `\getanddefine@fonts` global when we call `\math@fonts`.

```
785 \protected\def\M@SetMathConstants{%
786   \begingroup
787   \escapechar\m@ne
788   \expandafter\getanddefine@fonts
789   \csname symM#1\csname M@fontid@\m@th@const@nts@font\endcsname
790   \expandafter
791   \endcsname % expands to \symMupright<id>
792   \csname TU/\m@th@const@nts@font
793     /\seriesdefault
794     /\m@th@const@nts@font@sh@pe
795   \endcsname % expands to \TU/<nfss family name>/m/<shape>
796   \globaldefs\@ne
797   \expandafter\@gobbletwo\math@fonts % gobble to avoid infinite loop
798   \endgroup}
799 \fi
800 \ifM@adjust@font\else
801   \M@MathConstantsNoFontAdjustWarning
802 \fi}
803 \def\math@fonts{\aftergroup\M@SetMathConstants}
804 \@onlypreamble\mathconstantsfont
```

If the user has not enabled Lua font adjustments, then `\mathconstantsfont` will generate an error message and gobble its argument. This definition happens later in `mathfont.sty` when we define other Lua-related macros such as `\IntegralItalicFactor` to do the same thing absent font adjustments.

## 7 Local Font Changes

This section deals with local font changes. The `\newmathfontcommand` creates macros that change the font for math alphabet characters and is basically a wrapper around `\DeclareMathAlphabet`. First we code `\M@check@csarg`, which accepts two arguments. The `#1` argument is the user-level command that called `\M@check@csarg`, which we use for error messaging, and `#2` should be a single control sequence. The way `\M@check@csarg` scans the following tokens is a bit tricky: (1) check the length of the argument by seeing if `\@gobble` eats it completely; and (2) check that the argument is a control sequence. If the user specifies an argument of the form `{. .}`, i.e. extra text inside braces, the `\ifcat` will catch it and issue an error. If `\M@check@csarg` likes the input, it sets `\ifM@arg@good` to true, and otherwise, it sets `\ifM@arg@good` to false.

```

805 \def\M@check@csarg#1#2{%
806   \expandafter\ifx\expandafter\@nnil\@gobble#2\@nnil % good
807   \ifcat\relax\noexpand#2 % good
808     \M@arg@goodtrue
809   \else % if #2 not a control sequence
810     \M@MissingControlSequenceError#1{#2}
811     \M@arg@goodfalse
812   \fi
813 \else % if #2 is multiple tokens
814   \M@DoubleArgError#1{#2}
815   \M@arg@goodfalse
816 \fi}

```

Now declare the math alphabet. This macro first checks that its `#1` argument is a control sequence using `\M@check@csarg`. If yes, we feed the `#2` argument to `\M@newfont` for loading, print a message in the log file, and call `\DeclareMathAlphabet`.

```

817 \protected\def\newmathfontcommand#1#2#3#4{%
818   \M@check@csarg\newmathfontcommand{#1}
819   \ifM@arg@good
820     \M@newfont{#2}
821     \M@NewFontCommandInfo{#1}{\@tempbase}{\M@f@ntn@m@b@se}{#3}{#4}
822     \DeclareMathAlphabet{#1}{TU}{\M@f@ntn@m@b@se}{#3}{#4}
823   \fi}
824 \@onlypreamble\newmathfontcommand

```

Then define macros that create local font-changing commands with default series and shape information. Because they're all so similar, we metacode them. We define the commands themselves with `\define@newmath@cmd`. The argument structure is:

- `#1`—`\newmath<key>` macro name
- `#2`—font series
- `#3`—font shape
- `##1`—the user's control sequence
- `##2`—the user's font information (family name)

We feed `##1`, `##2`, `#2`, and `#3` to `\newmathfontcommand`, and we load `##2` with `\M@newfont`. Each `\newmath<key>` macro will check its first argument using `\M@check@csarg` and

then call `\newmathfontcommand` on both of its two arguments. We store the list of `\newmath⟨key⟩` commands that we want to define with their series and shape information in `\M@default@newmath@cmds`, and we loop through it with `\@for`.

```

825 \def\M@define@newmath@cmd#1#2#3{%
826   \protected\def#1##1##2{%
827     \M@check@csarg{#1}{##1}
828     \newmathfontcommand{##1}{##2}{#2}{#3}}
829 \def\M@default@newmath@cmds{%
830   \newmathrm{\mddefault}{\shapedefault},%
831   \newmathit{\mddefault}{\itdefault},%
832   \newmathbf{\bfdefault}{\shapedefault},%
833   \newmathbfit{\bfdefault}{\itdefault},%
834   \newmathsc{\mddefault}{\scdefault},%
835   \newmathscit{\mddefault}{\scdefault\itdefault},%
836   \newmathbfsc{\bfdefault}{\scdefault},%
837   \newmathbfscit{\bfdefault}{\scdefault\itdefault}}
838 \@for\@i:=\M@default@newmath@cmds\do{\expandafter\M@define@newmath@cmd\@i}
839 \@onlypreamble\newmathrm
840 \@onlypreamble\newmathit
841 \@onlypreamble\newmathbf
842 \@onlypreamble\newmathbfit
843 \@onlypreamble\newmathsc
844 \@onlypreamble\newmathscit
845 \@onlypreamble\newmathbfsc
846 \@onlypreamble\newmathbfscit
847 \@onlypreamble\M@define@newmath@cmd
848 \let\M@default@newmath@cmds\relax

```

The command `\setmathfontcommands` sets all the default local font-change commands at once.

```

849 \protected\def\setmathfontcommands#1{%
850   \newmathrm\mathrm{#1}
851   \newmathit\mathit{#1}
852   \newmathbf\mathbf{#1}
853   \newmathbfit\mathbfit{#1}
854   \newmathsc\mathsc{#1}
855   \newmathscit\mathscit{#1}
856   \newmathbfsc\mathbfsc{#1}
857   \newmathbfscit\mathbfscit{#1}}
858 \@onlypreamble\setmathfontcommands

```

We provide `\newmathbold` and `\newmathboldit` for backwards compatibility but issue a warning.

```

859 \protected\def\newmathbold{%
860   \M@DeprecatedWarning\newmathbold\newmathbf\newmathbf}
861 \protected\def\newmathboldit{%
862   \M@DeprecatedWarning\newmathboldit\newmathbfit\newmathbfit}

```

## 8 Miscellaneous Material

We begin this section with the user-level macros that provide information for Lua-based font adjustments. If font adjustments are allowed, we begin with a macro `\M@check@int` that passes the user's argument to Lua and determines whether it is an integer. We check whether the argument contains a backslash or quote mark similar to error checking later in `\CharmLine`. Depending on the result, `mathfont` sets `\ifM@arg@good` to true or false.

```

863 \ifM@adjust@font
864 \def\M@check@int#1{%
865 \M@arg@goodfalse
866 \begingroup
867 \edef\@tempa{\number0#1}
868 \edef\@tempa{\detokenize\expandafter{\@tempa}}
869 \@expandtwoargs\in@{"}{\@tempa}

If #1 contains a " or backslash, we set \M@arg@good to false and stop parsing the argument.
870 \ifin@ % is " in #1?
871 \endgroup % first branch \endgroup
872 \else
873 \@expandtwoargs\in@{\@backslashchar}{\@tempa}
874 \ifin@ % is backslash in #1?
875 \endgroup % second branch \endgroup
876 \else
877 \directlua{
878 local num = tonumber("\@tempa")
879 local bool = 0 % keep track if \@tempa is (int >= 0)
880 if num then % if number?
881 if num == num - (num \@percentchar 1) then % if integer?
882 if num >= 0 then % if nonnegative?
883 bool = 1
884 end
885 end
886 end
887 tex.print("\@backslashchar\@backslashchar endgroup")
888 if bool == 1 then
889 tex.print("\@backslashchar\@backslashchar csname M@arg@goodtrue%
890 \@backslashchar\@backslashchar endcsname")
891 end}
892 \fi
893 \fi}

```

We meta-code the definitions of `\RuleThicknessFactor`, etc. To keep the syntax relatively clean, we temporarily eliminate the `\escapechar` and redefine `~` to `\noexpand`.

```

894 \let\@tempa~
895 \let~\noexpand
896 \count@\escapechar
897 \escapechar\m@ne
898 \@tfor\@i:=\RuleThicknessFactor\IntegralItalicFactor

```

```

899   \SurdHorizontalFactor\SurdVerticalFactor\do{%
900     \protected\expandafter\edef\@i#1{%
901       ~\M@check@int{#1}%
902       ~\ifM@arg@good
903         ~\global
904           \expandafter~\csname M@\expandafter\string\@i\endcsname=#1\relax
905       ~\else
906         ~\M@BadIntegerError\expandafter~\@i{#1}%
907       ~\fi}}
908 \let~\@tempa
909 \escapechar\count@

```

If automatic font adjustments are disabled, we should also disable the related user-level commands. In this case, each of the font-adjustment macros expands to raise an `\M@NoFontAdjustError` and gobble its argument.

```

910 \else
911   \@tfor\@i:=\RuleThicknessFactor\IntegralItalicFactor\SurdHorizontalFactor
912     \SurdVerticalFactor\CharmLine\CharmFile
913     \do{%
914       \protected\expandafter\edef\@i{\noexpand\M@NoFontAdjustError
915         \expandafter\noexpand\@i
916         \noexpand@gobble}}
917 \fi

```

These commands should appear in the preamble only.

```

918 \@onlypreamble\RuleThicknessFactor
919 \@onlypreamble\IntegralItalicFactor
920 \@onlypreamble\SurdHorizontalFactor
921 \@onlypreamble\SurdVerticalFactor
922 \@onlypreamble\CharmLine
923 \@onlypreamble\CharmFile

```

Provide the command to reset the kernel. I am not sure that we need this macro, but it will stay in the package for backwards compatibility.

```

924 \def\restoremathinternals{%
925   \ifx\set@mathchar\@@set@mathchar
926   \else
927     \wlog{Package mathfont Info: Restoring \string\set@mathchar.}
928     \wlog{Package mathfont Info: Restoring \string\set@mathsymbol.}
929     \wlog{Package mathfont Info: Restoring \string\set@mathaccent.}
930     \wlog{Package mathfont Info: Restoring \string\DeclareSymbolFont.}
931     \let\set@mathchar\@@set@mathchar
932     \let\set@mathsymbol\@@set@mathsymbol
933     \let\set@mathaccent\@@set@mathaccent
934     \let\DeclareSymbolFont\@@DeclareSymbolFont
935   \fi}

```

Three macros used in defining `\simeq` and `\cong`. The construction is clunky and needs the intermediate macro `\st@ck@fl@trel` because `\mathchoice` is a bit of an odd macro. Instead of expanding to different replacement text depending on the math style, it fully typesets each

of its four arguments and then takes the one corresponding to the correct style. A cleaner implementation would use `\mathstyle` from Lua $\TeX$ —perhaps in a future version.

```

936 \protected\gdef\clap#1{\hb@xt@\z@{\hss#1\hss}}
937 \protected\def\stack@flatrel#1#2{\expandafter
938   \st@ck@fl@trel\expandafter#1\@firstofone#2}
939 \protected\gdef\st@ck@fl@trel#1#2#3{%
940   {\setbox0\hbox{\m@th#1#2$}% contains \mathrel symbol
941   \setbox1\hbox{\m@th#1#3$}% gets raised over \box0
942   \if\wd0>\wd1\relax
943     \hb@xt@\wd0{%
944       \hfil
945       \clap{\raise0.7\ht0\box1}%
946       \clap{\box0}\hfil}%
947   \else
948     \hb@xt@\wd1{%
949       \hfil
950       \clap{\raise0.7\ht0\box1}%
951       \clap{\box0}\hfil}%
952   \fi}}

```

Some fonts do not contain characters that `mathfont` can declare as math symbols. We want to make sure that if this happens,  $\TeX$  prints a message in the log file and terminal.

```

953 \ifnum\tracinglostchars<\tw@
954   \tracinglostchars\tw@
955 \fi

```

Write to the log file `\AtBeginDocument` all font changes carried out by `mathfont`. The command `\keyword@info@begindocument` accepts two arguments and is what actually prints the informational message after the preamble. One argument is a keyword-argument from `\mathfont`, and the other is a number of spaces. The spaces make the messages line up with each other in the log file.

```

956 \def\keyword@info@begindocument#1:#2\@nil{%
957   \expandafter\ifx % next line is two cs to be compared
958     \csname ifM@#1\expandafter\endcsname\csname iftrue\endcsname
959     \wlog{#1:#2@spaces Set to
960       \csname M@#1@fontinfo\endcsname,
961       \csname M@#1shape\endcsname\space shape.}
962   \else
963     \wlog{#1:#2@spaces No change.}
964   \fi}

```

Now print the messages.

```

965 \AtBeginDocument{%
966   \def\@tempa{%    <-- everything should be 14 characters long
967     upper:\@spaces\@spaces,%
968     lower:\@spaces\@spaces,%
969     diacritics:\space\space\space,%
970     greekupper:\space\space\space,%
971     greeklower:\space\space\space,%

```



```

972   agreekupper:\space\space,%
973   agreeklower:\space\space,%
974   cyrillicupper:,%
975   cyrilliclower:,%
976   hebrew:\@spaces\space\space\space,%
977   digits:\@spaces\space\space\space,%
978   operator:\@spaces\space,%
979   delimiters:\space\space\space,%
980   radical:\@spaces\space\space,%
981   bigops:\@spaces\space\space\space,%
982   extbigops:\@spaces,%
983   symbols:\@spaces\space\space,%
984   extsymbols:\space\space\space,%
985   arrows:\@spaces\space\space\space,%
986   bb:\@spaces\@spaces\space\space\space,%
987   cal:\@spaces\@spaces\space\space,%
988   frak:\@spaces\@spaces\space,%
989   bcal:\@spaces\@spaces\space,%
990   bfrak:\@spaces\@spaces}
991   \wlog{^^JPackage mathfont Info: List of changes made in the preamble.}
992   \@for\@i:=\@tempa\do{%
993     \expandafter\keyword@info@begindocument\@i\@nil}
994   \wlog{}}

```

Warn the user about possible problems with a multi-word optional package argument in  $\XeTeX$ .

```

995 \ifdefined\XeTeXrevision
996   \ifM@font@loaded
997     \AtEndOfPackage{%
998       \PackageWarningNoLine{mathfont}
999       {XeTeX detected. It looks like you\MessageBreak
1000       specified a font when you loaded\MessageBreak
1001       mathfont. If you run into problems\MessageBreak
1002       with a font whose name is multiple\MessageBreak
1003       words, try compiling with LuaLaTeX\MessageBreak
1004       or call \string\setfont\space or \string\mathfont\MessageBreak
1005       manually}}
1006   \fi
1007 \fi

```

If the user passed a font name to `mathfont`, we set it as the default `\AtEndOfPackage`.

```

1008 \ifM@font@loaded
1009   \AtEndOfPackage{\setfont\M@font@load}
1010 \fi

```

Finally, make all character-setting commands inaccessible outside the preamble.

```

1011 \@onlypreamble\M@upper@set
1012 \@onlypreamble\M@lower@set
1013 \@onlypreamble\M@diacritics@set

```

```

1014 \@onlypreamble\M@greekupper@set
1015 \@onlypreamble\M@greeklower@set
1016 \@onlypreamble\M@agreekupper@set
1017 \@onlypreamble\M@agreeklower@set
1018 \@onlypreamble\M@cyrillicupper@set
1019 \@onlypreamble\M@cyrilliclower@set
1020 \@onlypreamble\M@hebrew@set
1021 \@onlypreamble\M@digits@set
1022 \@onlypreamble\M@operator@set
1023 \@onlypreamble\M@delimiters@set
1024 \@onlypreamble\M@radical@set
1025 \@onlypreamble\M@bigops@set
1026 \@onlypreamble\M@extbigops@set
1027 \@onlypreamble\M@symbols@set
1028 \@onlypreamble\M@extsymbols@set
1029 \@onlypreamble\M@arrows@set
1030 \@onlypreamble\M@bb@set
1031 \@onlypreamble\M@cal@set
1032 \@onlypreamble\M@frak@set
1033 \@onlypreamble\M@bcal@set
1034 \@onlypreamble\M@bfrak@set

```

## 9 Adjust Fonts: Setup

The next three sections implement Lua-based font adjustments and apply only if the user has enabled font adjustment. Most of the implementation happens through Lua code, but we need some  $\TeX$  code in case the user wants to adjust character metric information. Here is a rough outline of what happens in the next three sections:

1. Initialize a Lua table that contains new metrics for certain characters specific to math mode, such as letters with wider bounding boxes and large operator symbols.
2. Provide an interface for the user to change this metric information.
3. Write functions that accept a fontdata object and (a) change top-level math specs to indicate that we have a math function; (b) alter characters according to our Lua table of new metric information; and (c) populate a `MathConstants` table for the font.
4. Create callbacks that call these functions. Insert them into `luaotfload.patch_font`.

Step 2 happens on the  $\TeX$  side of things and is documented next, and everything else happens inside `\directlua`. On the Lua side of things, we store all the functions and character metric information in the table `mathfont`. Every entry in `mathfont` is a function or is a subtable indexed within `mathfont` by an  $\langle integer \rangle$ . The *integer* is a unicode encoding number and tells which unicode character the subtable keeps track of. See tables 2 and 3 for a list of the functions in `mathfont` and the fields in character subtables. See section 11 for discussion of the callbacks for editing fontdata objects.

Changing top-level flags in a font object is straightforward. Creating a `MathConstants` table is complicated but largely self-contained. We take a few parameters that the user has set, define traditional  $\TeX$  math parameters based on the essential parameters of the font,

**Table 2: Fields of Character Subtables in `mathfont`**

Field	Data Type	In a?	In e?	In u?	Used For
<code>type</code>	string	Yes	Yes	Yes	Tells if type a, e, u
<code>next</code>	depends	Yes	Yes	No	Unicode index of next-larger character(s); integer for type a, table for type u
<code>left_stretch</code>	numeric	Yes	No	No	Stretch bounding box left
<code>right_stretch</code>	numeric	Yes	No	No	Stretch bounding box right
<code>top_accent_stretch</code>	numeric	Yes	Yes	Yes	Position top accent
<code>bot_accent_stretch</code>	numeric	Yes	Yes	Yes	Position bottom accent
<code>total_variants</code>	integer	No	Yes	No	Number of large variants
<code>smash</code>	integer	No	Yes	No	Unicode index for storing a smashed version
<code>data</code>	table	No	Yes	No	Scale factors

and assign their values to corresponding entries in a `MathConstants` table. However, editing character metrics is convoluted with many moving parts. For every glyph that we want modify when  $\TeX$  loads a text font, we store character metric information about that glyph as a subtable in `mathfont`. The entries of the subtable describe how to stretch the glyph bounds, scale the glyph itself, or determine math accent placement. For characters of type `u`, we only specify accent placement. For characters of type `a`, which is the upper and lower-case Latin letters, we stretch the bounding box of the glyph horizontally to widen the letters slightly. When we load a text font, we create 52 virtual characters in the Unicode Supplementary Private Use Area-A that typeset the Latin letter glyphs in elongated bounding boxes, and later in `mathfont`, we set the mathcodes of Latin letters to be these virtual characters. For type `e`, we do the same thing except that for each character, we create an ensemble of scaled versions, which we use as a family of large variants.

Here's how to think about the dynamics of our approach. We use character metric information at three different times: pre-processing, interim processing, and post-processing. In pre-processing, which we implement in this section, we assemble initial character metric information into entries in `mathfont`. In other words, pre-processing means creating the initial `mathfont` subtables and happens during package loading. Interim processing means the user altering entries in `mathfont` and happens through `\CharmLine` and `\CharmFile`. This can occur at any point in the preamble. In post-processing, which we implement in the next section, `mathfont` extracts information from the current state of the `mathfont` table and uses it to alter a `fontdata` object. Post-processing happens through the `luaotfload.patch_font` callback and occurs once at the point when  $\TeX$  loads the font file. As a rule,  $\LaTeX$  does not like to load fonts before it uses them, so post-processing typically happens `\AtBeginDocument` in the case of the main text font or whenever the user calls a `\text{font keyword}` command or enters math mode, whichever happens first. This is also why you cannot adjust fonts that  $\TeX$  loaded before `mathfont`.

We set `mathnolimitsmode` to 4 to make integral signs look nice. Or at least nicer than they would otherwise.

**Table 3: Functions in mathfont**

Function	Argument(s)	Used For
<code>new_type_a</code>	<code>index, next, data</code>	Add type a entry to mathfont
<code>new_type_e</code>	<code>index, smash, next, data</code>	Add type e entry to mathfont
<code>new_type_u</code>	<code>index, smash, next, data</code>	Add type u entry to mathfont
<code>add_to_charm</code>	string of new charm info	Add new charm into to mathfont
<code>parse_charm</code>	string of new charm info	Split the string, validate inputs
<code>empty</code>	none	Does nothing
<code>glyph_info</code>	character subtable	Return height, width, depth, italic
<code>make_a_commands</code>	<code>index, offset</code>	Return virtual font commands
<code>make_a_table</code>	<code>index, charm data, fontdata</code>	Make new character table for type a
<code>make_e_commands</code>	<code>index, scale factors</code>	Return virtual font commands
<code>make_e_table</code>	<code>index, charm data, fontdata</code>	Make new character table for type e
<code>make_hex_value</code>	integer	Return hexadecimal string
<code>make_u_commands</code>	<code>index, offset</code>	Return virtual font commands
<code>make_u_table</code>	<code>index, charm data, fontdata</code>	Make new character table for type u
<code>modify_e_base</code>	<code>index, offset</code>	Modify base glyph for type e
<code>smash_glyph</code>	<code>index, fontdata</code>	Return table for smashed character
<code>adjust_font</code>	fontdata	Call callbacks
<code>apply_charm_info</code>	fontdata	Change character metrics in fontdata
<code>get_font_name</code>	fontdata	Return font name
<code>info</code>	string	Writes a message in the log file
<code>math_constants</code>	fontdata	Creates a MathConstants table
<code>set_nomath_true</code>	fontdata	Set top-level font specs for math

```
1035 \ifM@adjust@font
```

```
1036 \mathnolimitsmode=4\relax
```

We need some error messages. We change the catcode of `\` to 12 in order to use it freely as a Lua escape character. We change `~` to catcode 0 to define the macros.

```
1037 \bgroup
```

```
1038 \catcode`\~=0
```

```
1039 ~catcode`~\=12
```

```
1040 ~@firstofone{
```

```
1041 ~egroup
```

```
1042 ~def~M@number@ssert{"\n%
```

```
1043 Package mathfont error: Nonnumeric charm value.\n\n%
```

```
1044 I'm having trouble with a character metric.\n%
```

```
1045 Your \\CharmLine or \\CharmFile contains \"\"..temp_string..\"\".\n%
```

```
1046 which is not a number. Make sure that your\n%
```

```
1047 charm information is all integers, floats,\n%
```

```
1048 or asterisks separated by commas or spaces.\n"}
```

```
1049 ~def~M@index@ssert{"\n%
```

```
1050 Package mathfont error: Invalid unicode index.\n\n%
```

```

1051 The unicode index \'"'.split_string[1]..'\" is invalid. Make sure\n%
1052 that the first number in your \\CharmLine and in each\n%
1053 line of your \\CharmFile is an integer between 0 and\n%
1054 1,114,111.\n"}
1055 ~def~M@entries@assert{"\n%
1056 Package mathfont error: Charm values too short.\n\n%
1057 Your charm information for U+"..index.." needs more\n%
1058 entries. Right now you have "..number_of_entries.." entries, and\n%
1059 you need at least "..entries_needed..". If you aren't sure what\n%
1060 to do, try adding asterisks to your \\CharmLine\n%
1061 or line in your \\CharmFile.\n"}}}

```

The user inputs charm information at the  $\TeX$  level. We define the macros `\CharmLine` that interfaces with `mathfont:add_to_charm` directly and `\CharmFile` that reads lines from a file and individually feeds them to `\CharmLine`. For `\CharmLine`, we check that the argument contains no " or \ symbols because that could mess up the Lua parsing.

```

1062 \protected\def\CharmLine#1{%
1063   \begingroup
1064   \edef\@tempa{#1}
1065   \edef\@tempa{\detokenize\expandafter{\@tempa}}
1066   \@expandtwoargs\in@{"}{\@tempa}

```

If #1 contains a ", we issue an error. The error help message is different depending on whether the `\CharmLine` came from a call to `\CharmFile` or not, which we check with `\ifM@fromCharmFile`.

```

1067   \ifin@ % is " in #1?
1068     \ifM@fromCharmFile
1069       \M@ForbiddenCharmFile{"}
1070     \else
1071       \M@ForbiddenCharmLine{"}
1072     \fi
1073   \else
1074     \@expandtwoargs\in@{\@backslashchar}{\@tempa}
1075     \ifin@ % is backslash in #1?
1076       \ifM@fromCharmFile
1077         \M@ForbiddenCharmFile{\@backslashchar}
1078       \else
1079         \M@ForbiddenCharmLine{\@backslashchar}
1080       \fi
1081     \else

```

If #1 does not contain a quotation mark or escape char, we feed it to `mathfont:add_to_charm` as a string.

```

1082       \directlua{mathfont:add_to_charm("\@tempa")}
1083     \fi
1084   \fi
1085 \endgroup}

```

The argument of `\CharmFile` should be a valid filename, and we open it in `\M@Charm`. The `\M@fromCharmFiletrue` command sets the boolean for an open charm file to true. This

command and the corresponding false command are global because of how the kernel defines `\newif`. We can't check `\ifeof\M@Charm` because during processing of the last line from `\M@Charm`, we are at the end of the file even though it is still open.

```
1086 \protected\def\CharmFile#1{%
1087   \begingroup
1088   \M@fromCharmFiletrue
1089   \immediate\openin\M@Charm{#1}
```

The macro `\@next` will read a line in `#1`, feed it to `\CharmLine`, and call itself if the file has more lines.

```
1090   \def\@next{%
1091     \read\M@Charm to \@tempa
1092     \CharmLine\@tempa
1093     \ifeof\M@Charm\else % if file has more lines?
1094       \expandafter\@next
1095     \fi}
```

Call `\@next`, close the file, and end the group.

```
1096   \@next
1097   \immediate\closein\M@Charm
1098   \M@fromCharmFilefalse
1099   \endgroup}
```

This concludes the  $\TeX$ -based portion of font adjustments. The rest of this section and the next two sections are the Lua code that adapts a text font for math mode. First, we create the `mathfont` table.

```
1100 \directlua{
1101 mathfont = {}
```

Each character whose metrics we want to change will have one of three types: `a` for alphabet, `e` for extensible, and `u` for (other) unicode. (We don't really create extensibles—type `e` means any character that we need artificially larger sizes for.) We begin with type `a`. The `index` is the base-10 unicode value of the character that we will later modify, and `next` is the base-10 unicode value of the slot we will use to store the modified glyph. The `data` variable is a table with 4 entries that stores sizing information and information regarding accent placement. We divide the information by 1000 as is standard in  $\TeX$ .

```
1102 function mathfont:new_type_a(index, next, data)
1103   self[index] = {}
1104   self[index].type = "a"
1105   self[index].next = next
1106   self[index].left_stretch = data[1] / 1000
1107   self[index].right_stretch = data[2] / 1000
1108   self[index].top_accent_stretch = data[3] / 1000
1109   self[index].bot_accent_stretch = data[4] / 1000
1110 end
```

Initializing type `e` characters is more complicated. The `index` argument is the base-10 unicode value of the character we will modify. The `smash` value is a unicode slot where we will store a smashed version of the glyph with no height, depth, or width, which we need to scale the glyph correctly. We use `next` and `data` to add large variants of characters to the font.

Specifically, `next` is a table of unicode slots where we will add larger versions of the character with unicode value `index`, and `data` stores the sizing information.

```
1111 function mathfont:new_type_e(index, smash, next, data)
```

We determine the number of larger variants `v` from the length of `next`, and we store that number in `total_variants`.

```
1112 local v = \string# next
1113 self[index] = {}
1114 self[index].type = "e"
1115 self[index].smash = smash
1116 self[index].next = next
1117 self[index].total_variants = v
1118 self[index].data = {}
```

We expect `data` to have  $2v + 2$  entries, which we consider in pairs. The  $i$ th pair (i.e. entries  $i$  and  $i + 1$  of `data`) encodes the horizontal and vertical scale factors for the  $i$ th large variant, and the final two entries determine top and bottom accent placement. We store each pair as a two-element table in the larger table `mathfont[index].data`, and we use `x` and `y` as the keys for the horizontal and vertical stretch. Again we divide both scale factors by 1000.

```
1119 for i = 1, v, 1 do
1120   self[index].data[i] = {}
1121   self[index].data[i].x = data[2*i-1] / 1000
1122   self[index].data[i].y = data[2*i] / 1000
1123 end
1124 self[index].top_accent_stretch = data[2*v+1] / 1000
1125 self[index].bot_accent_stretch = data[2*v+2] / 1000
1126 end
```

The type `u` characters are simplest. We need to specify the unicode index in the first argument. The second function argument is a table with two entries that stores accent information.

```
1127 function mathfont:new_type_u(index, data)
1128 self[index] = {}
1129 self[index].type = "u"
1130 self[index].top_accent_stretch = data[1] / 1000
1131 self[index].bot_accent_stretch = data[2] / 1000
1132 end
```

Interim processing. We provide a way for the user to edit resizing and accent information for the characters in `mathfont`. The function `mathfont.parse_charm` parses and validates the user's input, and the function `mathfont:add_to_charm` incorporates the user's information into the tables already in `mathfont`. The `mathfont:add_to_charm` function expects a single string of integers, floats, or asterisks separated by spaces or commas and immediately passes it to `parse_charm`. Our first task is to split the string into components, and we store the results in `split_string`. The dummy variable `i` keeps track of the number of entries currently in `split_string`.

```
1133 function mathfont.parse_charm(charm_input)
1134 local split_string = {}
1135 local charm_string = charm_input
```

```

1136 local temp_string = ""
1137 local i = 1

```

We loop through `charm_string` as long as it contains a comma or space. At each iteration, we remove the portion of `charm_string` preceding the first comma or space and append it to `split_string` as a separate entry.

```

1138 while string.find(charm_string, " ") or string.find(charm_string, ",") do
1139     local length = string.len(charm_string)
1140     local first_space = string.find(charm_string, " ") or length
1141     local first_comma = string.find(charm_string, ",") or length

```

We store the location of the first comma or space in `sep`.

```

1142     local sep = first_space
1143     if first_comma < first_space then
1144         sep = first_comma
1145     end

```

Now split `charm_string` at `sep`. We store the portion before `sep` in `temp_string`, and the portion after `sep` becomes the new `charm_string`.

```

1146     temp_string = string.sub(charm_string, 1, sep-1)
1147     charm_string = string.sub(charm_string, sep+1)

```

If `temp_string` is not empty, we store it in position `i` in `split_string`, then increment `i` by 1. If `temp_string` does not contain a number or asterisk, we raise an error.

```

1148     if temp_string \noexpand~= "" then
1149         if tonumber(temp_string) then % if a number, append number
1150             split_string[i] = tonumber(temp_string)
1151             i = i+1
1152         elseif temp_string == "*" then % if asterisk, append asterisk
1153             split_string[i] = temp_string
1154             i = i+1
1155         else % if neither, raise error
1156             error(\M@number@assert)
1157         end
1158     end
1159 end

```

After we iterate the splitting procedure, we have a final portion of `charm_string` with no commas or spaces, and we perform the same check as on `temp_string` above.

```

1160 temp_string = charm_string
1161 if temp_string \noexpand~= "" then
1162     if tonumber(temp_string) then % if a number, append number
1163         split_string[i] = tonumber(temp_string)
1164     elseif temp_string == "*" then % if asterisk, append asterisk
1165         split_string[i] = temp_string
1166     else % if neither, raise error
1167         error(\M@number@assert)
1168     end
1169 end

```



The last step is to make sure that the first entry of `split_string` is a valid unicode index. We know that the first entry is either an asterisk or a number, and we make sure it is not an asterisk.

```
1170 local index = split_string[1]
1171 if index == "*" then
1172   error(\M@index@assert)
1173 end
```

The last check is to make sure the entry is (1) an integer and not a float; (2) nonnegative; and (3) less than 1,114,111, the maximum unicode entry. We round the entry down by subtracting the decimal portion, and the result will be equal to the original entry if and only if we began with an integer. We perform the three checks inside an `assert` and issue an error if any of them fail, and if `split_string` is valid, we return it to `mathfont:add_to_charm`.

```
1174 local rounded = index - (index \@percentchar 1) % subtract decimal portion
1175 local max = 1114111
1176 assert(index == rounded and index >= 0 and index <= max, \M@index@assert)
1177 return split_string
1178 end
```

We feed the user's charm information directly to `mathfont:add_to_charm`, which first calls `parse_charm` to parse the input and then modifies `mathfont` accordingly. After being parsed, we store the user's input in `charm_metrics`. The `index` is the base-10 unicode value of the character whose information we want to modify, and the `number_of_entries` is the length of `charm_metrics`.

```
1179 function mathfont:add_to_charm(charm_string)
1180   local charm_metrics = self.parse_charm(charm_string)
1181   local index = charm_metrics[1]
1182   local number_of_entries = \string# charm_metrics
```

If `mathfont` does not already have an entry for the unicode character `index`, we create an entry with type `u`.

```
1183 if not self[index] then
1184   self:new_type_u(index, {0, 0})
1185 end
```

Handling the user's input depends on the type of entry `index`. The basic procedure is to first check that the input has enough entries and, if yes, to overwrite the numbers stored in `mathfont`'s corresponding subtable with the new information. If the user included an asterisk, we do nothing to that metric value. For type `a`, we need four entries besides the `index`. The first two will overwrite the left and right offset, and the last two overwrite accent placement.

```
1186 if self[index].type == "a" then
1187   local entries_needed = 5
1188   assert(number_of_entries >= entries_needed, \M@entries@assert)
1189   if charm_metrics[2] \noexpand~="*" then
1190     self[index].left_stretch = charm_metrics[2] / 1000
1191   end
1192   if charm_metrics[3] \noexpand~="*" then
1193     self[index].right_stretch = charm_metrics[3] / 1000
```

```

1194     end
1195     if charm_metrics[4] \noexpand~= "*" then
1196         self[index].top_accent_stretch = charm_metrics[4] / 1000
1197     end
1198     if charm_metrics[5] \noexpand~= "*" then
1199         self[index].bot_accent_stretch = charm_metrics[5] / 1000
1200     end

```

Type `e` is more complicated. The number of entries in the `charm_metrics` must be at least  $2 * \text{total\_variants} + 3$ . We loop through the information and, for each  $i$ th pair of charm values, set those numbers to be the horizontal and vertical stretch information for the  $i$ th variant. We handle type `r` in the same way.

```

1201     elseif self[index].type == "e" then
1202         local tot_variants = self[index].total_variants
1203         local entries_needed = 2 * tot_variants + 3
1204         assert(number_of_entries >= entries_needed, \M@entries@assert)
1205         for i = 1, tot_variants, 1 do
1206             if charm_metrics[2*i] \noexpand~= "*" then
1207                 self[index].data[i].x = charm_metrics[2*i] / 1000
1208             end
1209             if charm_metrics[2*i+1] \noexpand~= "*" then
1210                 self[index].data[i].y = charm_metrics[2*i+1] / 1000
1211             end
1212         end

```

The final two entries for type `e` or `r` are the accent information.

```

1213         if charm_metrics[2*tot_variants+2] \noexpand~= "*" then
1214             self[index].top_accent_stretch = charm_metrics[2*tot_variants+2] / 1000
1215         end
1216         if charm_metrics[2*tot_variants+3] \noexpand~= "*" then
1217             self[index].bot_accent_stretch = charm_metrics[2*tot_variants+3] / 1000
1218         end

```

Again the information for type `u` is the simplest. We need two values besides the `index`, one for the top accent and one for the bottom accent.

```

1219     elseif self[index].type == "u" then
1220         local entries_needed = 3
1221         assert(number_of_entries >= entries_needed, \M@entries@assert)
1222         if charm_metrics[2] \noexpand~= "*" then
1223             self[index].top_accent_stretch = charm_metrics[2] / 1000
1224         end
1225         if charm_metrics[3] \noexpand~= "*" then
1226             self[index].bot_accent_stretch = charm_metrics[3] / 1000
1227         end
1228     end
1229 end

```

We end this section with three general-purpose Lua functions. The `make_hex_value` function accepts a nonnegative integer and returns its hexadecimal representation as a string. The result will go in the variable `hex_string`. We handle the cases of 0 and 1 manually.

```

1230 function mathfont.make_hex_value(integer)
1231   if integer == 0 then
1232     return "0000"
1233   end
1234   if integer == 1 then
1235     return "0001"
1236   end
1237   local hex_digits = "0123456789ABCDEF" % for reference
1238   local hex_string = ""
1239   local curr_val = integer
1240   local remainder = 0

```

Otherwise, we find the number of hexadecimal digits that we will need to represent the `integer`. We loop through the integers and stop when we reach the first power of 16 that is greater than `integer`.

```

1241   local i = 0
1242   while 16i <= curr_val do
1243     i = i+1
1244   end

```

Once we know how many hex digits we will need, we subtract off successively smaller powers of 16. Our dummy variable `j` starts as the greatest power of 16 less than or equal to `integer`, and we divide by  $16^j$ . The quotient becomes the first hexadecimal digit, and we repeat the process with the remainder and a smaller value of `j`. The final result is the hexadecimal representation of our original `integer`.

```

1245   for j = i-1, 0, -1 do
1246     remainder = curr_val \%percentchar(16j)
1247     curr_val = (curr_val - remainder) / (16j)
1248     hex_string = hex_string .. string.sub(hex_digits, curr_val+1, curr_val+1)
1249     curr_val = remainder
1250   end

```

If `hex_string` has fewer than 4 digits, we add enough leading 0's to bring it to 4 digits.

```

1251   if \string# hex_string < 4 then
1252     for i = \string# hex_string, 4, 1 do
1253       hex_string = "0" .. hex_string
1254     end
1255   end
1256   return hex_string
1257 end

```

The `glyph_info` function does exactly what it sounds like. It accepts a character table from a font and returns the width, height, depth, and italic correction values.

```

1258 function mathfont.glyph_info(char)
1259   local glyph_width = char.width or 0
1260   local glyph_height = char.height or 0
1261   local glyph_depth = char.depth or 0
1262   local glyph_italic = char.italic or 0
1263   return glyph_width, glyph_height, glyph_depth, glyph_italic
1264 end

```

Table 4: Callbacks Created by **mathfont**

Callback Name	Called?	Default Behavior
"mathfont.inspect_font"	Always	none
"mathfont.pre_adjust"		none
"mathfont.disable_nomath"	If nomath	mathfont.set_nomath_true
"mathfont.add_math_constants"	in fontdata	mathfont.math_constants
"mathfont.fix_character_metrics"	is set to true	mathfont.apply_charm_info
"mathfont.post_adjust"		none

The `:smash_glyph` function returns a character table that will produce a smashed version of the unicode character with value `index`. The character has no width, height, or depth and typesets the glyph virtually using a `char` font command.

```

1265 function mathfont:smash_glyph(index, fontdata)
1266   local smash_table = {}
1267   smash_table.width = 0
1268   smash_table.height = 0
1269   smash_table.depth = 0
1270   smash_table.commands = {"char", index}
1271   return smash_table
1272 end

```

An empty function that does nothing. Used later for creating callbacks.

```

1273 function mathfont.empty(arg)
1274 end

```

## 10 Adjust Fonts: Changes

This section contains the Lua functions that actually modify the font during loading. The three functions `set_nomath_true`, `math_constants`, and `apply_charm_info` do most of the heavy lifting, and we set them as the default behavior for three callbacks. In total, **mathfont** defines six different callbacks and calls them inside the function `adjust_font`—see table 4 for a list. Each callback accepts a `fontdata` object as an argument and returns nothing. You can use these callbacks to change **mathfont**'s default modifications or to modify a `fontdata` object before or after **mathfont** looks at it. Be aware that if you add a function to any of the `disable_nomath`, `add_math_constants`, or `fix_character_metrics` callbacks, LuaTeX will not call the default **mathfont** function associated with the callback anymore. In other words, do not mess with these three callbacks unless you are duplicating the functionality of the corresponding “Default Behavior” function from table 4.

We begin with the functions that modify character subtables in the font table, and in all cases, we return a new character table (or set of character tables in the case of type `e`) that we insert into the font object. For types `a` and `e`, we code the table from scratch, and for type `u`, we add information to the character tables that already exist in the font object. The three functions for assembling character tables take three arguments. The `index` argument is the unicode index of the base character that the function is modifying. The `charm_data`

argument is the subtable in `mathfont` of charm information that corresponds to `index`, and the `fontdata` argument is a font object. We will pull information from `charm_data` and `fontdata` to assemble the new table.

We will incorporate five categories of information into our new character tables: glyph dimensions, unicode values, accent placement dimensions, virtual font commands, and math kerning. For type `a`, we increase the original horizontal glyph dimensions based on charm information, and for type `e`, we increase the width by horizontal scale factors and the height and depth by vertical scale factors. Accent placement dimensions come from charm information. For types `a` and `e`, we return a character table that will become a virtual character in the font, and we need to include commands to typeset certain base characters. For type `e`, we also create the large variants through `pdf` commands that stretch the base glyphs.

The type `a` commands include one command to move to the right by some offset and one command to typeset the base glyph.

```
1275 function mathfont.make_a_commands(index, offset)
1276   local c_1 = {"right", offset}
1277   local c_2 = {"char", index}
1278   return {c_1, c_2}
1279 end
```

The `:make_a_table` returns a character table for type `a` characters. We store the information to return in the variable `a_table` and the character subtable in `char`. The `slant` is the font's `slant` parameter and is used for calculating accent placement.

```
1280 function mathfont:make_a_table(index, charm_data, fontdata)
1281   local a_table = {}
1282   local char = fontdata.characters[index] or {}
1283   local slant = fontdata.parameters.slant / 65536 or 0
```

The `left_stretch` and `right_stretch` values come from charm data and tell us how much extra space to add to the left and right sides of the character. Importantly, these values are additive.

```
1284   local left_stretch = charm_data.left_stretch
1285   local right_stretch = charm_data.right_stretch
1286   local width, height, depth, italic = self.glyph_info(char)
```

Incorporate the italic correction into the character width.

```
1287   width = width + italic
```

The new width is  $1 + \text{left\_stretch} + \text{right\_stretch}$  times the original width. The horizontal offset that appears in the commands is the `left_stretch` portion of the new width.

```
1288   local offset = width * left_stretch
1289   a_table.width = width * (1 + left_stretch + right_stretch)
1290   a_table.height = height
1291   a_table.depth = depth
1292   a_table.italic = italic
1293   a_table.unicode = index
```

The `tounicode` entry is a hexadecimal string that encodes the unicode value of the base character.

```
1294   a_table.tounicode = self.make_hex_value(index)
```

We specify accent placement information by including `top_accent` and `bot_accent` entries in the `a_table`. We determine placement by setting the `top_accent` to be a base value plus a distance determined by the charm data and similarly for `bot_accent`. We imagine dividing up the character's bounding box as follows: (1) some rectangular portion of the left and right areas of the bounding box is empty space added according to `left_stretch` and `right_stretch`; (2) accordingly, the glyph occupies some rectangular area in the middle of the bounding box; (3) if the font is slanted, that rectangle will actually be a parallelogram where the rectangle overhangs both slanted edges of the parallelogram in two triangles; and (4) we can determine the size of these triangles according to the `slant` font parameter. We want the base measurement for the top accent to be located in the middle of the parallelogram from step (3) previously, and we end up with

$$\text{base measurement} = \text{left\_stretch} * \text{width} + 0.5 * (\text{width} - \sigma_1 * \text{height}) + \sigma_1 * \text{height},$$

where  $\sigma_1$  is the `slant` parameter and `width` and `height` refer to the character in question. This equation simplifies to

$$(0.5 + \text{left\_stretch}) * \text{width} + 0.5\sigma_1 * \text{height},$$

which is the formula we use for the base value of the top accent. We determine the base value of the bottom accent similarly. For the shift amount, we take the corresponding factor from the charm information and multiply it by the width of the character. Note that in all these cases, we use the `width`, not the `new_width` as our unit of measurement. This keeps the scaling of the accent placement independent of the `left_stretch` and `right_stretch` values.

```

1295 local top_base = (0.5 + left_stretch) * width + 0.5 * slant * height
1296 local bot_base = (0.5 + left_stretch) * width - 0.5 * slant * height
1297 local top_accent_shift = charm_data.top_accent_stretch * width
1298 local bot_accent_shift = charm_data.bot_accent_stretch * width
1299 a_table.top_accent = top_base + top_accent_shift
1300 a_table.bot_accent = bot_base + bot_accent_shift

```

Add the commands to the table.

```

1301 a_table.commands = self.make_a_commands(index, offset)

```

Because we are keeping the character's italic correction, we have superscripts and subscripts that are too far from the glyph if we leave things as is. The reason is that LuaTeX adds italic correction of the nucleus to the horizontal position of superscripts when it formats exponents. Accordingly we want to move both superscript and subscript left by the italic correction of the nucleus, so we add a mathkern table to the character. A mathkern table contains up to four subtables, one for each corner of the character. Within each subtable, we store pairs of `height` and `kern` values, where `height` means to apply `kern` to exponents at that height. In this case, we have a `kern` value of minus italic correction in the upper and lower right corners.

```

1302 a_table.mathkern = {}
1303 a_table.mathkern.top_right = {{height = 0, kern = -italic}}
1304 a_table.mathkern.bottom_right = {{height = 0, kern = -italic}}
1305 a_table.mathkern.top_left = {{height = 0, kern = 0}}

```

```

1306 a_table.mathkern.bottom_left = {{height = 0, kern = 0}}
1307 return a_table
1308 end

```

For type e characters, we need a function to modify the base glyph. We incorporate the italic correction into the width and add extra italic correction in the case of the integral symbol.

```

1309 function mathfont:modify_e_base(index, fontdata)
1310 local char = fontdata.characters[index] or {}
1311 local width, height, depth, italic = self.glyph_info(char)
1312 char.width = width + italic

```

We trim the bounding box on the surd if the user requests it. Some text fonts extend the bounding box of the surd past the edge of the glyph, and we trim the edge of the box according to the values of `\M@SurdHorizontalFactor` and `\M@SurdVerticalFactor`.

```

1313 if index == 8730 then

```

Now get the scale factors from the T<sub>E</sub>X side of things and scale down (or up) the height and width of the surd.

```

1314 local horizontal_scale = tex.getcount("M@SurdHorizontalFactor") / 1000
1315 local vertical_scale = tex.getcount("M@SurdVerticalFactor") / 1000
1316 char.width = horizontal_scale * char.width
1317 char.height = vertical_scale * height
1318 end

```

For the integral symbol, get the scale factor add the appropriate italic correction.

```

1319 if index == 8747 then
1320 local scale_factor = tex.getcount("M@IntegralItalicFactor") / 1000
1321 char.italic = scale_factor * width
1322 end
1323 end

```

For the e commands, we not only typeset a certain glyph but also instruct the pdf backend to scale by a horizontal and vertical factor before doing so. In this way, we artificially add larger variants of a particular base glyph. The pdf command sends code directly to the pdf backend that handles the transformation. The q command indicates a linear transformation of the output, and the following string contains the transformation coordinates. The Q command restores the original coordinate system, and because it occurs between the transformation commands, the typeset glyph from the char command will be enlarged according to the transformation matrix.

```

1324 function mathfont.make_e_commands(index, h_stretch, v_stretch)
1325 local c_1 = {"pdf", "origin", string.format(
1326 "q \@percentchar s 0 0 \@percentchar s 0 0 cm", h_stretch, v_stretch)}
1327 local c_2 = {"char", index}
1328 local c_3 = {"pdf", "origin", "Q"}
1329 return {c_1, c_2, c_3}
1330 end

```

The function for type e characters returns a table with different structure because we need to create multiple characters at once. Specifically, the function returns a table with one entry for each larger variant that we want to add to the font. Many of the variables are

the same as in `:make_type_a`. We store the base character subtable in `char` and the font's `slant` parameter in `slant`. The `tounicode` stores the hexadecimal unicode value of the base character for reference later, and `smash_index` is the index of the unicode slot that we are using to hold the smashed version of the base character.

```

1331 function mathfont:make_e_table(index, charm_data, fontdata)
1332   local e_table = {}
1333   local char = fontdata.characters[index] or {}
1334   local slant = fontdata.parameters.slant / 65536
1335   local tounicode = self.make_hex_value(index)
1336   local smash_index = charm_data.smash
1337   local width, height, depth, italic = self.glyph_info(char)

```

We will create a number of entries in `e_table` equal to the number of variants we want, which is stored in `charm_data.total_variants`. We iteratively assemble the `e_table`, and we begin the iteration by extracting the `i`th horizontal and vertical scale factors from `charm_data`. The width, height, and depth of the `i`th new character will be scalings of these values from the original character.

```

1338   for i = 1, charm_data.total_variants, 1 do
1339     local h_stretch = charm_data.data[i].x
1340     local v_stretch = charm_data.data[i].y
1341     local new_width = width * h_stretch
1342     local new_height = height * v_stretch
1343     local new_depth = depth * v_stretch
1344     local new_italic = italic * h_stretch

```

We add new character bounds to the `i`th entry of `e_table`.

```

1345     e_table[i] = {}
1346     e_table[i].width = new_width
1347     e_table[i].height = new_height
1348     e_table[i].depth = new_depth
1349     e_table[i].italic = new_italic

```

Add the unicode information.

```

1350     e_table[i].unicode = index
1351     e_table[i].tounicode = tounicode

```

We handle accent placement the same way as with type a characters.

```

1352     local base_top_accent = 0.5 * new_width + 0.5 * slant * new_height
1353     local base_bot_accent = 0.5 * new_width - 0.5 * slant * new_height
1354     local top_accent_shift = charm_data.top_accent_stretch * new_width
1355     local bot_accent_shift = charm_data.bot_accent_stretch * new_width
1356     e_table[i].top_accent = base_top_accent + top_accent_shift
1357     e_table[i].bot_accent = base_bot_accent + bot_accent_shift

```

Add the commands.

```

1358     e_table[i].commands =
1359       self.make_e_commands(smash_index, h_stretch, v_stretch)

```

If we aren't dealing with the last entry in the table, we need to add the character's next fields. The next larger variant after the `i`th character will be the `i + 1`st character, and we can extract the index from the `charm` information.



```

1360     if i < charm_data.total_variants then
1361         e_table[i].next = charm_data.next[i+1]
1362     end
1363 end
1364 return e_table
1365 end

```

Making the u table is the easiest. We take the character subtable from `fontdata` as our starting point rather than assembling a new character subtable from scratch. The structure here is very similar to type a without the extra space from the `left_stretch` and `right_stretch`. Again, we incorporate the italic correction into the bounding box and add a negative mathkern to compensate.

```

1366 function mathfont:make_u_table(index, charm_data, fontdata)
1367     local u_table = fontdata.characters[index] or {}
1368     local slant = fontdata.parameters.slant / 65536 or 0
1369     local width, height, depth, italic = self.glyph_info(u_table)
1370     local new_width = width + italic
1371     u_table.width = new_width

```

We handle accents in the same way as with the other types.

```

1372     local base_top_accent = 0.5 * new_width + 0.5 * slant * height
1373     local base_bot_accent = 0.5 * new_width - 0.5 * slant * height
1374     local top_accent_shift = charm_data.top_accent_stretch * new_width
1375     local bot_accent_shift = charm_data.bot_accent_stretch * new_width
1376     u_table.top_accent = base_top_accent + top_accent_shift
1377     u_table.bot_accent = base_bot_accent + bot_accent_shift

```

Add a mathkern table as in the case of type a characters.

```

1378     u_table.mathkern = {}
1379     u_table.mathkern.top_right = {{height = 0, kern = -italic}}
1380     u_table.mathkern.bottom_right = {{height = 0, kern = -italic}}
1381     u_table.mathkern.top_left = {{height = 0, kern = 0}}
1382     u_table.mathkern.bottom_left = {{height = 0, kern = 0}}
1383     return u_table
1384 end

```

Before we get to the main font-changing functions, we code `make_fake_angle`, which returns a character table for the fake angle brackets. The function accepts the index of the smashed character as `index` and the index of the smashed gillement as `smash`. We form the fake angle bracket by using only the top 90% of the original glyph, and we scale it to have the same height and depth as the left parenthesis.

```

1385 function mathfont.make_fake_angle(index, smash, fontdata)
1386     local temp = {}
1387     local lparen = fontdata.characters[40] or {}
1388     local lparen_height = lparen.height or 0
1389     local lparen_depth = lparen.depth or 0
1390     local glyph = fontdata.characters[index] or {}
1391     local glyph_height = glyph.height or 0
1392     local base_height = 0.9 * glyph_height

```

```

1393 local factor = 0
1394 if glyph_height \noexpand~ = 0 then
1395   factor = (lparen_height + lparen_depth) / base_height
1396 end
1397 local shift = 0.1 * glyph_height * factor + lparen_depth
1398 temp.height = lparen_height
1399 temp.depth = lparen_depth
1400 temp.width = glyph.width or 0
1401 temp.italic = glyph.italic or 0
1402 temp.top_accent = glyph.top_accent or 0.5 * temp.width
1403 temp.bot_accent = glyph.bot_accent or 0.5 * temp.width
1404 temp.commands = {
1405   {"down", shift},
1406   {"pdf", "origin", string.format("q 1 0 0 \@percentchar s 0 0 cm", factor)},
1407   {"char", smash},
1408   {"pdf", "origin", "Q"},
1409   {"down", -shift}}
1410 return temp
1411 end

```

We come to the main functions that modify the font. We need to accomplish three tasks, and we define separate functions for each one. First, we set the font's `nomath` entry to `false`. Second, we incorporate the modifications based on charm information into the font, i.e. set the font's character subtables using the previous functions from this section. Third, we need to add a `MathConstants` table. The first task is very easy.

```

1412 function mathfont.set_nomath_true(fontdata)
1413   fontdata.nomath = false
1414   fontdata.oldmath = false
1415 end

```

The second task is more involved. The basic idea is to loop through `mathfont`, and whenever we find an entry that is a subtable, we treat it as charm information that we use to modify the font object. We begin by storing the character information from the font in `chars` for easier reference later.

```

1416 function mathfont.apply_charm_info(fontdata)
1417   local chars = fontdata.characters or {}

```

Before we loop through the charm data, we need to add fake angle brackets and `\nabla` to the font. We begin with the angle brackets.

```

1418   chars[1044538] = mathfont.smash_glyph(8249, fontdata) % \lguil
1419   chars[1044539] = mathfont.smash_glyph(8250, fontdata) % \rguil
1420   chars[1044540] = mathfont.smash_glyph(171, fontdata) % \llguil
1421   chars[1044541] = mathfont.smash_glyph(187, fontdata) % \rrguil

```

Now add the characters to the font.

```

1422   chars[1044508] = mathfont.make_fake_angle(8249, 1044538, fontdata)
1423   chars[1044509] = mathfont.make_fake_angle(8250, 1044539, fontdata)
1424   chars[1044510] = mathfont.make_fake_angle(171, 1044540, fontdata)
1425   chars[1044511] = mathfont.make_fake_angle(187, 1044541, fontdata)

```

Add the nabla (inverted Delta) character to the font if it is missing.

```

1426  if not chars[8711] then
1427    chars[8710] = chars[8710] or {}
1428    chars[1044508] = mathfont:smash_glyph(8710, fontdata)
1429    chars[8711] = {}
1430    chars[8711].width = chars[8710].width or 0
1431    chars[8711].height = chars[8710].height or 0
1432    chars[8711].depth = chars[8710].depth or 0
1433    chars[8711].italic = chars[8710].italic or 0
1434    chars[8711].top_accent = chars[8710].top_accent or 0.5 * chars[8711].width
1435    chars[8711].bot_accent = chars[8710].bot_accent or 0.5 * chars[8711].width
1436    chars[8711].unicode = 8711
1437    chars[8711].tounicode = mathfont.make_hex_value(8711)
1438    chars[8711].commands = {
1439      {"down", -chars[8711].height},
1440      {"pdf", "origin", "q 1 0 0 -1 0 0 cm"},
1441      {"char", 1044508},
1442      {"pdf", "origin", "Q"},
1443      {"down", chars[8711].height}}
1444  end

```

Perform the loop. We care about entries `info` whose type is a table.

```

1445  for index, info in pairs(mathfont) do
1446    if type(info) == "table" then

```

If the character's type is `a`, all we need to do is replace the character subtable in the font with our version.

```

1447      if info.type == "a" then
1448        chars[info.next] = mathfont:make_a_table(index, info, fontdata)

```

Again, type `e` is more complicated. This time we need to insert multiple character subtables into the font, one for the smashed version of the base glyph and others corresponding to the large variants that we create using the `:make_e_table` function from above. We also need to add `next` entries to the characters in the font linking all the variants together.

```

1449      elseif info.type == "e" then
1450        local smash = info.smash
1451        chars[index] = chars[index] or {}

```

Set the `next` entry on the current character, modify the character's dimensions to incorporate italic correction into the width, and add a smashed version of the glyph into the font.

```

1452        chars[index].next = info.next[1]
1453        mathfont:modify_e_base(index, fontdata)
1454        chars[smash] = mathfont:smash_glyph(index, fontdata)

```

The function that creates the character table for type `e` produces one character subtable for each larger variant that we want to add, so we loop through the resulting table and add the contents to the font one at a time. Each subtable goes in unicode slots that we take from the charm information, specifically the `next` table from `info`.

```

1455        local variants_table = mathfont:make_e_table(index, info, fontdata)
1456        for i = 1, info.total_variants, 1 do

```

```

1457         chars[info.next[i]] = variants_table[i]
1458     end

```

We deal with type u in the same way as we do type a.

```

1459     elseif info.type == "u" then
1460         chars[index] = mathfont:make_u_table(index, info, fontdata)
1461     end
1462 end
1463 end
1464 end

```

The `populate_math_constants` function is even more complicated because we need to add a full `MathConstants` table to the font object, which has some fifty parameters that we need to set. To keep things simple, we set the font parameters in terms of traditional  $\TeX$  `\fontdimen` parameters. Besides the eight essential parameters found in all fonts,  $\TeX$  traditionally uses some fifteen extra parameters to typeset math formulas. To preserve whatever structure may already exist in the font object, we do not override any `MathConstants` that the font already contains.

```

1465 function mathfont.math_constants(fontdata)
1466     fontdata.MathConstants = fontdata.MathConstants or {}

```

First evaluate the dimensions from the font object that we will use in determining other math parameter values. The `A_height` is the height of the capital “A” character, and the `y_depth` is the depth of the lower-case “y” character. Both will be 0 if the font does not have the correct character.

```

1467     local size = fontdata.size or 0
1468     local ex = fontdata.parameters.x_height or 0
1469     local em = fontdata.parameters.quad or 0
1470     local A_height = 0
1471     local y_depth = 0
1472     if fontdata.characters[65] then
1473         A_height = fontdata.characters[65].height or 0 % A
1474     end
1475     if fontdata.characters[121] then
1476         y_depth = fontdata.characters[121].depth or 0 % y
1477     end

```

We begin by setting the axis height and default rule thickness. We need to start with these parameters because we will use them to calculate other constants. We set both values to 0 initially and then change them.

```

1478     local axis = 0
1479     local rule_thickness = 0

```

Set the default rule thickness. If the font already has a value set for the parameter `FractionRuleThickness`, we take that as the default rule thickness, and otherwise we set it to be 1/18 of the font size times the adjustment factor from `\M@RuleThicknessFactor`, which is the value of that `\count` divided by 1000.

```

1480     local dim = "FractionRuleThickness"
1481     if not fontdata.MathConstants[dim] then
1482         local scale_factor = tex.getcount("M@RuleThicknessFactor") / 1000

```

```

1483     rule_thickness = (size / 18) * scale_factor
1484     fontdata.MathConstants[dim] = rule_thickness
1485   else
1486     rule_thickness = fontdata.MathConstants[dim]
1487   end

```

If the font does not have `AxisHeight` already set, we set the axis to be the height of a minus sign (character 45). As a fallback, we set the axis to `0.8ex` if the font does not have a character in unicode slot 45. If the font has an `AxisHeight`, we take that value as the `axis`.

```

1488   local dim = "AxisHeight"
1489   if fontdata.MathConstants[dim] then
1490     axis = fontdata.MathConstants[dim]
1491   else
1492     if fontdata.characters[45] then
1493       axis = fontdata.characters[45].height - 0.5 * rule_thickness
1494     else
1495       axis = 0.8 * ex
1496     end
1497     fontdata.MathConstants[dim] = axis
1498   end

```

Apart from the axis height and rule thickness, we can group the traditional mathematics `\fontdimen` parameters into three categories: four for large operators, five for fractions, and six for superscripts and subscripts. (OpenType math does not use the fifth large-operator parameter  $\xi_{13}$  and the seventh script parameter  $\sigma_{14}$ .) We define variables with the same names as their traditional references from Appendix G in the *TEXBook*. I have taken the design approach of using twice the rule height as a standard minimum clearance, and I am assuming that script styles are roughly 70% as large as text and display styles. We begin with the parameters for large operators.

The parameter  $\xi_9$  is the minimum clearance between the top of a large operator and the limit above it, and we set it to be twice the rule thickness. Before ensuring that the bottom of the upper limit is at least  $\xi_9$  away from the operator character, `TEX` attempts to position the baseline of the limit at  $\xi_{10}$  distance above the operator character, and we set  $\xi_{10}$  to be slightly larger than  $\xi_9$ . If the upper limit has no descender, `TEX` will raise its baseline by  $\xi_{10}$ , and if it has a descender, `TEX` will position the bottom of the descender to be  $\xi_9$  above the operator, which in practice means it will be higher than limits without descenders. This approach balances the desire for consistency in whitespace with the desire for consistency in baseline height. Similarly, we set the minimum clearance  $\xi_{11}$  for the lower limit to be equal to the attempted clearance for the upper limit, and the attempted clearance  $\xi_{12}$  for the lower limit will be the minimum clearance plus the average of the `\scriptfont` x-height and `\scriptfont` A-height.

```

1499   local xi_9 = 2 * rule_thickness           % upper limit minimum clearance
1500   local xi_10 = xi_9 + 0.35 * y_depth     % upper limit attempt placement
1501   local xi_11 = xi_10                     % lower limit minimum clearance
1502   local xi_12 = xi_10 + 0.35 * (A_height + ex) % lower limit attempt placement

```

Our general approach for `\displaystyle` fractions is to place the baseline of the numerator numerator at a distance above the fraction rule of 1.5 times the rule height plus descender

depth plus a small extra space. The minimum clearance will be the rule height, so we expect the numerator to strictly exceed the minimum clearance in most situations. Doing so produces consistent baselines of numerators and gives our value for  $\sigma_8$ , the attempted height of the numerator in `\displaystyle` fractions. For smaller styles, we use a single rule height as clearance, so we add  $0.5 * \text{rule\_thickness} + \text{y\_depth}$  scaled down by 0.7 to the rule thickness. The minimum clearance for numerator and denominator are separate OpenType parameters, and we set them later. The extra 0.1 A-height in the attempted clearance relative to the minimum clearance appears because we measure attempted clearance from the axis, whereas we measure minimum clearance from the top or bottom of the fraction rule.

```
1503 local sigma_8 = axis + 1.5 * rule_thickness + y_depth + 0.1 * A_height
1504 local sigma_9 = (axis + 1.35 * rule_thickness + 0.7 * y_depth +
1505     0.07 * A_height)
1506 local sigma_10 = sigma_9
```

Our approach in the denominators is the same except that we add half the descender depth to the minimum clearance. This creates extra space below the fraction rule so that the typographical color above the rule matches that below the rule when the numerator contains descenders.

```
1507 local sigma_11 = (-axis + 1.5 * rule_thickness + 0.5 * y_depth +
1508     1.1 * A_height)
1509 local sigma_12 = (-axis + 1.35 * rule_thickness + 0.35 * y_depth +
1510     0.77 * A_height)
```

For superscripts we think in terms of the top of the superscript. We raise the baseline of the superscript by the desired height of the superscript top minus the `\scriptfont` A-height. Choosing  $1.3 * \text{A\_height}$  for regular styles and  $1.2 * \text{A\_height}$  for cramped styles was a design choice that worked well. The attempted drop for subscripts is one-fifth the A-height or slightly more than the y-depth, whichever is greater. This way the subscript baseline is slightly lower than any descenders, and for fonts without descenders, we still clearly lower the subscript. Setting  $\sigma_{18}$  and  $\sigma_{19}$  was another design choice that worked well.

```
1511 local sigma_13 = 0.6 * A_height      % attempted superscript height
1512 local sigma_15 = 0.5 * A_height      % attempted superscript for \cramped
1513 local sigma_16 = 1.1 * y_depth       % attempted subscript lower
1514 if sigma_16 < 0.2 * A_height then
1515     sigma_16 = 0.2 * A_height
1516 end
1517 local sigma_17 = sigma_16             % sigma_16 when superscript present
1518 local sigma_18 = 0.5 * A_height      % superscript lower for boxed subformula
1519 local sigma_19 = 0.1 * A_height      % subscript lower for boxed subformula
```

The MathConstants themselves come from the unicode equivalents of the traditional `\TeX` `\fontdimen` parameters where appropriate. Where not appropriate, I made design choices as indicated. Setting the next three parameters was purely a design choice.

```
1520 local dim = "DisplayOperatorMinHeight"
1521 if not fontdata.MathConstants[dim] then
1522     fontdata.MathConstants[dim] = 1.8 * A_height
1523 end
1524 local dim = "FractionDelimiterDisplayStyleSize"
```

```

1525 if not fontdata.MathConstants[dim] then
1526   fontdata.MathConstants[dim] = 2 * size
1527 end
1528 local dim = "FractionDelimiterSize"
1529 if not fontdata.MathConstants[dim] then
1530   fontdata.MathConstants[dim] = 1.3 * size
1531 end
1532 local dim = "FractionDenominatorDisplayStyleShiftDown"
1533 if not fontdata.MathConstants[dim] then
1534   fontdata.MathConstants[dim] = sigma_11
1535 end
1536 local dim = "FractionDenominatorShiftDown"
1537 if not fontdata.MathConstants[dim] then
1538   fontdata.MathConstants[dim] = sigma_12
1539 end

```

We set the minimum clearance for the numerator to be twice the rule height in `\displaystyle` and the rule height in other styles. Our approach in setting the attempted height of the numerator ( $\sigma_8$  and  $\sigma_9$ ) was to add the minimum clearance plus the descender depth plus a small extra space, so in general, we do not expect the numerator to run into the minimum clearance. For the denominator, we do the same thing except add half the descender depth to the clearance, which balances the amount of color above and below the fraction rule and is similar to what we did for the lower limits on big operators when we set  $\xi_{11}$  larger than  $\xi_9$ .

```

1540 local dim = "FractionDenominatorDisplayStyleGapMin"
1541 if not fontdata.MathConstants[dim] then
1542   fontdata.MathConstants[dim] = rule_thickness + 0.5 * y_depth
1543 end
1544 local dim = "FractionDenominatorGapMin"
1545 if not fontdata.MathConstants[dim] then
1546   fontdata.MathConstants[dim] = rule_thickness + 0.35 * y_depth
1547 end
1548 local dim = "FractionNumeratorDisplayStyleShiftUp"
1549 if not fontdata.MathConstants[dim] then
1550   fontdata.MathConstants[dim] = sigma_8
1551 end
1552 local dim = "FractionNumeratorShiftUp"
1553 if not fontdata.MathConstants[dim] then
1554   fontdata.MathConstants[dim] = sigma_9
1555 end
1556 local dim = "FractionNumeratorDisplayStyleGapMin"
1557 if not fontdata.MathConstants[dim] then
1558   fontdata.MathConstants[dim] = rule_thickness
1559 end
1560 local dim = "FractionNumeratorGapMin"
1561 if not fontdata.MathConstants[dim] then
1562   fontdata.MathConstants[dim] = rule_thickness
1563 end

```

The `SkewedFractionHorizontalGap` and `SkewedFractionVerticalGap` take the values that Lua $\TeX$  would set for a traditional  $\TeX$  font.

```

1564 local dim = "SkewedFractionHorizontalGap"
1565 if not fontdata.MathConstants[dim] then
1566     fontdata.MathConstants[dim] = 0.5 * em
1567 end
1568 local dim = "SkewedFractionVerticalGap"
1569 if not fontdata.MathConstants[dim] then
1570     fontdata.MathConstants[dim] = ex
1571 end

```

The `UpperLimit` and `LowerLimit` dimensions correspond exactly to traditional  $\TeX$  math `\fontdimen` parameters.

```

1572 local dim = "UpperLimitBaselineRiseMin"
1573 if not fontdata.MathConstants[dim] then
1574     fontdata.MathConstants[dim] = xi_11
1575 end
1576 local dim = "UpperLimitGapMin"
1577 if not fontdata.MathConstants[dim] then
1578     fontdata.MathConstants[dim] = xi_9
1579 end
1580 local dim = "LowerLimitBaselineDropMin"
1581 if not fontdata.MathConstants[dim] then
1582     fontdata.MathConstants[dim] = xi_12
1583 end
1584 local dim = "LowerLimitGapMin"
1585 if not fontdata.MathConstants[dim] then
1586     fontdata.MathConstants[dim] = xi_10
1587 end

```

Traditional  $\TeX$  doesn't have stack objects, but they are meant to be similar to large operators, so we set the same parameters.

```

1588 local dim = "StretchStackGapBelowMin"
1589 if not fontdata.MathConstants[dim] then
1590     fontdata.MathConstants[dim] = xi_10
1591 end
1592 local dim = "StretchStackTopShiftUp"
1593 if not fontdata.MathConstants[dim] then
1594     fontdata.MathConstants[dim] = xi_11
1595 end
1596 local dim = "StretchStackGapAboveMin"
1597 if not fontdata.MathConstants[dim] then
1598     fontdata.MathConstants[dim] = xi_9
1599 end
1600 local dim = "StretchStackBottomShiftDown"
1601 if not fontdata.MathConstants[dim] then
1602     fontdata.MathConstants[dim] = xi_12
1603 end

```



For the three `Overbar` parameters, we take the approach that the bar itself should be as thick as the rule height. The gap will be twice the rule height, and the extra clearance will be a single rule height.

```

1604 local dim = "OverbarExtraAscender"
1605 if not fontdata.MathConstants[dim] then
1606   fontdata.MathConstants[dim] = rule_thickness
1607 end
1608 local dim = "OverbarRuleThickness"
1609 if not fontdata.MathConstants[dim] then
1610   fontdata.MathConstants[dim] = rule_thickness
1611 end
1612 local dim = "OverbarVerticalGap"
1613 if not fontdata.MathConstants[dim] then
1614   fontdata.MathConstants[dim] = 2 * rule_thickness
1615 end

```

For the radical sign, we take the same approach as with the `Overbar` parameters. We insert one rule thickness of extra space above the radical symbol and two rule thickness of extra space under it. For `\textstyle` and smaller, we reduce the space to a single rule height.

```

1616 local dim = "RadicalExtraAscender"
1617 if not fontdata.MathConstants[dim] then
1618   fontdata.MathConstants[dim] = rule_thickness
1619 end
1620 local dim = "RadicalRuleThickness"
1621 if not fontdata.MathConstants[dim] then
1622   fontdata.MathConstants[dim] = rule_thickness
1623 end
1624 local dim = "RadicalDisplayStyleVerticalGap"
1625 if not fontdata.MathConstants[dim] then
1626   fontdata.MathConstants[dim] = 2 * rule_thickness
1627 end
1628 local dim = "RadicalVerticalGap"
1629 if not fontdata.MathConstants[dim] then
1630   fontdata.MathConstants[dim] = rule_thickness
1631 end

```

The final three `Radical` parameters aren't used if we handle degree placement at the macro level rather than at the font level. We set them to the default values that Lua $\TeX$  uses for traditional tfm fonts.

```

1632 local dim = "RadicalKernBeforeDegree"
1633 if not fontdata.MathConstants[dim] then
1634   fontdata.MathConstants[dim] = (5/18) * em
1635 end
1636 local dim = "RadicalKernAfterDegree"
1637 if not fontdata.MathConstants[dim] then
1638   fontdata.MathConstants[dim] = (10/18) * em
1639 end
1640 local dim = "RadicalDegreeBottomRaisePercent"

```

```

1641 if not fontdata.MathConstants[dim] then
1642   fontdata.MathConstants[dim] = 60
1643 end

```

The `SpaceAfterShift` is a design choice. Somewhat arbitrary.

```

1644 local dim = "SpaceAfterScript"
1645 if not fontdata.MathConstants[dim] then
1646   fontdata.MathConstants[dim] = 0.1 * em
1647 end

```

The `Stack` parameters come from their traditional `\fontdimen` analogues.

```

1648 local dim = "StackBottomDisplayStyleShiftDown"
1649 if not fontdata.MathConstants[dim] then
1650   fontdata.MathConstants[dim] = sigma_11
1651 end
1652 local dim = "StackBottomShiftDown"
1653 if not fontdata.MathConstants[dim] then
1654   fontdata.MathConstants[dim] = sigma_12
1655 end
1656 local dim = "StackTopDisplayStyleShiftUp"
1657 if not fontdata.MathConstants[dim] then
1658   fontdata.MathConstants[dim] = sigma_8
1659 end
1660 local dim = "StackTopShiftUp"
1661 if not fontdata.MathConstants[dim] then
1662   fontdata.MathConstants[dim] = sigma_10
1663 end

```

Traditionally  $\TeX$  uses an internal method rather than a parameter to determine the minimum distance between two boxes in an `\atop` stack. We set the minimum distance to be one rule thickness plus the combined minimum clearance for numerators and denominators in fractions. For `\displaystyle`, that gives us

$$\text{rule\_thickness} + (2 * \text{rule\_thickness}) + (2 * \text{rule\_thickness} + 0.5 * \text{y\_depth})$$

For smaller styles, we use single rule height values and scale down the `y_depth` by 0.7.

```

1664 local dim = "StackDisplayStyleGapMin"
1665 if not fontdata.MathConstants[dim] then
1666   fontdata.MathConstants[dim] = 5 * rule_thickness + 0.5 * y_depth
1667 end
1668 local dim = "StackGapMin"
1669 if not fontdata.MathConstants[dim] then
1670   fontdata.MathConstants[dim] = 3 * rule_thickness + 0.35 * y_depth
1671 end

```

With three exceptions, superscript and subscript parameters come from traditional  $\TeX$  dimensions.

```

1672 local dim = "SubscriptShiftDown"
1673 if not fontdata.MathConstants[dim] then
1674   fontdata.MathConstants[dim] = sigma_16

```

```

1675 end
1676 local dim = "SubscriptBaselineDropMin"
1677 if not fontdata.MathConstants[dim] then
1678   fontdata.MathConstants[dim] = sigma_19
1679 end
1680 local dim = "SubscriptShiftDownWithSuperscript"
1681 if not fontdata.MathConstants[dim] then
1682   fontdata.MathConstants[dim] = sigma_17
1683 end

```

The top of a subscript should be less than half the A-height. This is a somewhat arbitrary design choice.

```

1684 local dim = "SubscriptTopMax"
1685 if not fontdata.MathConstants[dim] then
1686   fontdata.MathConstants[dim] = 0.5 * A_height
1687 end

```

The minimum gap between superscripts and subscripts will be the height of the rule. This is less space than  $\text{\TeX}$  traditionally allocates.

```

1688 local dim = "SubSuperscriptGapMin"
1689 if not fontdata.MathConstants[dim] then
1690   fontdata.MathConstants[dim] = rule_thickness
1691 end

```

We set the minimum height for the bottom of a subscript to be the height of a superscript in cramped styles minus the depth of a possible descender. Theoretically this is the lowest that any portion of a superscript should ever be if it contains only text.

```

1692 local dim = "SuperscriptBottomMin"
1693 if not fontdata.MathConstants[dim] then
1694   fontdata.MathConstants[dim] = sigma_15 - 0.7 * y_depth
1695 end
1696 local dim = "SuperscriptBaselineDropMax"
1697 if not fontdata.MathConstants[dim] then
1698   fontdata.MathConstants[dim] = sigma_18
1699 end
1700 local dim = "SuperscriptShiftUp"
1701 if not fontdata.MathConstants[dim] then
1702   fontdata.MathConstants[dim] = sigma_13
1703 end
1704 local dim = "SuperscriptShiftUpCramped"
1705 if not fontdata.MathConstants[dim] then
1706   fontdata.MathConstants[dim] = sigma_15
1707 end

```

If the superscript and subscript overlap, we choose the new position such that the baselines of subscripts are roughly consistent across subformulas. In this case, the bottom of the superscript box will rise at most to the point such that a subscript containing only text at 70% of the next-larger style will align with all similar subscripts. The top of the subscript will have approximate height  $-\sigma_{16} + 0.7 * A\_height$  above the baseline, so to find our desired position for the bottom of the superscript, we add the minimum clearance of a single rule

thickness. Putting this parameter in terms of the subscript sizing is necessary because we don't know how large the descender will be in a given subscript.

```

1708 local dim = "SuperscriptBottomMaxWithSubscript"
1709 if not fontdata.MathConstants[dim] then
1710   fontdata.MathConstants[dim] = -sigma_16 + 0.7 * A_height + rule_thickness
1711 end

```

As with the Overbar parameters, we set the extra clearance to be the rule height and the gap to be twice the rule height.

```

1712 local dim = "UnderbarExtraDescender"
1713 if not fontdata.MathConstants[dim] then
1714   fontdata.MathConstants[dim] = rule_thickness
1715 end
1716 local dim = "UnderbarRuleThickness"
1717 if not fontdata.MathConstants[dim] then
1718   fontdata.MathConstants[dim] = rule_thickness
1719 end
1720 local dim = "UnderbarVerticalGap"
1721 if not fontdata.MathConstants[dim] then
1722   fontdata.MathConstants[dim] = 2 * rule_thickness
1723 end

```

No reason not to set `MinConnectorOverlap` to 0. It doesn't matter for our purposes because `mathfont` doesn't use extensibles.

```

1724 local dim = "MinConnectorOverlap"
1725 if not fontdata.MathConstants[dim] then
1726   fontdata.MathConstants[dim] = 0
1727 end
1728 end

```

Time for callbacks! We create six of them.

```

1729 luatexbase.create_callback("mathfont.inspect_font", "simple", mathfont.empty)
1730 luatexbase.create_callback("mathfont.pre_adjust", "simple", mathfont.empty)
1731 luatexbase.create_callback("mathfont.disable_nomath", "simple",
1732   mathfont.set_nomath_true)
1733 luatexbase.create_callback("mathfont.add_math_constants", "simple",
1734   mathfont.math_constants)
1735 luatexbase.create_callback("mathfont.fix_character_metrics", "simple",
1736   mathfont.apply_charm_info)
1737 luatexbase.create_callback("mathfont.post_adjust", "simple", mathfont.empty)

```

The functions `mathfont.info` and `mathfont.get_font_name` are used for informational messaging. The first prints a message in the log file, and the second returns a font name.

```

1738 function mathfont.info(msg)
1739   texio.write_nl("log", "Package mathfont Info: " .. msg)
1740 end
1741 function mathfont.get_font_name(fontdata)
1742   return fontdata.fullname or fontdata.psname or fontdata.name or "<??"
1743 end

```

The `adjust_font` function is what we will actually be adding to `luaotfload.patch_font`. This function calls the six callbacks at appropriate times and writes informational messages in the log file.

```

1744 function mathfont.adjust_font(fontdata)
1745   luatexbase.call_callback("mathfont.inspect_font", fontdata)
1746   if fontdata.nomath then
1747     mathfont.info("Adjusting font " .. mathfont.get_font_name(fontdata) .. ".")
1748     luatexbase.call_callback("mathfont.pre_adjust", fontdata)
1749     luatexbase.call_callback("mathfont.disable_nomath", fontdata)
1750     luatexbase.call_callback("mathfont.add_math_constants", fontdata)
1751     luatexbase.call_callback("mathfont.fix_character_metrics", fontdata)
1752     luatexbase.call_callback("mathfont.post_adjust", fontdata)
1753   else
1754     mathfont.info("No changes made to " ..
1755       mathfont.get_font_name(fontdata) .. ".")
1756   end
1757 end

```

Finally, add the processing function to `luaotfload's patch_font` callback.

```

1758 luatexbase.add_to_callback("luaotfload.patch_font", mathfont.adjust_font,
1759   "mathfont.adjust_font")

```

## 11 Adjust Fonts: Metrics

This section contains the default charm information for the characters that `mathfont` adjusts upon loading a font. We will make new variants in the private use area of the font. Lower-case Latin letters will fill unicode slots U+FF000 through U+FF021, which are located in the Supplemental Private Use Area-A portion of the unicode table.

```

1760 mathfont:new_type_a(97, 1044480, {50, 50, -50, 0}) % a
1761 mathfont:new_type_a(98, 1044481, {50, 50, -50, 0}) % b
1762 mathfont:new_type_a(99, 1044482, {50, 50, 0, 0}) % c
1763 mathfont:new_type_a(100, 1044483, {50, -50, -50, 0}) % d
1764 mathfont:new_type_a(101, 1044484, {50, 50, 0, 0}) % e
1765 mathfont:new_type_a(102, 1044485, {200, 0, 0, 0}) % f
1766 mathfont:new_type_a(103, 1044486, {100, 50, -50, 0}) % g
1767 mathfont:new_type_a(104, 1044487, {50, 0, -50, 0}) % h
1768 mathfont:new_type_a(105, 1044488, {50, 100, -100, 0}) % i
1769 mathfont:new_type_a(106, 1044489, {400, 50, -50, 0}) % j
1770 mathfont:new_type_a(107, 1044490, {50, 50, -100, 0}) % k
1771 mathfont:new_type_a(108, 1044491, {100, 150, -100, 0}) % l
1772 mathfont:new_type_a(109, 1044492, {50, 0, 0, 0}) % m
1773 mathfont:new_type_a(110, 1044493, {50, 0, 0, 0}) % n
1774 mathfont:new_type_a(111, 1044494, {50, 0, 0, 0}) % o
1775 mathfont:new_type_a(112, 1044495, {200, 50, -50, 0}) % p
1776 mathfont:new_type_a(113, 1044496, {50, 0, -50, 0}) % q
1777 mathfont:new_type_a(114, 1044497, {100, 100, -50, 0}) % r
1778 mathfont:new_type_a(115, 1044498, {50, 50, -50, 0}) % s

```

```

1779 mathfont:new_type_a(116, 1044499, {50, 50, -50, 0}) % t
1780 mathfont:new_type_a(117, 1044500, {0, 50, 0, 0}) % u
1781 mathfont:new_type_a(118, 1044501, {0, 50, -50, 0}) % v
1782 mathfont:new_type_a(119, 1044502, {0, 50, 0, 0}) % w
1783 mathfont:new_type_a(120, 1044503, {50, 0, -50, 0}) % x
1784 mathfont:new_type_a(121, 1044504, {150, 50, -50, 0}) % y
1785 mathfont:new_type_a(122, 1044505, {100, 50, -100, 0}) % z
1786 mathfont:new_type_a(305, 1044506, {100, 100, -150, 0}) % \imath
1787 mathfont:new_type_a(567, 1044507, {700, 50, -150, 0}) % \jmath

```

Upper-case Latin letters will fill unicode slots U+FF020 through U+FF039.

```

1788 mathfont:new_type_a(65, 1044512, {50, 0, 150, 0}) % A
1789 mathfont:new_type_a(66, 1044513, {50, 0, 0, 0}) % B
1790 mathfont:new_type_a(67, 1044514, {0, 0, 0, 0}) % C
1791 mathfont:new_type_a(68, 1044515, {50, 0, -50, 0}) % D
1792 mathfont:new_type_a(69, 1044516, {50, 0, 0, 0}) % E
1793 mathfont:new_type_a(70, 1044517, {50, 0, 0, 0}) % F
1794 mathfont:new_type_a(71, 1044518, {0, 0, 0, 0}) % G
1795 mathfont:new_type_a(72, 1044519, {50, 0, -50, 0}) % H
1796 mathfont:new_type_a(73, 1044520, {100, 0, 0, 0}) % I
1797 mathfont:new_type_a(74, 1044521, {50, 0, 100, 0}) % J
1798 mathfont:new_type_a(75, 1044522, {50, 0, 0, 0}) % K
1799 mathfont:new_type_a(76, 1044523, {50, 0, -180, 0}) % L
1800 mathfont:new_type_a(77, 1044524, {50, 0, -50, 0}) % M
1801 mathfont:new_type_a(78, 1044525, {50, 0, -50, 0}) % N
1802 mathfont:new_type_a(79, 1044526, {0, 0, 0, 0}) % O
1803 mathfont:new_type_a(80, 1044527, {0, 0, -50, 0}) % P
1804 mathfont:new_type_a(81, 1044528, {0, 50, 0, 0}) % Q
1805 mathfont:new_type_a(82, 1044529, {50, 0, -50, 0}) % R
1806 mathfont:new_type_a(83, 1044530, {0, 0, -50, 0}) % S
1807 mathfont:new_type_a(84, 1044531, {0, 0, -50, 0}) % T
1808 mathfont:new_type_a(85, 1044532, {0, 0, -50, 0}) % U
1809 mathfont:new_type_a(86, 1044533, {0, 50, 0, 0}) % V
1810 mathfont:new_type_a(87, 1044534, {0, 50, -50, 0}) % W
1811 mathfont:new_type_a(88, 1044535, {50, 0, 0, 0}) % X
1812 mathfont:new_type_a(89, 1044536, {0, 0, -50, 0}) % Y
1813 mathfont:new_type_a(90, 1044537, {50, 0, -50, 0}) % Z

```

The Greek characters will be type u, so we don't need extra unicode slots for them. In future editions of `mathfont`, they may become type a with adjusted bounding boxes, but I don't have immediate plans for such a change.

```

1814 mathfont:new_type_u(945, {0, 0}) % \alpha
1815 mathfont:new_type_u(946, {0, 0}) % \beta
1816 mathfont:new_type_u(947, {-50, 0}) % \gamma
1817 mathfont:new_type_u(948, {0, 0}) % \delta
1818 mathfont:new_type_u(1013, {50, 0}) % \epsilon
1819 mathfont:new_type_u(950, {0, 0}) % \zeta
1820 mathfont:new_type_u(951, {-50, 0}) % \eta
1821 mathfont:new_type_u(952, {0, 0}) % \theta

```

```

1822 mathfont:new_type_u(953, {-50, 0}) % \iota
1823 mathfont:new_type_u(954, {0, 0})   % \kappa
1824 mathfont:new_type_u(955, {-150, 0}) % \lambda
1825 mathfont:new_type_u(956, {0, 0})   % \mu
1826 mathfont:new_type_u(957, {-50, 0}) % \nu
1827 mathfont:new_type_u(958, {0, 0})   % \xi
1828 mathfont:new_type_u(959, {0, 0})   % \omicron
1829 mathfont:new_type_u(960, {-100, 0}) % \pi
1830 mathfont:new_type_u(961, {-50, 0}) % \rho
1831 mathfont:new_type_u(963, {-100, 0}) % \sigma
1832 mathfont:new_type_u(964, {-100, 0}) % \tau
1833 mathfont:new_type_u(965, {-50, 0}) % \upsilon
1834 mathfont:new_type_u(981, {0, 0})   % \phi
1835 mathfont:new_type_u(967, {-50, 0}) % \chi
1836 mathfont:new_type_u(968, {-50, 0}) % \psi
1837 mathfont:new_type_u(969, {0, 0})   % \omega
1838 mathfont:new_type_u(976, {0, 0})   % \varbeta
1839 mathfont:new_type_u(949, {-50, 0}) % \varepsilon
1840 mathfont:new_type_u(977, {50, 0})  % \vartheta
1841 mathfont:new_type_u(1009, {-50, 0}) % \varrho
1842 mathfont:new_type_u(962, {-50, 0}) % \varsigma
1843 mathfont:new_type_u(966, {0, 0})   % \varphi

```

Upper-case Greek characters. Same as previously.

```

1844 mathfont:new_type_u(913, {0, 0})   % \Alpha
1845 mathfont:new_type_u(914, {0, 0})   % \Beta
1846 mathfont:new_type_u(915, {0, 0})   % \Gamma
1847 mathfont:new_type_u(916, {0, 0})   % \Delta
1848 mathfont:new_type_u(917, {0, 0})   % \Epsilon
1849 mathfont:new_type_u(918, {0, 0})   % \Zeta
1850 mathfont:new_type_u(919, {0, 0})   % \Eta
1851 mathfont:new_type_u(920, {0, 0})   % \Theta
1852 mathfont:new_type_u(921, {0, 0})   % \Iota
1853 mathfont:new_type_u(922, {0, 0})   % \Kappa
1854 mathfont:new_type_u(923, {0, 0})   % \Lambda
1855 mathfont:new_type_u(924, {0, 0})   % \Mu
1856 mathfont:new_type_u(925, {0, 0})   % \Nu
1857 mathfont:new_type_u(926, {0, 0})   % \Xi
1858 mathfont:new_type_u(927, {0, 0})   % \Omicron
1859 mathfont:new_type_u(928, {0, 0})   % \Pi
1860 mathfont:new_type_u(929, {0, 0})   % \Rho
1861 mathfont:new_type_u(931, {0, 0})   % \Sigma
1862 mathfont:new_type_u(932, {0, 0})   % \Tau
1863 mathfont:new_type_u(933, {0, 0})   % \Upsilon
1864 mathfont:new_type_u(934, {0, 0})   % \Phi
1865 mathfont:new_type_u(935, {0, 0})   % \Chi
1866 mathfont:new_type_u(936, {0, 0})   % \Psi
1867 mathfont:new_type_u(937, {0, 0})   % \Omega

```

```
1868 mathfont:new_type_u(1012, {0, 0}) % \varTheta
```

We add the charm information for delimiters and other resizable characters. We divide the characters into four categories depending on how we want to magnify the base glyph to create large variants: delimiters, big operators, vertical characters, and the integral sign. We automate the process by putting charm information for each category of character into a separate table and feeding the whole thing to a wrapper around `:new_type_e`.

```
1869 local delim_glyphs = {40, % (
1870 41, % )
1871 47, % /
1872 91, % [
1873 92, % backslash
1874 93, % ]
1875 123, % {
1876 125, % }
1877 8249, % \lguil
1878 8250, % \rguil
1879 171, % \llguil
1880 187, % \rrguil
1881 1044508, % \fakelangle
1882 1044509, % \fakerrangle
1883 1044510, % \fakellangle
1884 1044511} % \fakerrangle
1885 local big_op_glyphs = {33, % !
1886 35, % #
1887 36, % $
1888 37, % %
1889 38, % &
1890 43, % +
1891 63, % ?
1892 64, % @
1893 167, % \S
1894 215, % \times
1895 247, % \div
1896 8719, % \prod
1897 8721, % \sum
1898 8720, % \coprod
1899 8897, % \bigvee
1900 8896, % \bigwedge
1901 8899, % \bigcup
1902 8898, % \bigcap
1903 10753, % \bigoplus
1904 10754, % \bigotimes
1905 10752, % \bigodot
1906 10757, % \bigsqcap
1907 10758} % \bigsqcup
1908 local vert_glyphs = {124, 8730} % | and \surd
1909 local int_glyphs = {8747, % \intop
```



```

1910 8748,    % \iint
1911 8749,    % \iiint
1912 8750,    % \oint
1913 8751,    % \oiint
1914 8752}    % \oiiint

```

The variable `smash` will keep track of the unicode index used to store the smashed version of the character.

```
1915 local smash = 1044544
```

Each category of type `e` character will have its own table of charm information with different magnification values. each table is initially empty.

```

1916 local delim_scale = {}
1917 local big_op_scale = {}
1918 local vert_scale = {}
1919 local int_scale = {}

```

Populate each table with magnification information. For every type `e` character we will create fifteen larger variants in the font. Delimiters stretch mostly vertically and some horizontally. Vertical characters stretch vertically only, so their horizontal scale factors are all constant. Big operators stretch the same in vertical and horizontal directions.

```

1920 for i = 1, 15, 1 do
1921   delim_scale[2*i-1] = 1000 + 100*i % horizontal - delimiters
1922   delim_scale[2*i] = 1000 + 500*i   % vertical - delimiters
1923   vert_scale[2*i-1] = 1000
1924   vert_scale[2*i] = 1000 + 500*i    % vertical - vertically scaled chars
1925   big_op_scale[2*i-1] = 1000 + 100*i % horizontal - big operators
1926   big_op_scale[2*i] = 1000 + 100*i  % vertical - big operators

```

The integral sign is particular. Visually, we would like an integral symbol that is larger than the large operators, which means that the integral sign should have no variants between the font's value of `\Umathoperatorssize` and the desired larger size. Accordingly, I decided it would be easiest to have large variants of the integral sign jump by large enough scale factors that the smallest variant larger than the regular size is already significantly larger than the `\Umathoperatorssize` setting in `populate_math_constants`. Effectively this means that the user should take the size of the integral operator as fixed and should set `\Umathoperatorssize` to make all other big operators the desired size.

```

1927   int_scale[2*i-1] = 1000 + 500*i    % horizontal - integral sign
1928   int_scale[2*i] = 1000 + 1500*i    % vertical - integral sign
1929 end

```

We do not modify accent placement.

```

1930 delim_scale[31] = 0
1931 delim_scale[32] = 0
1932 big_op_scale[31] = 0
1933 big_op_scale[32] = 0
1934 vert_scale[31] = 0
1935 vert_scale[32] = 0
1936 int_scale[31] = 0
1937 int_scale[32] = 0

```

The wrapper for `:new_type_e`. We feed it the index to use for the smashed base character, a list of characters to create charm information for, and a table of scaling information.

```

1938 function mathfont:add_extensible_variants(first_smash, glyph_list, scale_list)
1939   local variants = (\string# scale_list - 2) / 2
1940   local curr_smash = first_smash
1941   for i = 1, \string# glyph_list, 1 do
1942     local curr_char = glyph_list[i]

```

The `curr_slots` list will hold the base-10 unicode index values of each larger variant of the base character. We will take a number of unicode slots following the smashed character equal to the number of large variants we want to create, which we stored in `variants`.

```

1943     local curr_slots = {}
1944     for j = 1, variants, 1 do
1945       curr_slots[j] = curr_smash + j
1946     end

```

Add the charm information and increment `smash`.

```

1947     self:new_type_e(curr_char, curr_smash, curr_slots, scale_list)
1948     smash = smash + variants + 1
1949     curr_smash = smash
1950   end
1951 end

```

Add the charm information for the type `e` characters.

```

1952 mathfont:add_extensible_variants(smash, delim_glyphs, delim_scale)
1953 mathfont:add_extensible_variants(smash, big_op_glyphs, big_op_scale)
1954 mathfont:add_extensible_variants(smash, vert_glyphs, vert_scale)
1955 mathfont:add_extensible_variants(smash, int_glyphs, int_scale)

```

Finally, end the call to `\directlua` and balance the preceding conditional.

```

1956 }
1957 \fi % matches previous \ifM@adjust@font

```

## 12 Unicode Hex Values

Set upper-case Latin characters. We use an `\edef` for `\M@upper@font` because every expansion now will save  $\LaTeX$  twenty-six expansions later when it evaluates each `\DeclareMathSymbol`. If the user has enabled Lua font adjustments, we set the math codes to be the large values from the Supplemental Private Use Area-A.

```

1958 \ifM@adjust@font
1959   \def\M@upper@set{%
1960     \edef\M@upper@font{M\M@uppershape\@tempa}
1961     \DeclareMathSymbol{A}{\mathalpha}{\M@upper@font}{1044512}
1962     \DeclareMathSymbol{B}{\mathalpha}{\M@upper@font}{1044513}
1963     \DeclareMathSymbol{C}{\mathalpha}{\M@upper@font}{1044514}
1964     \DeclareMathSymbol{D}{\mathalpha}{\M@upper@font}{1044515}
1965     \DeclareMathSymbol{E}{\mathalpha}{\M@upper@font}{1044516}
1966     \DeclareMathSymbol{F}{\mathalpha}{\M@upper@font}{1044517}

```

```

1967 \DeclareMathSymbol{G}{\mathalpha}{\M@upper@font}{1044518}
1968 \DeclareMathSymbol{H}{\mathalpha}{\M@upper@font}{1044519}
1969 \DeclareMathSymbol{I}{\mathalpha}{\M@upper@font}{1044520}
1970 \DeclareMathSymbol{J}{\mathalpha}{\M@upper@font}{1044521}
1971 \DeclareMathSymbol{K}{\mathalpha}{\M@upper@font}{1044522}
1972 \DeclareMathSymbol{L}{\mathalpha}{\M@upper@font}{1044523}
1973 \DeclareMathSymbol{M}{\mathalpha}{\M@upper@font}{1044524}
1974 \DeclareMathSymbol{N}{\mathalpha}{\M@upper@font}{1044525}
1975 \DeclareMathSymbol{O}{\mathalpha}{\M@upper@font}{1044526}
1976 \DeclareMathSymbol{P}{\mathalpha}{\M@upper@font}{1044527}
1977 \DeclareMathSymbol{Q}{\mathalpha}{\M@upper@font}{1044528}
1978 \DeclareMathSymbol{R}{\mathalpha}{\M@upper@font}{1044529}
1979 \DeclareMathSymbol{S}{\mathalpha}{\M@upper@font}{1044530}
1980 \DeclareMathSymbol{T}{\mathalpha}{\M@upper@font}{1044531}
1981 \DeclareMathSymbol{U}{\mathalpha}{\M@upper@font}{1044532}
1982 \DeclareMathSymbol{V}{\mathalpha}{\M@upper@font}{1044533}
1983 \DeclareMathSymbol{W}{\mathalpha}{\M@upper@font}{1044534}
1984 \DeclareMathSymbol{X}{\mathalpha}{\M@upper@font}{1044535}
1985 \DeclareMathSymbol{Y}{\mathalpha}{\M@upper@font}{1044536}
1986 \DeclareMathSymbol{Z}{\mathalpha}{\M@upper@font}{1044537}}
1987 \else
1988 \def\M@upper@set{%
1989 \edef\M@upper@font{M\M@uppershape\@tempa}
1990 \DeclareMathSymbol{A}{\mathalpha}{\M@upper@font}{`A}
1991 \DeclareMathSymbol{B}{\mathalpha}{\M@upper@font}{`B}
1992 \DeclareMathSymbol{C}{\mathalpha}{\M@upper@font}{`C}
1993 \DeclareMathSymbol{D}{\mathalpha}{\M@upper@font}{`D}
1994 \DeclareMathSymbol{E}{\mathalpha}{\M@upper@font}{`E}
1995 \DeclareMathSymbol{F}{\mathalpha}{\M@upper@font}{`F}
1996 \DeclareMathSymbol{G}{\mathalpha}{\M@upper@font}{`G}
1997 \DeclareMathSymbol{H}{\mathalpha}{\M@upper@font}{`H}
1998 \DeclareMathSymbol{I}{\mathalpha}{\M@upper@font}{`I}
1999 \DeclareMathSymbol{J}{\mathalpha}{\M@upper@font}{`J}
2000 \DeclareMathSymbol{K}{\mathalpha}{\M@upper@font}{`K}
2001 \DeclareMathSymbol{L}{\mathalpha}{\M@upper@font}{`L}
2002 \DeclareMathSymbol{M}{\mathalpha}{\M@upper@font}{`M}
2003 \DeclareMathSymbol{N}{\mathalpha}{\M@upper@font}{`N}
2004 \DeclareMathSymbol{O}{\mathalpha}{\M@upper@font}{`O}
2005 \DeclareMathSymbol{P}{\mathalpha}{\M@upper@font}{`P}
2006 \DeclareMathSymbol{Q}{\mathalpha}{\M@upper@font}{`Q}
2007 \DeclareMathSymbol{R}{\mathalpha}{\M@upper@font}{`R}
2008 \DeclareMathSymbol{S}{\mathalpha}{\M@upper@font}{`S}
2009 \DeclareMathSymbol{T}{\mathalpha}{\M@upper@font}{`T}
2010 \DeclareMathSymbol{U}{\mathalpha}{\M@upper@font}{`U}
2011 \DeclareMathSymbol{V}{\mathalpha}{\M@upper@font}{`V}
2012 \DeclareMathSymbol{W}{\mathalpha}{\M@upper@font}{`W}
2013 \DeclareMathSymbol{X}{\mathalpha}{\M@upper@font}{`X}

```

```

2014 \DeclareMathSymbol{Y}{\mathalpha}{\M@upper@font}{`Y}
2015 \DeclareMathSymbol{Z}{\mathalpha}{\M@upper@font}{`Z}}
2016 \fi

Set lower-case Latin characters.

2017 \ifM@adjust@font
2018 \def\M@lower@set{%
2019 \edef\M@lower@font{M\M@lowershape\@tempa}
2020 \DeclareMathSymbol{a}{\mathalpha}{\M@lower@font}{1044480}
2021 \DeclareMathSymbol{b}{\mathalpha}{\M@lower@font}{1044481}
2022 \DeclareMathSymbol{c}{\mathalpha}{\M@lower@font}{1044482}
2023 \DeclareMathSymbol{d}{\mathalpha}{\M@lower@font}{1044483}
2024 \DeclareMathSymbol{e}{\mathalpha}{\M@lower@font}{1044484}
2025 \DeclareMathSymbol{f}{\mathalpha}{\M@lower@font}{1044485}
2026 \DeclareMathSymbol{g}{\mathalpha}{\M@lower@font}{1044486}
2027 \DeclareMathSymbol{h}{\mathalpha}{\M@lower@font}{1044487}
2028 \DeclareMathSymbol{i}{\mathalpha}{\M@lower@font}{1044488}
2029 \DeclareMathSymbol{j}{\mathalpha}{\M@lower@font}{1044489}
2030 \DeclareMathSymbol{k}{\mathalpha}{\M@lower@font}{1044490}
2031 \DeclareMathSymbol{l}{\mathalpha}{\M@lower@font}{1044491}
2032 \DeclareMathSymbol{m}{\mathalpha}{\M@lower@font}{1044492}
2033 \DeclareMathSymbol{n}{\mathalpha}{\M@lower@font}{1044493}
2034 \DeclareMathSymbol{o}{\mathalpha}{\M@lower@font}{1044494}
2035 \DeclareMathSymbol{p}{\mathalpha}{\M@lower@font}{1044495}
2036 \DeclareMathSymbol{q}{\mathalpha}{\M@lower@font}{1044496}
2037 \DeclareMathSymbol{r}{\mathalpha}{\M@lower@font}{1044497}
2038 \DeclareMathSymbol{s}{\mathalpha}{\M@lower@font}{1044498}
2039 \DeclareMathSymbol{t}{\mathalpha}{\M@lower@font}{1044499}
2040 \DeclareMathSymbol{u}{\mathalpha}{\M@lower@font}{1044500}
2041 \DeclareMathSymbol{v}{\mathalpha}{\M@lower@font}{1044501}
2042 \DeclareMathSymbol{w}{\mathalpha}{\M@lower@font}{1044502}
2043 \DeclareMathSymbol{x}{\mathalpha}{\M@lower@font}{1044503}
2044 \DeclareMathSymbol{y}{\mathalpha}{\M@lower@font}{1044504}
2045 \DeclareMathSymbol{z}{\mathalpha}{\M@lower@font}{1044505}
2046 \DeclareMathSymbol{\imath}{\mathalpha}{\M@lower@font}{1044506}
2047 \DeclareMathSymbol{\jmath}{\mathalpha}{\M@lower@font}{1044507}
2048 \DeclareMathSymbol{\hbar}{\mathord}{\M@lower@font}{"127}}
2049 \else
2050 \def\M@lower@set{%
2051 \edef\M@lower@font{M\M@lowershape\@tempa}
2052 \DeclareMathSymbol{a}{\mathalpha}{\M@lower@font}{`a}
2053 \DeclareMathSymbol{b}{\mathalpha}{\M@lower@font}{`b}
2054 \DeclareMathSymbol{c}{\mathalpha}{\M@lower@font}{`c}
2055 \DeclareMathSymbol{d}{\mathalpha}{\M@lower@font}{`d}
2056 \DeclareMathSymbol{e}{\mathalpha}{\M@lower@font}{`e}
2057 \DeclareMathSymbol{f}{\mathalpha}{\M@lower@font}{`f}
2058 \DeclareMathSymbol{g}{\mathalpha}{\M@lower@font}{`g}
2059 \DeclareMathSymbol{h}{\mathalpha}{\M@lower@font}{`h}

```

```

2060 \DeclareMathSymbol{i}{\mathalpha}{\M@lower@font}{`i}
2061 \DeclareMathSymbol{j}{\mathalpha}{\M@lower@font}{`j}
2062 \DeclareMathSymbol{k}{\mathalpha}{\M@lower@font}{`k}
2063 \DeclareMathSymbol{l}{\mathalpha}{\M@lower@font}{`l}
2064 \DeclareMathSymbol{m}{\mathalpha}{\M@lower@font}{`m}
2065 \DeclareMathSymbol{n}{\mathalpha}{\M@lower@font}{`n}
2066 \DeclareMathSymbol{o}{\mathalpha}{\M@lower@font}{`o}
2067 \DeclareMathSymbol{p}{\mathalpha}{\M@lower@font}{`p}
2068 \DeclareMathSymbol{q}{\mathalpha}{\M@lower@font}{`q}
2069 \DeclareMathSymbol{r}{\mathalpha}{\M@lower@font}{`r}
2070 \DeclareMathSymbol{s}{\mathalpha}{\M@lower@font}{`s}
2071 \DeclareMathSymbol{t}{\mathalpha}{\M@lower@font}{`t}
2072 \DeclareMathSymbol{u}{\mathalpha}{\M@lower@font}{`u}
2073 \DeclareMathSymbol{v}{\mathalpha}{\M@lower@font}{`v}
2074 \DeclareMathSymbol{w}{\mathalpha}{\M@lower@font}{`w}
2075 \DeclareMathSymbol{x}{\mathalpha}{\M@lower@font}{`x}
2076 \DeclareMathSymbol{y}{\mathalpha}{\M@lower@font}{`y}
2077 \DeclareMathSymbol{z}{\mathalpha}{\M@lower@font}{`z}
2078 \DeclareMathSymbol{\imath}{\mathalpha}{\M@lower@font}{"131}
2079 \DeclareMathSymbol{\jmath}{\mathalpha}{\M@lower@font}{"237}
2080 \DeclareMathSymbol{\hbar}{\mathord}{\M@lower@font}{"127}}
2081 \fi

```

Set diacritics.

```

2082 \def\M@diacritics@set{%
2083 \edef\M@diacritics@font{M\M@diacriticsshape\@tempa}
2084 \DeclareMathAccent{\acute}{\mathalpha}{\M@diacritics@font}{"B4}
2085 \DeclareMathAccent{\aacute}{\mathalpha}{\M@diacritics@font}{"2DD}
2086 \DeclareMathAccent{\dot}{\mathalpha}{\M@diacritics@font}{"2D9}
2087 \DeclareMathAccent{\ddot}{\mathalpha}{\M@diacritics@font}{"A8}
2088 \DeclareMathAccent{\grave}{\mathalpha}{\M@diacritics@font}{"60}
2089 \DeclareMathAccent{\breve}{\mathalpha}{\M@diacritics@font}{"2D8}
2090 \DeclareMathAccent{\hat}{\mathalpha}{\M@diacritics@font}{"2C6}
2091 \DeclareMathAccent{\check}{\mathalpha}{\M@diacritics@font}{"2C7}
2092 \DeclareMathAccent{\bar}{\mathalpha}{\M@diacritics@font}{"2C9}
2093 \DeclareMathAccent{\mathring}{\mathalpha}{\M@diacritics@font}{"2DA}
2094 \DeclareMathAccent{\tilde}{\mathalpha}{\M@diacritics@font}{"2DC}}

```

Set capital Greek characters.

```

2095 \def\M@greekupper@set{%
2096 \edef\M@greekupper@font{M\M@greekuppershape\@tempa}
2097 \DeclareMathSymbol{\Alpha}{\mathalpha}{\M@greekupper@font}{"391}
2098 \DeclareMathSymbol{\Beta}{\mathalpha}{\M@greekupper@font}{"392}
2099 \DeclareMathSymbol{\Gamma}{\mathalpha}{\M@greekupper@font}{"393}
2100 \DeclareMathSymbol{\Delta}{\mathalpha}{\M@greekupper@font}{"394}
2101 \DeclareMathSymbol{\Epsilon}{\mathalpha}{\M@greekupper@font}{"395}
2102 \DeclareMathSymbol{\Zeta}{\mathalpha}{\M@greekupper@font}{"396}
2103 \DeclareMathSymbol{\Eta}{\mathalpha}{\M@greekupper@font}{"397}
2104 \DeclareMathSymbol{\Theta}{\mathalpha}{\M@greekupper@font}{"398}

```

```

2105 \DeclareMathSymbol{\Iota}{\mathalpha}{\M@greekupper@font}{"399}
2106 \DeclareMathSymbol{\Kappa}{\mathalpha}{\M@greekupper@font}{"39A}
2107 \DeclareMathSymbol{\Lambda}{\mathalpha}{\M@greekupper@font}{"39B}
2108 \DeclareMathSymbol{\Mu}{\mathalpha}{\M@greekupper@font}{"39C}
2109 \DeclareMathSymbol{\Nu}{\mathalpha}{\M@greekupper@font}{"39D}
2110 \DeclareMathSymbol{\Xi}{\mathalpha}{\M@greekupper@font}{"39E}
2111 \DeclareMathSymbol{\Omicron}{\mathalpha}{\M@greekupper@font}{"39F}
2112 \DeclareMathSymbol{\Pi}{\mathalpha}{\M@greekupper@font}{"3A0}
2113 \DeclareMathSymbol{\Rho}{\mathalpha}{\M@greekupper@font}{"3A1}
2114 \DeclareMathSymbol{\Sigma}{\mathalpha}{\M@greekupper@font}{"3A3}
2115 \DeclareMathSymbol{\Tau}{\mathalpha}{\M@greekupper@font}{"3A4}
2116 \DeclareMathSymbol{\Upsilon}{\mathalpha}{\M@greekupper@font}{"3A5}
2117 \DeclareMathSymbol{\Phi}{\mathalpha}{\M@greekupper@font}{"3A6}
2118 \DeclareMathSymbol{\Chi}{\mathalpha}{\M@greekupper@font}{"3A7}
2119 \DeclareMathSymbol{\Psi}{\mathalpha}{\M@greekupper@font}{"3A8}
2120 \DeclareMathSymbol{\Omega}{\mathalpha}{\M@greekupper@font}{"3A9}
2121 \DeclareMathSymbol{\varTheta}{\mathalpha}{\M@greekupper@font}{"3F4}

```

Declare `\increment` and `\nabla` if they haven't already been declared in the symbols or `extsymbols` fonts.

```

2122 \ifM@adjust@font
2123   \ifM@symbols\else
2124     \DeclareMathSymbol{\increment}{\mathord}{\M@greekupper@font}{"2206}
2125     \DeclareMathSymbol{\nabla}{\mathord}{\M@greekupper@font}{"2207}
2126   \fi
2127 \else
2128   \ifM@symbols\else
2129     \DeclareMathSymbol{\increment}{\mathord}{\M@greekupper@font}{"2206}
2130   \fi
2131   \ifM@extsymbols\else
2132     \DeclareMathSymbol{\nabla}{\mathord}{\M@greekupper@font}{"2207}
2133   \fi
2134 \fi}

```

Set minuscule Greek characters.

```

2135 \def\M@greeklower@set{%
2136   \edef\M@greeklower@font{M\M@greeklower@shape\@tempa}
2137   \DeclareMathSymbol{\alpha}{\mathalpha}{\M@greeklower@font}{"3B1}
2138   \DeclareMathSymbol{\beta}{\mathalpha}{\M@greeklower@font}{"3B2}
2139   \DeclareMathSymbol{\gamma}{\mathalpha}{\M@greeklower@font}{"3B3}
2140   \DeclareMathSymbol{\delta}{\mathalpha}{\M@greeklower@font}{"3B4}
2141   \DeclareMathSymbol{\epsilon}{\mathalpha}{\M@greeklower@font}{"3B5}
2142   \DeclareMathSymbol{\zeta}{\mathalpha}{\M@greeklower@font}{"3B6}
2143   \DeclareMathSymbol{\eta}{\mathalpha}{\M@greeklower@font}{"3B7}
2144   \DeclareMathSymbol{\theta}{\mathalpha}{\M@greeklower@font}{"3B8}
2145   \DeclareMathSymbol{\iota}{\mathalpha}{\M@greeklower@font}{"3B9}
2146   \DeclareMathSymbol{\kappa}{\mathalpha}{\M@greeklower@font}{"3BA}
2147   \DeclareMathSymbol{\lambda}{\mathalpha}{\M@greeklower@font}{"3BB}
2148   \DeclareMathSymbol{\mu}{\mathalpha}{\M@greeklower@font}{"3BC}

```

```

2149 \DeclareMathSymbol{\nu}{\mathalpha}{\M@greeklower@font}{"3BD}
2150 \DeclareMathSymbol{\xi}{\mathalpha}{\M@greeklower@font}{"3BE}
2151 \DeclareMathSymbol{\omicron}{\mathalpha}{\M@greeklower@font}{"3BF}
2152 \DeclareMathSymbol{\pi}{\mathalpha}{\M@greeklower@font}{"3C0}
2153 \DeclareMathSymbol{\rho}{\mathalpha}{\M@greeklower@font}{"3C1}
2154 \DeclareMathSymbol{\sigma}{\mathalpha}{\M@greeklower@font}{"3C3}
2155 \DeclareMathSymbol{\tau}{\mathalpha}{\M@greeklower@font}{"3C4}
2156 \DeclareMathSymbol{\upsilon}{\mathalpha}{\M@greeklower@font}{"3C5}
2157 \DeclareMathSymbol{\phi}{\mathalpha}{\M@greeklower@font}{"3C6}
2158 \DeclareMathSymbol{\chi}{\mathalpha}{\M@greeklower@font}{"3C7}
2159 \DeclareMathSymbol{\psi}{\mathalpha}{\M@greeklower@font}{"3C8}
2160 \DeclareMathSymbol{\omega}{\mathalpha}{\M@greeklower@font}{"3C9}
2161 \DeclareMathSymbol{\varbeta}{\mathalpha}{\M@greeklower@font}{"3D0}
2162 \DeclareMathSymbol{\varepsilon}{\mathalpha}{\M@greeklower@font}{"3F5}
2163 \DeclareMathSymbol{\varkappa}{\mathalpha}{\M@greeklower@font}{"3F0}
2164 \DeclareMathSymbol{\vartheta}{\mathalpha}{\M@greeklower@font}{"3D1}
2165 \DeclareMathSymbol{\varrho}{\mathalpha}{\M@greeklower@font}{"3F1}
2166 \DeclareMathSymbol{\varsigma}{\mathalpha}{\M@greeklower@font}{"3C2}
2167 \DeclareMathSymbol{\varphi}{\mathalpha}{\M@greeklower@font}{"3D5}

```

Set capital ancient Greek characters.

```

2168 \def\M@agreekupper@set{%
2169 \edef\M@agreekupper@font{M\M@agreekupper@shape\@tempa}
2170 \DeclareMathSymbol{\Heta}{\mathalpha}{\M@agreekupper@font}{"370}
2171 \DeclareMathSymbol{\Sampi}{\mathalpha}{\M@agreekupper@font}{"3E0}
2172 \DeclareMathSymbol{\Digamma}{\mathalpha}{\M@agreekupper@font}{"3DC}
2173 \DeclareMathSymbol{\Koppa}{\mathalpha}{\M@agreekupper@font}{"3D8}
2174 \DeclareMathSymbol{\Stigma}{\mathalpha}{\M@agreekupper@font}{"3DA}
2175 \DeclareMathSymbol{\Sho}{\mathalpha}{\M@agreekupper@font}{"3F7}
2176 \DeclareMathSymbol{\San}{\mathalpha}{\M@agreekupper@font}{"3FA}
2177 \DeclareMathSymbol{\varSampi}{\mathalpha}{\M@agreekupper@font}{"372}
2178 \DeclareMathSymbol{\varDigamma}{\mathalpha}{\M@agreekupper@font}{"376}
2179 \DeclareMathSymbol{\varKoppa}{\mathalpha}{\M@agreekupper@font}{"3DE}}

```

Set minuscule ancient Greek characters.

```

2180 \def\M@greeklower@set{%
2181 \edef\M@greeklower@font{M\M@greeklower@shape\@tempa}
2182 \DeclareMathSymbol{\heta}{\mathalpha}{\M@greeklower@font}{"371}
2183 \DeclareMathSymbol{\sampi}{\mathalpha}{\M@greeklower@font}{"3E1}
2184 \DeclareMathSymbol{\digamma}{\mathalpha}{\M@greeklower@font}{"3DD}
2185 \DeclareMathSymbol{\koppa}{\mathalpha}{\M@greeklower@font}{"3D9}
2186 \DeclareMathSymbol{\stigma}{\mathalpha}{\M@greeklower@font}{"3DB}
2187 \DeclareMathSymbol{\sho}{\mathalpha}{\M@greeklower@font}{"3F8}
2188 \DeclareMathSymbol{\san}{\mathalpha}{\M@greeklower@font}{"3FB}
2189 \DeclareMathSymbol{\varsampi}{\mathalpha}{\M@greeklower@font}{"373}
2190 \DeclareMathSymbol{\vardigamma}{\mathalpha}{\M@greeklower@font}{"377}
2191 \DeclareMathSymbol{\varkoppa}{\mathalpha}{\M@greeklower@font}{"3DF}}

```

Set capital Cyrillic characters.

```

2192 \def\M@cyrillicupper@set{%
2193   \edef\M@cyrillicupper@font{M\M@cyrillicupper@shape\@tempa}
2194   \DeclareMathSymbol{\cyrA}{\mathalpha}{\M@cyrillicupper@font}{"410}
2195   \DeclareMathSymbol{\cyrBe}{\mathalpha}{\M@cyrillicupper@font}{"411}
2196   \DeclareMathSymbol{\cyrVe}{\mathalpha}{\M@cyrillicupper@font}{"412}
2197   \DeclareMathSymbol{\cyrGhe}{\mathalpha}{\M@cyrillicupper@font}{"413}
2198   \DeclareMathSymbol{\cyrDe}{\mathalpha}{\M@cyrillicupper@font}{"414}
2199   \DeclareMathSymbol{\cyrIe}{\mathalpha}{\M@cyrillicupper@font}{"415}
2200   \DeclareMathSymbol{\cyrZhe}{\mathalpha}{\M@cyrillicupper@font}{"416}
2201   \DeclareMathSymbol{\cyrZe}{\mathalpha}{\M@cyrillicupper@font}{"417}
2202   \DeclareMathSymbol{\cyrI}{\mathalpha}{\M@cyrillicupper@font}{"418}
2203   \DeclareMathSymbol{\cyrKa}{\mathalpha}{\M@cyrillicupper@font}{"41A}
2204   \DeclareMathSymbol{\cyrEl}{\mathalpha}{\M@cyrillicupper@font}{"41B}
2205   \DeclareMathSymbol{\cyrEm}{\mathalpha}{\M@cyrillicupper@font}{"41C}
2206   \DeclareMathSymbol{\cyrEn}{\mathalpha}{\M@cyrillicupper@font}{"41D}
2207   \DeclareMathSymbol{\cyrO}{\mathalpha}{\M@cyrillicupper@font}{"41E}
2208   \DeclareMathSymbol{\cyrPe}{\mathalpha}{\M@cyrillicupper@font}{"41F}
2209   \DeclareMathSymbol{\cyrEr}{\mathalpha}{\M@cyrillicupper@font}{"420}
2210   \DeclareMathSymbol{\cyrEs}{\mathalpha}{\M@cyrillicupper@font}{"421}
2211   \DeclareMathSymbol{\cyrTe}{\mathalpha}{\M@cyrillicupper@font}{"422}
2212   \DeclareMathSymbol{\cyrU}{\mathalpha}{\M@cyrillicupper@font}{"423}
2213   \DeclareMathSymbol{\cyrEf}{\mathalpha}{\M@cyrillicupper@font}{"424}
2214   \DeclareMathSymbol{\cyrHa}{\mathalpha}{\M@cyrillicupper@font}{"425}
2215   \DeclareMathSymbol{\cyrTse}{\mathalpha}{\M@cyrillicupper@font}{"426}
2216   \DeclareMathSymbol{\cyrChe}{\mathalpha}{\M@cyrillicupper@font}{"427}
2217   \DeclareMathSymbol{\cyrSha}{\mathalpha}{\M@cyrillicupper@font}{"428}
2218   \DeclareMathSymbol{\cyrShcha}{\mathalpha}{\M@cyrillicupper@font}{"429}
2219   \DeclareMathSymbol{\cyrHard}{\mathalpha}{\M@cyrillicupper@font}{"42A}
2220   \DeclareMathSymbol{\cyrYeru}{\mathalpha}{\M@cyrillicupper@font}{"42B}
2221   \DeclareMathSymbol{\cyrSoft}{\mathalpha}{\M@cyrillicupper@font}{"42C}
2222   \DeclareMathSymbol{\cyrE}{\mathalpha}{\M@cyrillicupper@font}{"42D}
2223   \DeclareMathSymbol{\cyrYu}{\mathalpha}{\M@cyrillicupper@font}{"42E}
2224   \DeclareMathSymbol{\cyrYa}{\mathalpha}{\M@cyrillicupper@font}{"42F}
2225   \DeclareMathSymbol{\cyrvarI}{\mathalpha}{\M@cyrillicupper@font}{"419}}

```

Set minuscule Cyrillic characters.

```

2226 \def\M@cyrilliclower@set{%
2227   \edef\M@cyrilliclower@font{M\M@cyrilliclower@shape\@tempa}
2228   \DeclareMathSymbol{\cyra}{\mathalpha}{\M@cyrilliclower@font}{"430}
2229   \DeclareMathSymbol{\cyrbe}{\mathalpha}{\M@cyrilliclower@font}{"431}
2230   \DeclareMathSymbol{\cyrve}{\mathalpha}{\M@cyrilliclower@font}{"432}
2231   \DeclareMathSymbol{\cyrghhe}{\mathalpha}{\M@cyrilliclower@font}{"433}
2232   \DeclareMathSymbol{\cyrde}{\mathalpha}{\M@cyrilliclower@font}{"434}
2233   \DeclareMathSymbol{\cyrIe}{\mathalpha}{\M@cyrilliclower@font}{"435}
2234   \DeclareMathSymbol{\cyrzhe}{\mathalpha}{\M@cyrilliclower@font}{"436}
2235   \DeclareMathSymbol{\cyrze}{\mathalpha}{\M@cyrilliclower@font}{"437}
2236   \DeclareMathSymbol{\cyri}{\mathalpha}{\M@cyrilliclower@font}{"438}
2237   \DeclareMathSymbol{\cyrka}{\mathalpha}{\M@cyrilliclower@font}{"43A}

```



```

2238 \DeclareMathSymbol{\cyrel}{\mathalpha}{\M@cyrilliclower@font}{"43B}
2239 \DeclareMathSymbol{\cyrem}{\mathalpha}{\M@cyrilliclower@font}{"43C}
2240 \DeclareMathSymbol{\cyren}{\mathalpha}{\M@cyrilliclower@font}{"43D}
2241 \DeclareMathSymbol{\cyro}{\mathalpha}{\M@cyrilliclower@font}{"43E}
2242 \DeclareMathSymbol{\cyrpe}{\mathalpha}{\M@cyrilliclower@font}{"43F}
2243 \DeclareMathSymbol{\cyrer}{\mathalpha}{\M@cyrilliclower@font}{"440}
2244 \DeclareMathSymbol{\cyres}{\mathalpha}{\M@cyrilliclower@font}{"441}
2245 \DeclareMathSymbol{\cyrte}{\mathalpha}{\M@cyrilliclower@font}{"442}
2246 \DeclareMathSymbol{\cyrur}{\mathalpha}{\M@cyrilliclower@font}{"443}
2247 \DeclareMathSymbol{\cyref}{\mathalpha}{\M@cyrilliclower@font}{"444}
2248 \DeclareMathSymbol{\cyrha}{\mathalpha}{\M@cyrilliclower@font}{"445}
2249 \DeclareMathSymbol{\cyrtese}{\mathalpha}{\M@cyrilliclower@font}{"446}
2250 \DeclareMathSymbol{\cyrche}{\mathalpha}{\M@cyrilliclower@font}{"447}
2251 \DeclareMathSymbol{\cyrsha}{\mathalpha}{\M@cyrilliclower@font}{"448}
2252 \DeclareMathSymbol{\cyrshcha}{\mathalpha}{\M@cyrilliclower@font}{"449}
2253 \DeclareMathSymbol{\cyrhard}{\mathalpha}{\M@cyrilliclower@font}{"44A}
2254 \DeclareMathSymbol{\cyryeru}{\mathalpha}{\M@cyrilliclower@font}{"44B}
2255 \DeclareMathSymbol{\cyrsoft}{\mathalpha}{\M@cyrilliclower@font}{"44C}
2256 \DeclareMathSymbol{\cyre}{\mathalpha}{\M@cyrilliclower@font}{"44D}
2257 \DeclareMathSymbol{\cyruru}{\mathalpha}{\M@cyrilliclower@font}{"44E}
2258 \DeclareMathSymbol{\cyrura}{\mathalpha}{\M@cyrilliclower@font}{"44F}
2259 \DeclareMathSymbol{\cyrvari}{\mathalpha}{\M@cyrilliclower@font}{"439}

```

Set Hebrew characters.

```

2260 \def\M@hebrew@set{%
2261   \edef\M@hebrew@font{M\M@hebrewshape\@tempa}
2262   \DeclareMathSymbol{\aleph}{\mathalpha}{\M@hebrew@font}{"5D0}
2263   \DeclareMathSymbol{\beth}{\mathalpha}{\M@hebrew@font}{"5D1}
2264   \DeclareMathSymbol{\gimel}{\mathalpha}{\M@hebrew@font}{"5D2}
2265   \DeclareMathSymbol{\daleth}{\mathalpha}{\M@hebrew@font}{"5D3}
2266   \DeclareMathSymbol{\he}{\mathalpha}{\M@hebrew@font}{"5D4}
2267   \DeclareMathSymbol{\vav}{\mathalpha}{\M@hebrew@font}{"5D5}
2268   \DeclareMathSymbol{\zayin}{\mathalpha}{\M@hebrew@font}{"5D6}
2269   \DeclareMathSymbol{\het}{\mathalpha}{\M@hebrew@font}{"5D7}
2270   \DeclareMathSymbol{\tet}{\mathalpha}{\M@hebrew@font}{"5D8}
2271   \DeclareMathSymbol{\yod}{\mathalpha}{\M@hebrew@font}{"5D9}
2272   \DeclareMathSymbol{\kaf}{\mathalpha}{\M@hebrew@font}{"5DB}
2273   \DeclareMathSymbol{\lamed}{\mathalpha}{\M@hebrew@font}{"5DC}
2274   \DeclareMathSymbol{\mem}{\mathalpha}{\M@hebrew@font}{"5DE}
2275   \DeclareMathSymbol{\nun}{\mathalpha}{\M@hebrew@font}{"5E0}
2276   \DeclareMathSymbol{\samekh}{\mathalpha}{\M@hebrew@font}{"5E1}
2277   \DeclareMathSymbol{\ayin}{\mathalpha}{\M@hebrew@font}{"5E2}
2278   \DeclareMathSymbol{\pe}{\mathalpha}{\M@hebrew@font}{"5E4}
2279   \DeclareMathSymbol{\tsadi}{\mathalpha}{\M@hebrew@font}{"5E6}
2280   \DeclareMathSymbol{\qof}{\mathalpha}{\M@hebrew@font}{"5E7}
2281   \DeclareMathSymbol{\resh}{\mathalpha}{\M@hebrew@font}{"5E8}
2282   \DeclareMathSymbol{\shin}{\mathalpha}{\M@hebrew@font}{"5E9}
2283   \DeclareMathSymbol{\tav}{\mathalpha}{\M@hebrew@font}{"5EA}

```

```

2284 \DeclareMathSymbol{\varkaf}{\mathalpha}{\M@hebrew@font}{"5DA}
2285 \DeclareMathSymbol{\varmem}{\mathalpha}{\M@hebrew@font}{"5DD}
2286 \DeclareMathSymbol{\varnun}{\mathalpha}{\M@hebrew@font}{"5DF}
2287 \DeclareMathSymbol{\varpe}{\mathalpha}{\M@hebrew@font}{"5E3}
2288 \DeclareMathSymbol{\vartsadi}{\mathalpha}{\M@hebrew@font}{"5E5}}

```

Set digits.

```

2289 \def\M@digits@set{%
2290 \edef\M@digits@font{M\M@digitsshape\@tempa}
2291 \DeclareMathSymbol{0}{\mathalpha}{\M@digits@font}{`0}
2292 \DeclareMathSymbol{1}{\mathalpha}{\M@digits@font}{`1}
2293 \DeclareMathSymbol{2}{\mathalpha}{\M@digits@font}{`2}
2294 \DeclareMathSymbol{3}{\mathalpha}{\M@digits@font}{`3}
2295 \DeclareMathSymbol{4}{\mathalpha}{\M@digits@font}{`4}
2296 \DeclareMathSymbol{5}{\mathalpha}{\M@digits@font}{`5}
2297 \DeclareMathSymbol{6}{\mathalpha}{\M@digits@font}{`6}
2298 \DeclareMathSymbol{7}{\mathalpha}{\M@digits@font}{`7}
2299 \DeclareMathSymbol{8}{\mathalpha}{\M@digits@font}{`8}
2300 \DeclareMathSymbol{9}{\mathalpha}{\M@digits@font}{`9}}

```

Set new operator font. If `mathfont` is set to adjust fonts, we will have a problem when typesetting operators because the `\operator@font` will pull modified (lengthened) letters from the operator font. Traditional T<sub>E</sub>X addressed this problem by storing the Latin letters for math in the same encoding slots but in a different font from Computer Modern Roman and switching to Computer Modern Roman. Here we want to use the same font but different encoding slots. The macro `\M@default@latin` changes all `\Umathcodes` of Latin letters from their big (lengthened) values to their original values. Because `\operator@font` is always called inside a group, we don't have to worry about messing up any other math.

```

2301 \def\M@operator@set{%
2302 \ifM@adjust@font
2303 \edef\M@operator@num{\number\csname symM\M@operatorshape\@tempa\endcsname}
2304 \protected\edef\M@operator@mathcodes{%
2305 \Umathcode`A=7+\M@operator@num+`A\relax
2306 \Umathcode`B=7+\M@operator@num+`B\relax
2307 \Umathcode`C=7+\M@operator@num+`C\relax
2308 \Umathcode`D=7+\M@operator@num+`D\relax
2309 \Umathcode`E=7+\M@operator@num+`E\relax
2310 \Umathcode`F=7+\M@operator@num+`F\relax
2311 \Umathcode`G=7+\M@operator@num+`G\relax
2312 \Umathcode`H=7+\M@operator@num+`H\relax
2313 \Umathcode`I=7+\M@operator@num+`I\relax
2314 \Umathcode`J=7+\M@operator@num+`J\relax
2315 \Umathcode`K=7+\M@operator@num+`K\relax
2316 \Umathcode`L=7+\M@operator@num+`L\relax
2317 \Umathcode`M=7+\M@operator@num+`M\relax
2318 \Umathcode`N=7+\M@operator@num+`N\relax
2319 \Umathcode`O=7+\M@operator@num+`O\relax
2320 \Umathcode`P=7+\M@operator@num+`P\relax

```

```

2321 \Umathcode`Q=7+\M@operator@num+`Q\relax
2322 \Umathcode`R=7+\M@operator@num+`R\relax
2323 \Umathcode`S=7+\M@operator@num+`S\relax
2324 \Umathcode`T=7+\M@operator@num+`T\relax
2325 \Umathcode`U=7+\M@operator@num+`U\relax
2326 \Umathcode`V=7+\M@operator@num+`V\relax
2327 \Umathcode`W=7+\M@operator@num+`W\relax
2328 \Umathcode`X=7+\M@operator@num+`X\relax
2329 \Umathcode`Y=7+\M@operator@num+`Y\relax
2330 \Umathcode`Z=7+\M@operator@num+`Z\relax
2331 \Umathcode`a=7+\M@operator@num+`a\relax
2332 \Umathcode`b=7+\M@operator@num+`b\relax
2333 \Umathcode`c=7+\M@operator@num+`c\relax
2334 \Umathcode`d=7+\M@operator@num+`d\relax
2335 \Umathcode`e=7+\M@operator@num+`e\relax
2336 \Umathcode`f=7+\M@operator@num+`f\relax
2337 \Umathcode`g=7+\M@operator@num+`g\relax
2338 \Umathcode`h=7+\M@operator@num+`h\relax
2339 \Umathcode`i=7+\M@operator@num+`i\relax
2340 \Umathcode`j=7+\M@operator@num+`j\relax
2341 \Umathcode`k=7+\M@operator@num+`k\relax
2342 \Umathcode`l=7+\M@operator@num+`l\relax
2343 \Umathcode`m=7+\M@operator@num+`m\relax
2344 \Umathcode`n=7+\M@operator@num+`n\relax
2345 \Umathcode`o=7+\M@operator@num+`o\relax
2346 \Umathcode`p=7+\M@operator@num+`p\relax
2347 \Umathcode`q=7+\M@operator@num+`q\relax
2348 \Umathcode`r=7+\M@operator@num+`r\relax
2349 \Umathcode`s=7+\M@operator@num+`s\relax
2350 \Umathcode`t=7+\M@operator@num+`t\relax
2351 \Umathcode`u=7+\M@operator@num+`u\relax
2352 \Umathcode`v=7+\M@operator@num+`v\relax
2353 \Umathcode`w=7+\M@operator@num+`w\relax
2354 \Umathcode`x=7+\M@operator@num+`x\relax
2355 \Umathcode`y=7+\M@operator@num+`y\relax
2356 \Umathcode`z=7+\M@operator@num+`z\relax
2357 \Umathchardef\imath=7+\M@operator@num+1044506\relax
2358 \Umathchardef\jmath=7+\M@operator@num+1044500\relax}
2359 \else
2360 \let\M@operator@mathcodes\@empty
2361 \fi

```

Then we change the `\operator@font` definition and, if applicable, change the math codes.

```

2362 \xdef\operator@font{\noexpand\mathgroup
2363 \csname symM\M@operatorshape\@tempa\endcsname\M@operator@mathcodes}}

```

Set delimiters.

```

2364 \ifM@adjust@font
2365 \def\M@delimiters@set{%

```

```

2366 \edef\M@delimiters@font{M\M@delimitersshape\@tempa}
2367 \DeclareMathSymbol{(\mathopen}{\M@delimiters@font}{"28}
2368 \DeclareMathSymbol{)}{\mathclose}{\M@delimiters@font}{"29}
2369 \DeclareMathSymbol{[}{\mathopen}{\M@delimiters@font}{"5B}
2370 \DeclareMathSymbol{]}\mathclose}{\M@delimiters@font}{"5D}
2371 \ifM@symbols\else
2372   \DeclareMathSymbol{|}{\mathord}{\M@delimiters@font}{"7C}
2373 \fi
2374 \DeclareMathSymbol{\leftbrace}{\mathopen}{\M@delimiters@font}{"7B}
2375 \DeclareMathSymbol{\rightbrace}{\mathclose}{\M@delimiters@font}{"7D}
2376 \global\Udelcode40+=\csname sym\M@delimiters@font\endcsname+40\relax % (
2377 \global\Udelcode41+=\csname sym\M@delimiters@font\endcsname+41\relax % )
2378 \global\Udelcode47+=\csname sym\M@delimiters@font\endcsname+47\relax % /
2379 \global\Udelcode91+=\csname sym\M@delimiters@font\endcsname+91\relax % [
2380 \global\Udelcode93+=\csname sym\M@delimiters@font\endcsname+93\relax % ]
2381 \global\Udelcode124+=\csname sym\M@delimiters@font\endcsname+124\relax % |
2382 \global\let\vert=|
2383 \protected\gdef\backslash{\ifmmode\mathbackslash\else\textbackslash\fi}
2384 \protected\xdef\mathbackslash{%
2385   \Udelimiter+2+\number\csname sym\M@delimiters@font\endcsname
2386     +92\relax} % backslash
2387 \protected\xdef\lbrace{%
2388   \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname
2389     +123\relax} % {
2390 \protected\xdef\rbrace{%
2391   \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
2392     +125\relax} % }
2393 \protected\xdef\lguil{%
2394   \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname
2395     +8249\relax} % single left guilement
2396 \protected\xdef\rguil{%
2397   \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
2398     +8250\relax} % single right guilement
2399 \protected\xdef\llguil{%
2400   \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname
2401     +171\relax} % double left guilement
2402 \protected\xdef\rrguil{%
2403   \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
2404     +187\relax} % double right guilement
2405 \protected\xdef\fakelangle{%
2406   \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname
2407     +1044508\relax} % fake left angle
2408 \protected\xdef\fakerangle{%
2409   \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
2410     +1044509\relax} % fake right angle
2411 \protected\xdef\fakellangle{%
2412   \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname

```

```

2413     +1044510\relax} % fake double left angle
2414 \protected\xdef\fakerrangle{%
2415   \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
2416     +1044511\relax} % fake double right angle
2417   }
2418 \else
2419   \def\M@delimiters@set{%
2420     \edef\M@delimiters@font{M\M@delimitersshape\@tempa}
2421     \DeclareMathSymbol{\}\{\mathopen}\{\M@delimiters@font}\{"28}
2422     \DeclareMathSymbol{\}\{\mathclose}\{\M@delimiters@font}\{"29}
2423     \DeclareMathSymbol{\}\{\mathopen}\{\M@delimiters@font}\{"5B}
2424     \DeclareMathSymbol{\}\{\mathclose}\{\M@delimiters@font}\{"5D}
2425     \DeclareMathSymbol{\lguil}\{\mathopen}\{\M@delimiters@font}\{"2039}
2426     \DeclareMathSymbol{\rguil}\{\mathclose}\{\M@delimiters@font}\{"203A}
2427     \DeclareMathSymbol{\llguil}\{\mathopen}\{\M@delimiters@font}\{"AB}
2428     \DeclareMathSymbol{\rrguil}\{\mathclose}\{\M@delimiters@font}\{"BB}
2429     \DeclareMathSymbol{\leftbrace}\{\mathopen}\{\M@delimiters@font}\{"7B}
2430     \DeclareMathSymbol{\rightbrace}\{\mathclose}\{\M@delimiters@font}\{"7D}}
2431 \fi

```

Radicals.

```

2432 \ifM@adjust@font
2433   \def\M@radical@set{%
2434     \edef\M@radical@font{M\M@radicalshape\@tempa}
2435     \DeclareMathSymbol{\surd}\{\mathord}\{\M@radical@font}\{"221A}
2436     \xdef\@sqrts@gn##1{%
2437       \Uradical+\number\csname sym\M@radical@font\endcsname+8730\relax{##1}}

```

We redefine `\r@t`, which typesets the degree symbol on an  $n$ th root. We set the placement so that right side of the box containing the degree lies 60% of the horizontal distance across the surd symbol, and the baseline of the degree symbol is 60% of the vertical distance up the surd.

```

2438   \gdef\r@t##1##2{%
2439     \setbox\z@ \hbox{\m@th##1\sqrtsign{##2}$}%
2440     \setbox\surdbox \hbox{\m@th##1\@sqrts@gn{
2441       \hbox{\vphantom{\m@th##1##2$}}}$}
2442     \dimen@ \ht\surdbox
2443     \advance\dimen@ \dp\surdbox
2444     \dimen@ = 0.6\dimen@
2445     \advance\dimen@ - \dp\surdbox
2446     \ifdim\wd\rootbox < 0.6\wd\surdbox
2447       \kern 0.6\wd\surdbox
2448     \else
2449       \kern\wd\rootbox
2450     \fi
2451     \raise\dimen@ \hbox{\llap{\copy\rootbox}}
2452     \kern -0.6\wd\surdbox
2453     \box\z@}
2454   \gdef\sqrtsign##1{\@sqrts@gn{\mkern\radicandoffset##1}}

```

```

2455 \else
2456   \def\M@radical@set{%
2457     \edef\M@radical@font{M\M@radicalshape\@tempa}
2458     \DeclareMathSymbol{\surd}{\mathord}{\M@radical@font}{"221A}}
2459 \fi

```

Big operators.

```

2460 \def\M@bigops@set{%
2461   \edef\M@bigops@font{M\M@bigopsshape\@tempa}
2462   \let\sum\@undefined
2463   \let\prod\@undefined
2464   \DeclareMathSymbol{\sum}{\mathop}{\M@bigops@font}{"2211}
2465   \DeclareMathSymbol{\prod}{\mathop}{\M@bigops@font}{"220F}
2466   \DeclareMathSymbol{\intop}{\mathop}{\M@bigops@font}{"222B}}

```

Extended big operators.

```

2467 \def\M@extbigops@set{%
2468   \edef\M@extbigops@font{M\M@extbigopsshape\@tempa}
2469   \let\coprod\@undefined
2470   \let\bigvee\@undefined
2471   \let\bigwedge\@undefined
2472   \let\bigcup\@undefined
2473   \let\bigcap\@undefined
2474   \let\bigoplus\@undefined
2475   \let\bigotimes\@undefined
2476   \let\bigodot\@undefined
2477   \let\bigsqcup\@undefined
2478   \DeclareMathSymbol{\coprod}{\mathop}{\M@extbigops@font}{"2210}
2479   \DeclareMathSymbol{\bigvee}{\mathop}{\M@extbigops@font}{"22C1}
2480   \DeclareMathSymbol{\bigwedge}{\mathop}{\M@extbigops@font}{"22C0}
2481   \DeclareMathSymbol{\bigcup}{\mathop}{\M@extbigops@font}{"22C3}
2482   \DeclareMathSymbol{\bigcap}{\mathop}{\M@extbigops@font}{"22C2}
2483   \DeclareMathSymbol{\iintop}{\mathop}{\M@extbigops@font}{"222C}
2484   \protected\gdef\iint{\iintop\nolimits}
2485   \DeclareMathSymbol{\iiintop}{\mathop}{\M@extbigops@font}{"222D}
2486   \protected\gdef\iiint{\iiintop\nolimits}
2487   \DeclareMathSymbol{\ointop}{\mathop}{\M@extbigops@font}{"222E}
2488   \protected\gdef\oint{\ointop\nolimits}
2489   \DeclareMathSymbol{\oiintop}{\mathop}{\M@extbigops@font}{"222F}
2490   \protected\gdef\oiint{\oiintop\nolimits}
2491   \DeclareMathSymbol{\oiiintop}{\mathop}{\M@extbigops@font}{"2230}
2492   \protected\gdef\oiiint{\oiiintop\nolimits}
2493   \DeclareMathSymbol{\bigoplus}{\mathop}{\M@extbigops@font}{"2A01}
2494   \DeclareMathSymbol{\bigotimes}{\mathop}{\M@extbigops@font}{"2A02}
2495   \DeclareMathSymbol{\bigodot}{\mathop}{\M@extbigops@font}{"2A00}
2496   \DeclareMathSymbol{\bigsqcap}{\mathop}{\M@extbigops@font}{"2A05}
2497   \DeclareMathSymbol{\bigsqcup}{\mathop}{\M@extbigops@font}{"2A06}}

```

Set symbols.

```

2498 \def\M@symbols@set{%
2499   \edef\M@symbols@font{M\M@symbols@shape\@tempa}
2500   \let\colon\@undefined
2501   \let\mathellipsis\@undefined
2502   \DeclareMathSymbol{.}{\mathord}{\M@symbols@font}{"2E}
2503   \DeclareMathSymbol{@}{\mathord}{\M@symbols@font}{"40}
2504   \DeclareMathSymbol{'}{\mathord}{\M@symbols@font}{"2032}
2505   \DeclareMathSymbol{\prime}{\mathord}{\M@symbols@font}{"2032}
2506   \DeclareMathSymbol{"}{\mathord}{\M@symbols@font}{"2033}
2507   \DeclareMathSymbol{\mathhash}{\mathord}{\M@symbols@font}{"23}
2508   \DeclareMathSymbol{\mathdollar}{\mathord}{\M@symbols@font}{"24}
2509   \DeclareMathSymbol{\mathpercent}{\mathord}{\M@symbols@font}{"25}
2510   \DeclareMathSymbol{\mathand}{\mathord}{\M@symbols@font}{"26}
2511   \DeclareMathSymbol{\mathparagraph}{\mathord}{\M@symbols@font}{"B6}
2512   \DeclareMathSymbol{\mathsection}{\mathord}{\M@symbols@font}{"A7}
2513   \DeclareMathSymbol{\mathsterling}{\mathord}{\M@symbols@font}{"A3}
2514   \DeclareMathSymbol{\neg}{\mathord}{\M@symbols@font}{"AC}
2515   \DeclareMathSymbol{|}{\mathord}{\M@symbols@font}{"7C}
2516   \DeclareMathSymbol{\infty}{\mathord}{\M@symbols@font}{"221E}
2517   \DeclareMathSymbol{\partial}{\mathord}{\M@symbols@font}{"2202}
2518   \DeclareMathSymbol{\degree}{\mathord}{\M@symbols@font}{"B0}
2519   \DeclareMathSymbol{\increment}{\mathord}{\M@symbols@font}{"2206}
2520   \DeclareMathSymbol{\comma}{\mathord}{\M@symbols@font}{"2C}
2521   \DeclareMathSymbol{+}{\mathbin}{\M@symbols@font}{"2B}
2522   \DeclareMathSymbol{-}{\mathbin}{\M@symbols@font}{"2212}
2523   \DeclareMathSymbol{*}{\mathbin}{\M@symbols@font}{"2A}
2524   \DeclareMathSymbol{\times}{\mathbin}{\M@symbols@font}{"D7}
2525   \DeclareMathSymbol{/}{\mathord}{\M@symbols@font}{"2F}
2526   \DeclareMathSymbol{\fractionslash}{\mathord}{\M@symbols@font}{"2215}
2527   \DeclareMathSymbol{\div}{\mathbin}{\M@symbols@font}{"F7}
2528   \DeclareMathSymbol{\pm}{\mathbin}{\M@symbols@font}{"B1}
2529   \DeclareMathSymbol{\bullet}{\mathbin}{\M@symbols@font}{"2022}
2530   \DeclareMathSymbol{\dagger}{\mathbin}{\M@symbols@font}{"2020}
2531   \DeclareMathSymbol{\ddagger}{\mathbin}{\M@symbols@font}{"2021}
2532   \DeclareMathSymbol{\cdot}{\mathbin}{\M@symbols@font}{"2219}
2533   \DeclareMathSymbol{\setminus}{\mathbin}{\M@symbols@font}{"5C}
2534   \DeclareMathSymbol{=}{\mathrel}{\M@symbols@font}{"3D}
2535   \DeclareMathSymbol{<}{\mathrel}{\M@symbols@font}{"3C}
2536   \DeclareMathSymbol{>}{\mathrel}{\M@symbols@font}{"3E}
2537   \DeclareMathSymbol{\leq}{\mathrel}{\M@symbols@font}{"2264}
2538   \DeclareMathSymbol{\geq}{\mathrel}{\M@symbols@font}{"2265}
2539   \DeclareMathSymbol{\sim}{\mathrel}{\M@symbols@font}{"7E}
2540   \DeclareMathSymbol{\approx}{\mathrel}{\M@symbols@font}{"2248}
2541   \DeclareMathSymbol{\equiv}{\mathrel}{\M@symbols@font}{"2261}
2542   \DeclareMathSymbol{\mid}{\mathrel}{\M@symbols@font}{"7C}
2543   \DeclareMathSymbol{\parallel}{\mathrel}{\M@symbols@font}{"2016}
2544   \DeclareMathSymbol{:}{\mathrel}{\M@symbols@font}{"3A}

```

```

2545 \DeclareMathSymbol{?}{\mathclose}{\M@symbols@font}{"3F}
2546 \DeclareMathSymbol{!}{\mathclose}{\M@symbols@font}{"21}
2547 \DeclareMathSymbol{,}{\mathpunct}{\M@symbols@font}{"2C}
2548 \DeclareMathSymbol{;}{\mathpunct}{\M@symbols@font}{"3B}
2549 \DeclareMathSymbol{\colon}{\mathord}{\M@symbols@font}{"3A}
2550 \DeclareMathSymbol{\matheellipsis}{\mathinner}{\M@symbols@font}{"2026}

```

Now a bit of housekeeping. We redefine `\#`, `\%`, and `\&` as robust commands that expand to previously declared `\mathhash`, etc. commands in math mode and retain their standard `\char` definitions otherwise. Other commands that function in both math and horizontal modes such as `\S` or `\dag` also use this technique. Then we define macros `\cong` and `\simeq`. The last three commands defined here preserve the Computer Modern font for characters used in several math-mode symbols.

```

2551 \protected\gdef\#{\ifmmode\mathhash\else\char"23\relax\fi}
2552 \protected\gdef\%{\ifmmode\mathpercent\else\char"25\relax\fi}
2553 \protected\gdef\&{\ifmmode\mathand\else\char"26\relax\fi}
2554 \DeclareMathSymbol{@relbar}{\mathbin}{symbols}{"00}
2555 \DeclareMathSymbol{@Relbar}{\mathrel}{operators}{"3D}
2556 \DeclareMathSymbol{@verticalbar}{\mathord}{symbols}{"6A}
2557 \ifM@extsymbols\else
2558   \protected\gdef\simeq{\mathrel{\mathpalette\stack@flatrel{-}{\sim}}}}
2559   \protected\gdef\cong{\mathrel{\mathpalette\stack@flatrel{=}{\sim}}}}
2560 \fi
2561 \protected\gdef\relbar{\mathrel{\smash{@relbar}}}
2562 \protected\gdef\Relbar{\mathrel{@Relbar}}
2563 \protected\gdef\models{\mathrel{@verticalbar}\joinrel\Relbar}

```

If the user enabled Lua-based font adjustments, we declare a few more big operators for fun. For brevity, we put the `adjust@font` conditional here rather than redefining `\M@symbols@set`.

```

2564 \ifM@adjust@font
2565   \DeclareMathSymbol{\bigat}{\mathop}{\M@symbols@font}{"40}
2566   \DeclareMathSymbol{\bighash}{\mathop}{\M@symbols@font}{"23}
2567   \DeclareMathSymbol{\bigdollar}{\mathop}{\M@symbols@font}{"24}
2568   \DeclareMathSymbol{\bigpercent}{\mathop}{\M@symbols@font}{"25}
2569   \DeclareMathSymbol{\bigand}{\mathop}{\M@symbols@font}{"26}
2570   \DeclareMathSymbol{\bigplus}{\mathop}{\M@symbols@font}{"2B}
2571   \DeclareMathSymbol{\bigp}{\mathop}{\M@symbols@font}{"21}
2572   \DeclareMathSymbol{\bigq}{\mathop}{\M@symbols@font}{"3F}
2573   \DeclareMathSymbol{\bigS}{\mathop}{\M@symbols@font}{"A7}
2574   \DeclareMathSymbol{\bigtimes}{\mathop}{\M@symbols@font}{"D7}
2575   \DeclareMathSymbol{\bigdiv}{\mathop}{\M@symbols@font}{"F7}

```

Define `\nabla` if we're adjusting the font. If not, this declaration will go in `extsymbols`.

```

2576   \DeclareMathSymbol{\nabla}{\mathord}{\M@symbols@font}{"2207}
2577 \fi}

```

Set extended symbols.

```

2578 \def\M@extsymbols@set{%
2579   \edef\M@extsymbols@font{M\M@extsymbolsshape\@tempa}
2580   \let\angle\@undefined

```



```

2581 \let\simeq\@undefined
2582 \let\sqssubset\@undefined
2583 \let\sqsupset\@undefined
2584 \let\bowtie\@undefined
2585 \let\doteq\@undefined
2586 \let\neq\@undefined
2587 \DeclareMathSymbol{\wp}{\mathord}{\M@extsymbols@font}{"2118}
2588 \DeclareMathSymbol{\Re}{\mathord}{\M@extsymbols@font}{"211C}
2589 \DeclareMathSymbol{\Im}{\mathord}{\M@extsymbols@font}{"2111}
2590 \DeclareMathSymbol{\ell}{\mathord}{\M@extsymbols@font}{"2113}
2591 \DeclareMathSymbol{\forall}{\mathord}{\M@extsymbols@font}{"2200}
2592 \DeclareMathSymbol{\exists}{\mathord}{\M@extsymbols@font}{"2203}
2593 \DeclareMathSymbol{\emptyset}{\mathord}{\M@extsymbols@font}{"2205}
2594 \DeclareMathSymbol{\in}{\mathord}{\M@extsymbols@font}{"2208}
2595 \DeclareMathSymbol{\ni}{\mathord}{\M@extsymbols@font}{"220B}
2596 \DeclareMathSymbol{\mp}{\mathord}{\M@extsymbols@font}{"2213}
2597 \DeclareMathSymbol{\angle}{\mathord}{\M@extsymbols@font}{"2220}
2598 \DeclareMathSymbol{\top}{\mathord}{\M@extsymbols@font}{"22A4}
2599 \DeclareMathSymbol{\bot}{\mathord}{\M@extsymbols@font}{"22A5}
2600 \DeclareMathSymbol{\vdash}{\mathord}{\M@extsymbols@font}{"22A2}
2601 \DeclareMathSymbol{\dashv}{\mathord}{\M@extsymbols@font}{"22A3}
2602 \DeclareMathSymbol{\flat}{\mathord}{\M@extsymbols@font}{"266D}
2603 \DeclareMathSymbol{\natural}{\mathord}{\M@extsymbols@font}{"266E}
2604 \DeclareMathSymbol{\sharp}{\mathord}{\M@extsymbols@font}{"266F}
2605 \DeclareMathSymbol{\fflat}{\mathord}{\M@extsymbols@font}{"1D12B}
2606 \DeclareMathSymbol{\ssharp}{\mathord}{\M@extsymbols@font}{"1D12A}
2607 \DeclareMathSymbol{\bclubsuit}{\mathord}{\M@extsymbols@font}{"2663}
2608 \DeclareMathSymbol{\bdiamondsuit}{\mathord}{\M@extsymbols@font}{"2666}
2609 \DeclareMathSymbol{\bheartsuit}{\mathord}{\M@extsymbols@font}{"2665}
2610 \DeclareMathSymbol{\bspadesuit}{\mathord}{\M@extsymbols@font}{"2660}
2611 \DeclareMathSymbol{\wclubsuit}{\mathord}{\M@extsymbols@font}{"2667}
2612 \DeclareMathSymbol{\wdiamondsuit}{\mathord}{\M@extsymbols@font}{"2662}
2613 \DeclareMathSymbol{\wheartsuit}{\mathord}{\M@extsymbols@font}{"2661}
2614 \DeclareMathSymbol{\wspadesuit}{\mathord}{\M@extsymbols@font}{"2664}
2615 \global\let\spadesuit\bspadesuit
2616 \global\let\heartsuit\wheartsuit
2617 \global\let\diamondsuit\wdiamondsuit
2618 \global\let\clubsuit\bclubsuit
2619 \DeclareMathSymbol{\wedge}{\mathbin}{\M@extsymbols@font}{"2227}
2620 \DeclareMathSymbol{\vee}{\mathbin}{\M@extsymbols@font}{"2228}
2621 \DeclareMathSymbol{\cap}{\mathord}{\M@extsymbols@font}{"2229}
2622 \DeclareMathSymbol{\cup}{\mathbin}{\M@extsymbols@font}{"222A}
2623 \DeclareMathSymbol{\sqcap}{\mathbin}{\M@extsymbols@font}{"2293}
2624 \DeclareMathSymbol{\sqcup}{\mathbin}{\M@extsymbols@font}{"2294}
2625 \DeclareMathSymbol{\amalg}{\mathbin}{\M@extsymbols@font}{"2A3F}
2626 \DeclareMathSymbol{\wr}{\mathbin}{\M@extsymbols@font}{"2240}
2627 \DeclareMathSymbol{\ast}{\mathbin}{\M@extsymbols@font}{"2217}

```

```

2628 \DeclareMathSymbol{\star}{\mathbin}{\M@extsymbols@font}{"22C6}
2629 \DeclareMathSymbol{\diamond}{\mathbin}{\M@extsymbols@font}{"22C4}
2630 \DeclareMathSymbol{\varcdot}{\mathbin}{\M@extsymbols@font}{"22C5}
2631 \DeclareMathSymbol{\varsetminus}{\mathbin}{\M@extsymbols@font}{"2216}
2632 \DeclareMathSymbol{\oplus}{\mathbin}{\M@extsymbols@font}{"2295}
2633 \DeclareMathSymbol{\otimes}{\mathbin}{\M@extsymbols@font}{"2297}
2634 \DeclareMathSymbol{\ominus}{\mathbin}{\M@extsymbols@font}{"2296}
2635 \DeclareMathSymbol{\odiv}{\mathbin}{\M@extsymbols@font}{"2A38}
2636 \DeclareMathSymbol{\oslash}{\mathbin}{\M@extsymbols@font}{"2298}
2637 \DeclareMathSymbol{\odot}{\mathbin}{\M@extsymbols@font}{"2299}
2638 \DeclareMathSymbol{\sqplus}{\mathbin}{\M@extsymbols@font}{"229E}
2639 \DeclareMathSymbol{\sqtimes}{\mathbin}{\M@extsymbols@font}{"22A0}
2640 \DeclareMathSymbol{\sqminus}{\mathbin}{\M@extsymbols@font}{"229F}
2641 \DeclareMathSymbol{\sqdot}{\mathbin}{\M@extsymbols@font}{"22A1}
2642 \DeclareMathSymbol{\in}{\mathrel}{\M@extsymbols@font}{"2208}
2643 \DeclareMathSymbol{\ni}{\mathrel}{\M@extsymbols@font}{"220B}
2644 \DeclareMathSymbol{\subset}{\mathrel}{\M@extsymbols@font}{"2282}
2645 \DeclareMathSymbol{\supset}{\mathrel}{\M@extsymbols@font}{"2283}
2646 \DeclareMathSymbol{\subseteq}{\mathrel}{\M@extsymbols@font}{"2286}
2647 \DeclareMathSymbol{\supseteq}{\mathrel}{\M@extsymbols@font}{"2287}
2648 \DeclareMathSymbol{\sqsubset}{\mathrel}{\M@extsymbols@font}{"228F}
2649 \DeclareMathSymbol{\sqsupset}{\mathrel}{\M@extsymbols@font}{"2290}
2650 \DeclareMathSymbol{\sqsubseteq}{\mathrel}{\M@extsymbols@font}{"2291}
2651 \DeclareMathSymbol{\sqsupseteq}{\mathrel}{\M@extsymbols@font}{"2292}
2652 \DeclareMathSymbol{\triangleleft}{\mathrel}{\M@extsymbols@font}{"22B2}
2653 \DeclareMathSymbol{\triangleright}{\mathrel}{\M@extsymbols@font}{"22B3}
2654 \DeclareMathSymbol{\trianglelefteq}{\mathrel}{\M@extsymbols@font}{"22B4}
2655 \DeclareMathSymbol{\trianglerighteq}{\mathrel}{\M@extsymbols@font}{"22B5}
2656 \DeclareMathSymbol{\propto}{\mathrel}{\M@extsymbols@font}{"221D}
2657 \DeclareMathSymbol{\bowtie}{\mathrel}{\M@extsymbols@font}{"22C8}
2658 \DeclareMathSymbol{\hourglass}{\mathrel}{\M@extsymbols@font}{"29D6}
2659 \DeclareMathSymbol{\therefore}{\mathrel}{\M@extsymbols@font}{"2234}
2660 \DeclareMathSymbol{\because}{\mathrel}{\M@extsymbols@font}{"2235}
2661 \DeclareMathSymbol{\ratio}{\mathrel}{\M@extsymbols@font}{"2236}
2662 \DeclareMathSymbol{\proportion}{\mathrel}{\M@extsymbols@font}{"2237}
2663 \DeclareMathSymbol{\ll}{\mathrel}{\M@extsymbols@font}{"226A}
2664 \DeclareMathSymbol{\gg}{\mathrel}{\M@extsymbols@font}{"226B}
2665 \DeclareMathSymbol{\lll}{\mathrel}{\M@extsymbols@font}{"22D8}
2666 \DeclareMathSymbol{\ggg}{\mathrel}{\M@extsymbols@font}{"22D9}
2667 \DeclareMathSymbol{\leqq}{\mathrel}{\M@extsymbols@font}{"2266}
2668 \DeclareMathSymbol{\geqq}{\mathrel}{\M@extsymbols@font}{"2267}
2669 \DeclareMathSymbol{\lapprox}{\mathrel}{\M@extsymbols@font}{"2A85}
2670 \DeclareMathSymbol{\gapprox}{\mathrel}{\M@extsymbols@font}{"2A86}
2671 \DeclareMathSymbol{\simeq}{\mathrel}{\M@extsymbols@font}{"2243}
2672 \DeclareMathSymbol{\eqsim}{\mathrel}{\M@extsymbols@font}{"2242}
2673 \DeclareMathSymbol{\simeqq}{\mathrel}{\M@extsymbols@font}{"2245}
2674 \global\let\cong\simeqq

```

```

2675 \DeclareMathSymbol{\approxeq}{\mathrel}{\M@extsymbols@font}{"224A}
2676 \DeclareMathSymbol{\sssim}{\mathrel}{\M@extsymbols@font}{"224B}
2677 \DeclareMathSymbol{\seq}{\mathrel}{\M@extsymbols@font}{"224C}
2678 \DeclareMathSymbol{\doteq}{\mathrel}{\M@extsymbols@font}{"2250}
2679 \DeclareMathSymbol{\coloneq}{\mathrel}{\M@extsymbols@font}{"2254}
2680 \DeclareMathSymbol{\eqcolon}{\mathrel}{\M@extsymbols@font}{"2255}
2681 \DeclareMathSymbol{\ringeq}{\mathrel}{\M@extsymbols@font}{"2257}
2682 \DeclareMathSymbol{\arceq}{\mathrel}{\M@extsymbols@font}{"2258}
2683 \DeclareMathSymbol{\wedgreek}{\mathrel}{\M@extsymbols@font}{"2259}
2684 \DeclareMathSymbol{\veeeq}{\mathrel}{\M@extsymbols@font}{"225A}
2685 \DeclareMathSymbol{\stareq}{\mathrel}{\M@extsymbols@font}{"225B}
2686 \DeclareMathSymbol{\triangleeq}{\mathrel}{\M@extsymbols@font}{"225C}
2687 \DeclareMathSymbol{\defeq}{\mathrel}{\M@extsymbols@font}{"225D}
2688 \DeclareMathSymbol{\qeq}{\mathrel}{\M@extsymbols@font}{"225F}
2689 \DeclareMathSymbol{\lsim}{\mathrel}{\M@extsymbols@font}{"2272}
2690 \DeclareMathSymbol{\gsim}{\mathrel}{\M@extsymbols@font}{"2273}
2691 \DeclareMathSymbol{\prec}{\mathrel}{\M@extsymbols@font}{"227A}
2692 \DeclareMathSymbol{\succ}{\mathrel}{\M@extsymbols@font}{"227B}
2693 \DeclareMathSymbol{\preceq}{\mathrel}{\M@extsymbols@font}{"227C}
2694 \DeclareMathSymbol{\succeq}{\mathrel}{\M@extsymbols@font}{"227D}
2695 \DeclareMathSymbol{\preceqq}{\mathrel}{\M@extsymbols@font}{"2AB3}
2696 \DeclareMathSymbol{\succeqq}{\mathrel}{\M@extsymbols@font}{"2AB4}
2697 \DeclareMathSymbol{\precsim}{\mathrel}{\M@extsymbols@font}{"227E}
2698 \DeclareMathSymbol{\succsim}{\mathrel}{\M@extsymbols@font}{"227F}
2699 \DeclareMathSymbol{\precapprox}{\mathrel}{\M@extsymbols@font}{"2AB7}
2700 \DeclareMathSymbol{\succapprox}{\mathrel}{\M@extsymbols@font}{"2AB8}
2701 \DeclareMathSymbol{\preceqprec}{\mathrel}{\M@extsymbols@font}{"2ABB}
2702 \DeclareMathSymbol{\succsucc}{\mathrel}{\M@extsymbols@font}{"2ABC}
2703 \DeclareMathSymbol{\asymp}{\mathrel}{\M@extsymbols@font}{"224D}
2704 \DeclareMathSymbol{\nin}{\mathrel}{\M@extsymbols@font}{"2209}
2705 \DeclareMathSymbol{\nni}{\mathrel}{\M@extsymbols@font}{"220C}
2706 \DeclareMathSymbol{\nsubset}{\mathrel}{\M@extsymbols@font}{"2284}
2707 \DeclareMathSymbol{\nsupset}{\mathrel}{\M@extsymbols@font}{"2285}
2708 \DeclareMathSymbol{\nsubseteq}{\mathrel}{\M@extsymbols@font}{"2288}
2709 \DeclareMathSymbol{\nsupseteq}{\mathrel}{\M@extsymbols@font}{"2289}
2710 \DeclareMathSymbol{\subsetneq}{\mathrel}{\M@extsymbols@font}{"228A}
2711 \DeclareMathSymbol{\supsetneq}{\mathrel}{\M@extsymbols@font}{"228B}
2712 \DeclareMathSymbol{\nsubsetseteq}{\mathrel}{\M@extsymbols@font}{"22E2}
2713 \DeclareMathSymbol{\nsubsetseteq}{\mathrel}{\M@extsymbols@font}{"22E3}
2714 \DeclareMathSymbol{\sqsubsetneq}{\mathrel}{\M@extsymbols@font}{"22E4}
2715 \DeclareMathSymbol{\sqsupsetneq}{\mathrel}{\M@extsymbols@font}{"22E5}
2716 \DeclareMathSymbol{\neq}{\mathrel}{\M@extsymbols@font}{"2260}
2717 \DeclareMathSymbol{\nleq}{\mathrel}{\M@extsymbols@font}{"226E}
2718 \DeclareMathSymbol{\nleqq}{\mathrel}{\M@extsymbols@font}{"2270}
2719 \DeclareMathSymbol{\ngeq}{\mathrel}{\M@extsymbols@font}{"2271}
2720 \DeclareMathSymbol{\lneq}{\mathrel}{\M@extsymbols@font}{"2A87}
2721 \DeclareMathSymbol{\gneq}{\mathrel}{\M@extsymbols@font}{"2A88}

```

```

2722 \DeclareMathSymbol{\lneqq}{\mathrel}{\M@extsymbols@font}{"2268}
2723 \DeclareMathSymbol{\gneqq}{\mathrel}{\M@extsymbols@font}{"2269}
2724 \DeclareMathSymbol{\ntriangleleft}{\mathrel}{\M@extsymbols@font}{"22EA}
2725 \DeclareMathSymbol{\ntriangleright}{\mathrel}{\M@extsymbols@font}{"22EB}
2726 \DeclareMathSymbol{\ntrianglelefteq}{\mathrel}{\M@extsymbols@font}{"22EC}
2727 \DeclareMathSymbol{\ntrianglerighteq}{\mathrel}{\M@extsymbols@font}{"22ED}
2728 \DeclareMathSymbol{\nsim}{\mathrel}{\M@extsymbols@font}{"2241}
2729 \DeclareMathSymbol{\napprox}{\mathrel}{\M@extsymbols@font}{"2249}
2730 \DeclareMathSymbol{\nsimeq}{\mathrel}{\M@extsymbols@font}{"2244}
2731 \DeclareMathSymbol{\nsimeqq}{\mathrel}{\M@extsymbols@font}{"2247}
2732 \DeclareMathSymbol{\simneqq}{\mathrel}{\M@extsymbols@font}{"2246}
2733 \DeclareMathSymbol{\nlsim}{\mathrel}{\M@extsymbols@font}{"2274}
2734 \DeclareMathSymbol{\ngsim}{\mathrel}{\M@extsymbols@font}{"2275}
2735 \DeclareMathSymbol{\lnsim}{\mathrel}{\M@extsymbols@font}{"22E6}
2736 \DeclareMathSymbol{\gnsim}{\mathrel}{\M@extsymbols@font}{"22E7}
2737 \DeclareMathSymbol{\lnapprox}{\mathrel}{\M@extsymbols@font}{"2A89}
2738 \DeclareMathSymbol{\gnapprox}{\mathrel}{\M@extsymbols@font}{"2A8A}
2739 \DeclareMathSymbol{\nprec}{\mathrel}{\M@extsymbols@font}{"2280}
2740 \DeclareMathSymbol{\nsucc}{\mathrel}{\M@extsymbols@font}{"2281}
2741 \DeclareMathSymbol{\npreceq}{\mathrel}{\M@extsymbols@font}{"22E0}
2742 \DeclareMathSymbol{\nsucceq}{\mathrel}{\M@extsymbols@font}{"22E1}
2743 \DeclareMathSymbol{\precneq}{\mathrel}{\M@extsymbols@font}{"2AB1}
2744 \DeclareMathSymbol{\succneq}{\mathrel}{\M@extsymbols@font}{"2AB2}
2745 \DeclareMathSymbol{\precneqq}{\mathrel}{\M@extsymbols@font}{"2AB5}
2746 \DeclareMathSymbol{\succneqq}{\mathrel}{\M@extsymbols@font}{"2AB6}
2747 \DeclareMathSymbol{\precnsim}{\mathrel}{\M@extsymbols@font}{"22E8}
2748 \DeclareMathSymbol{\succnsim}{\mathrel}{\M@extsymbols@font}{"22E9}
2749 \DeclareMathSymbol{\precnapprox}{\mathrel}{\M@extsymbols@font}{"2AB9}
2750 \DeclareMathSymbol{\succnapprox}{\mathrel}{\M@extsymbols@font}{"2ABA}
2751 \DeclareMathSymbol{\nequiv}{\mathrel}{\M@extsymbols@font}{"2262}

```

We handle `\ng` specially. The L<sup>A</sup>T<sub>E</sub>X kernel defines `\ng` as a text symbol, so we define `\mathng` like for `\$,` etc.

```

2752 \global\let\textng\ng
2753 \let\ng\@undefined
2754 \DeclareMathSymbol{\mathng}{\mathrel}{\M@extsymbols@font}{"226F}
2755 \protected\gdef\ng{\ifmmode\mathng\else\textng\fi}

```

If we're not adjusting the font, we declare `\nabla` here.

```

2756 \ifM@adjust@font\else
2757   \DeclareMathSymbol{\nabla}{\mathord}{\M@extsymbols@font}{"2207}
2758 \fi}

```

Set arrows.

```

2759 \def\M@arrows@set{%
2760   \edef\M@arrows@font{M\M@arrowsshape\@tempa}
2761   \let\uparrow\@undefined
2762   \let\Uparrow\@undefined
2763   \let\downarrow\@undefined

```

```

2764 \let\Downarrow\@undefined
2765 \let\updownarrow\@undefined
2766 \let\Updownarrow\@undefined
2767 \let\longrightarrow\@undefined
2768 \let\longleftarrow\@undefined
2769 \let\longleftrightarrow\@undefined
2770 \let\hookrightarrow\@undefined
2771 \let\hookleftarrow\@undefined
2772 \let\Longrightarrow\@undefined
2773 \let\Longleftarrow\@undefined
2774 \let\Longleftrightarrow\@undefined
2775 \let\rightleftharpoons\@undefined
2776 \DeclareMathSymbol{\rightarrow}{\mathrel}{\M@arrows@font}{"2192}
2777   \global\let\to\rightarrow
2778 \DeclareMathSymbol{\nrightarrow}{\mathrel}{\M@arrows@font}{"219B}
2779 \DeclareMathSymbol{\Rightarrow}{\mathrel}{\M@arrows@font}{"21D2}
2780 \DeclareMathSymbol{\nRightarrow}{\mathrel}{\M@arrows@font}{"21CF}
2781 \DeclareMathSymbol{\Rrightarrow}{\mathrel}{\M@arrows@font}{"21DB}
2782 \DeclareMathSymbol{\longrightarrow}{\mathrel}{\M@arrows@font}{"27F6}
2783 \DeclareMathSymbol{\Longrightarrow}{\mathrel}{\M@arrows@font}{"27F9}
2784 \DeclareMathSymbol{\rightbararrow}{\mathrel}{\M@arrows@font}{"21A6}
2785   \global\let\mapsto\rightbararrow
2786 \DeclareMathSymbol{\Rightbararrow}{\mathrel}{\M@arrows@font}{"2907}
2787 \DeclareMathSymbol{\longrightbararrow}{\mathrel}{\M@arrows@font}{"27FC}
2788   \global\let\longmapsto\longrightbararrow
2789 \DeclareMathSymbol{\Longrightbararrow}{\mathrel}{\M@arrows@font}{"27FE}
2790 \DeclareMathSymbol{\hookrightarrow}{\mathrel}{\M@arrows@font}{"21AA}
2791 \DeclareMathSymbol{\rightdasharrow}{\mathrel}{\M@arrows@font}{"21E2}
2792 \DeclareMathSymbol{\rightharpoonup}{\mathrel}{\M@arrows@font}{"21C0}
2793 \DeclareMathSymbol{\rightharpoondown}{\mathrel}{\M@arrows@font}{"21C1}
2794 \DeclareMathSymbol{\rightarrowtail}{\mathrel}{\M@arrows@font}{"21A3}
2795 \DeclareMathSymbol{\rightplusarrow}{\mathrel}{\M@arrows@font}{"27F4}
2796 \DeclareMathSymbol{\rightwvearrow}{\mathrel}{\M@arrows@font}{"219D}
2797 \DeclareMathSymbol{\rightsquigarrow}{\mathrel}{\M@arrows@font}{"21DD}
2798 \DeclareMathSymbol{\longrightsquigarrow}{\mathrel}{\M@arrows@font}{"27FF}
2799 \DeclareMathSymbol{\looparrowright}{\mathrel}{\M@arrows@font}{"21AC}
2800 \DeclareMathSymbol{\curvearrowright}{\mathrel}{\M@arrows@font}{"293B}
2801 \DeclareMathSymbol{\circlearrowright}{\mathrel}{\M@arrows@font}{"21BB}
2802 \DeclareMathSymbol{\twoheadrightarrow}{\mathrel}{\M@arrows@font}{"21A0}
2803 \DeclareMathSymbol{\rightarrowto bar}{\mathrel}{\M@arrows@font}{"21E5}
2804 \DeclareMathSymbol{\rightwhitearrow}{\mathrel}{\M@arrows@font}{"21E8}
2805 \DeclareMathSymbol{\rightrightarrowto bar}{\mathrel}{\M@arrows@font}{"21C9}
2806 \DeclareMathSymbol{\rightrightarrowto bar}{\mathrel}{\M@arrows@font}{"21F6}
2807 \DeclareMathSymbol{\leftarrow}{\mathrel}{\M@arrows@font}{"2190}
2808   \global\let\from\leftarrow
2809 \DeclareMathSymbol{\nleftarrow}{\mathrel}{\M@arrows@font}{"219A}
2810 \DeclareMathSymbol{\Leftarrow}{\mathrel}{\M@arrows@font}{"21D0}

```

```

2811 \DeclareMathSymbol{\nLeftarrow}{\mathrel}{\M@arrows@font}{"21CD}
2812 \DeclareMathSymbol{\Lleftarrow}{\mathrel}{\M@arrows@font}{"21DA}
2813 \DeclareMathSymbol{\longleftarrow}{\mathrel}{\M@arrows@font}{"27F5}
2814 \DeclareMathSymbol{\Longleftarrow}{\mathrel}{\M@arrows@font}{"27F8}
2815 \DeclareMathSymbol{\leftbararrow}{\mathrel}{\M@arrows@font}{"21A4}
2816   \global\let\mapsfrom\leftbararrow
2817 \DeclareMathSymbol{\Leftbararrow}{\mathrel}{\M@arrows@font}{"2906}
2818 \DeclareMathSymbol{\longleftbararrow}{\mathrel}{\M@arrows@font}{"27FB}
2819   \global\let\longmapsfrom\longleftbararrow
2820 \DeclareMathSymbol{\Longleftbararrow}{\mathrel}{\M@arrows@font}{"27FD}
2821 \DeclareMathSymbol{\hookrightarrow}{\mathrel}{\M@arrows@font}{"21A9}
2822 \DeclareMathSymbol{\leftdasharrow}{\mathrel}{\M@arrows@font}{"21E0}
2823 \DeclareMathSymbol{\leftharpoonup}{\mathrel}{\M@arrows@font}{"21BC}
2824 \DeclareMathSymbol{\leftharpoondown}{\mathrel}{\M@arrows@font}{"21BD}
2825 \DeclareMathSymbol{\leftarrowtail}{\mathrel}{\M@arrows@font}{"21A2}
2826 \DeclareMathSymbol{\leftoplusarrow}{\mathrel}{\M@arrows@font}{"2B32}
2827 \DeclareMathSymbol{\leftwvearrow}{\mathrel}{\M@arrows@font}{"219C}
2828 \DeclareMathSymbol{\leftsquigarrow}{\mathrel}{\M@arrows@font}{"21DC}
2829 \DeclareMathSymbol{\longleftsquigarrow}{\mathrel}{\M@arrows@font}{"2B33}
2830 \DeclareMathSymbol{\looparrowleft}{\mathrel}{\M@arrows@font}{"21AB}
2831 \DeclareMathSymbol{\curvearrowleft}{\mathrel}{\M@arrows@font}{"293A}
2832 \DeclareMathSymbol{\circlearrowleft}{\mathrel}{\M@arrows@font}{"21BA}
2833 \DeclareMathSymbol{\twoheadleftarrow}{\mathrel}{\M@arrows@font}{"219E}
2834 \DeclareMathSymbol{\leftarrowto bar}{\mathrel}{\M@arrows@font}{"21E4}
2835 \DeclareMathSymbol{\leftwhitearrow}{\mathrel}{\M@arrows@font}{"21E6}
2836 \DeclareMathSymbol{\leftleftarrows}{\mathrel}{\M@arrows@font}{"21C7}
2837 \DeclareMathSymbol{\leftleftleftarrows}{\mathrel}{\M@arrows@font}{"2B31}
2838 \DeclareMathSymbol{\leftrightarrow}{\mathrel}{\M@arrows@font}{"2194}
2839 \DeclareMathSymbol{\Leftrightarrow}{\mathrel}{\M@arrows@font}{"21D4}
2840 \DeclareMathSymbol{\nLeftrightarrow}{\mathrel}{\M@arrows@font}{"21CE}
2841 \DeclareMathSymbol{\longletrightarrow}{\mathrel}{\M@arrows@font}{"27F7}
2842 \DeclareMathSymbol{\Longletrightarrow}{\mathrel}{\M@arrows@font}{"27FA}
2843 \DeclareMathSymbol{\leftrightharpoonup}{\mathrel}{\M@arrows@font}{"21AD}
2844 \DeclareMathSymbol{\leftrightharpoons}{\mathrel}{\M@arrows@font}{"21C6}
2845 \DeclareMathSymbol{\leftrightharpoons}{\mathrel}{\M@arrows@font}{"21CB}
2846 \DeclareMathSymbol{\leftrightharpoonustobar}{\mathrel}{\M@arrows@font}{"21B9}
2847 \DeclareMathSymbol{\rightleftarrows}{\mathrel}{\M@arrows@font}{"21C4}
2848 \DeclareMathSymbol{\rightleftharpoons}{\mathrel}{\M@arrows@font}{"21CC}
2849 \DeclareMathSymbol{\uparrow}{\mathrel}{\M@arrows@font}{"2191}
2850 \DeclareMathSymbol{\Uparrow}{\mathrel}{\M@arrows@font}{"21D1}
2851 \DeclareMathSymbol{\Upuparrow}{\mathrel}{\M@arrows@font}{"290A}
2852 \DeclareMathSymbol{\upbararrow}{\mathrel}{\M@arrows@font}{"21A5}
2853 \DeclareMathSymbol{\updasharrow}{\mathrel}{\M@arrows@font}{"21E1}
2854 \DeclareMathSymbol{\upharpoonleft}{\mathrel}{\M@arrows@font}{"21BF}
2855 \DeclareMathSymbol{\upharpoonright}{\mathrel}{\M@arrows@font}{"21BE}
2856 \DeclareMathSymbol{\twoheaduparrow}{\mathrel}{\M@arrows@font}{"219F}
2857 \DeclareMathSymbol{\uparrowto bar}{\mathrel}{\M@arrows@font}{"2912}

```

```

2858 \DeclareMathSymbol{\upwhitearrow}{\mathrel}{\M@arrows@font}{"21E7}
2859 \DeclareMathSymbol{\upwhitebararrow}{\mathrel}{\M@arrows@font}{"21EA}
2860 \DeclareMathSymbol{\upuparrows}{\mathrel}{\M@arrows@font}{"21C8}
2861 \DeclareMathSymbol{\downarrow}{\mathrel}{\M@arrows@font}{"2193}
2862 \DeclareMathSymbol{\Downarrow}{\mathrel}{\M@arrows@font}{"21D3}
2863 \DeclareMathSymbol{\Ddownarrow}{\mathrel}{\M@arrows@font}{"290B}
2864 \DeclareMathSymbol{\downbararrow}{\mathrel}{\M@arrows@font}{"21A7}
2865 \DeclareMathSymbol{\downdasharrow}{\mathrel}{\M@arrows@font}{"21E3}
2866 \DeclareMathSymbol{\zigzagarrow}{\mathrel}{\M@arrows@font}{"21AF}
2867 \global\let\lightningboltarrow\zigzagarrow
2868 \DeclareMathSymbol{\downharpoonleft}{\mathrel}{\M@arrows@font}{"21C3}
2869 \DeclareMathSymbol{\downharpoonright}{\mathrel}{\M@arrows@font}{"21C2}
2870 \DeclareMathSymbol{\twoheaddownarrow}{\mathrel}{\M@arrows@font}{"21A1}
2871 \DeclareMathSymbol{\downarrowto bar}{\mathrel}{\M@arrows@font}{"2913}
2872 \DeclareMathSymbol{\downwhitearrow}{\mathrel}{\M@arrows@font}{"21E9}
2873 \DeclareMathSymbol{\downdownarrows}{\mathrel}{\M@arrows@font}{"21CA}
2874 \DeclareMathSymbol{\updownarrow}{\mathrel}{\M@arrows@font}{"2195}
2875 \DeclareMathSymbol{\Udownarrow}{\mathrel}{\M@arrows@font}{"21D5}
2876 \DeclareMathSymbol{\updownarrows}{\mathrel}{\M@arrows@font}{"21C5}
2877 \DeclareMathSymbol{\downuparrows}{\mathrel}{\M@arrows@font}{"21F5}
2878 \DeclareMathSymbol{\updownharpoons}{\mathrel}{\M@arrows@font}{"296E}
2879 \DeclareMathSymbol{\downupharpoons}{\mathrel}{\M@arrows@font}{"296F}
2880 \DeclareMathSymbol{\nearrow}{\mathrel}{\M@arrows@font}{"2197}
2881 \DeclareMathSymbol{\Narrow}{\mathrel}{\M@arrows@font}{"21D7}
2882 \DeclareMathSymbol{\narrow}{\mathrel}{\M@arrows@font}{"2196}
2883 \DeclareMathSymbol{\Nwarrow}{\mathrel}{\M@arrows@font}{"21D6}
2884 \DeclareMathSymbol{\searrow}{\mathrel}{\M@arrows@font}{"2198}
2885 \DeclareMathSymbol{\Searrow}{\mathrel}{\M@arrows@font}{"21D8}
2886 \DeclareMathSymbol{\swarrow}{\mathrel}{\M@arrows@font}{"2199}
2887 \DeclareMathSymbol{\Swarrow}{\mathrel}{\M@arrows@font}{"21D9}
2888 \DeclareMathSymbol{\nwsearrow}{\mathrel}{\M@arrows@font}{"2921}
2889 \DeclareMathSymbol{\neswarrow}{\mathrel}{\M@arrows@font}{"2922}
2890 \DeclareMathSymbol{\lrcirclearrow}{\mathrel}{\M@arrows@font}{"27F2}
2891 \DeclareMathSymbol{\rccirclearrow}{\mathrel}{\M@arrows@font}{"27F3}

```

Set blackboard bold letters and numbers. The alphanumeric keywords work a bit differently from the other font-setting commands. We define `\mathbb` here, which takes a single argument and is essentially a wrapper around `\M@bb@mathcodes`. That command changes the `\Umathcodes` of letters to the unicode hex values of corresponding blackboard-bold characters, and throughout, `\M@bb@num` stores the family number of the symbol font for the `bb` character class. In the definition of `\mathbb`, we use `\begingroup` and `\endgroup` to avoid creating unexpected atoms. The other alphanumeric keywords work similarly.

```

2892 \def\M@bb@set{%
2893 \protected\def\mathbb##1{\relax
2894 \ifmmode\else
2895 \M@HModeError\mathbb
2896 $%
2897 \fi

```

```

2898   \begingroup
2899     \M@bb@mathcodes
2900     ##1%
2901   \endgroup}
2902   \edef\M@bb@num{\number\csname symM\M@bbshape\@tempa\endcsname}
2903   \protected\edef\M@bb@mathcodes{%
2904   \Umathcode`A=0+\M@bb@num"1D538\relax
2905   \Umathcode`B=0+\M@bb@num"1D539\relax
2906   \Umathcode`C=0+\M@bb@num"2102\relax
2907   \Umathcode`D=0+\M@bb@num"1D53B\relax
2908   \Umathcode`E=0+\M@bb@num"1D53C\relax
2909   \Umathcode`F=0+\M@bb@num"1D53D\relax
2910   \Umathcode`G=0+\M@bb@num"1D53E\relax
2911   \Umathcode`H=0+\M@bb@num"210D\relax
2912   \Umathcode`I=0+\M@bb@num"1D540\relax
2913   \Umathcode`J=0+\M@bb@num"1D541\relax
2914   \Umathcode`K=0+\M@bb@num"1D542\relax
2915   \Umathcode`L=0+\M@bb@num"1D543\relax
2916   \Umathcode`M=0+\M@bb@num"1D544\relax
2917   \Umathcode`N=0+\M@bb@num"2115\relax
2918   \Umathcode`O=0+\M@bb@num"1D546\relax
2919   \Umathcode`P=0+\M@bb@num"2119\relax
2920   \Umathcode`Q=0+\M@bb@num"211A\relax
2921   \Umathcode`R=0+\M@bb@num"211D\relax
2922   \Umathcode`S=0+\M@bb@num"1D54A\relax
2923   \Umathcode`T=0+\M@bb@num"1D54B\relax
2924   \Umathcode`U=0+\M@bb@num"1D54C\relax
2925   \Umathcode`V=0+\M@bb@num"1D54D\relax
2926   \Umathcode`W=0+\M@bb@num"1D54E\relax
2927   \Umathcode`X=0+\M@bb@num"1D54F\relax
2928   \Umathcode`Y=0+\M@bb@num"1D550\relax
2929   \Umathcode`Z=0+\M@bb@num"2124\relax
2930   \Umathcode`a=0+\M@bb@num"1D552\relax
2931   \Umathcode`b=0+\M@bb@num"1D553\relax
2932   \Umathcode`c=0+\M@bb@num"1D554\relax
2933   \Umathcode`d=0+\M@bb@num"1D555\relax
2934   \Umathcode`e=0+\M@bb@num"1D556\relax
2935   \Umathcode`f=0+\M@bb@num"1D557\relax
2936   \Umathcode`g=0+\M@bb@num"1D558\relax
2937   \Umathcode`h=0+\M@bb@num"1D559\relax
2938   \Umathcode`i=0+\M@bb@num"1D55A\relax
2939   \Umathcode`j=0+\M@bb@num"1D55B\relax
2940   \Umathcode`k=0+\M@bb@num"1D55C\relax
2941   \Umathcode`l=0+\M@bb@num"1D55D\relax
2942   \Umathcode`m=0+\M@bb@num"1D55E\relax
2943   \Umathcode`n=0+\M@bb@num"1D55F\relax
2944   \Umathcode`o=0+\M@bb@num"1D560\relax

```



```

2945 \Umathcode`p=0+\M@bb@num"1D561\relax
2946 \Umathcode`q=0+\M@bb@num"1D562\relax
2947 \Umathcode`r=0+\M@bb@num"1D563\relax
2948 \Umathcode`s=0+\M@bb@num"1D564\relax
2949 \Umathcode`t=0+\M@bb@num"1D565\relax
2950 \Umathcode`u=0+\M@bb@num"1D566\relax
2951 \Umathcode`v=0+\M@bb@num"1D567\relax
2952 \Umathcode`w=0+\M@bb@num"1D568\relax
2953 \Umathcode`x=0+\M@bb@num"1D569\relax
2954 \Umathcode`y=0+\M@bb@num"1D56A\relax
2955 \Umathcode`z=0+\M@bb@num"1D56B\relax
2956 \Umathcode`0=0+\M@bb@num"1D7D8\relax
2957 \Umathcode`1=0+\M@bb@num"1D7D9\relax
2958 \Umathcode`2=0+\M@bb@num"1D7DA\relax
2959 \Umathcode`3=0+\M@bb@num"1D7DB\relax
2960 \Umathcode`4=0+\M@bb@num"1D7DC\relax
2961 \Umathcode`5=0+\M@bb@num"1D7DD\relax
2962 \Umathcode`6=0+\M@bb@num"1D7DE\relax
2963 \Umathcode`7=0+\M@bb@num"1D7DF\relax
2964 \Umathcode`8=0+\M@bb@num"1D7E0\relax
2965 \Umathcode`9=0+\M@bb@num"1D7E1\relax}}

```

Set caligraphic letters.

```

2966 \def\M@cal@set{%
2967   \protected\def\mathcal##1{\relax
2968     \ifmmode\else
2969       \M@HModeError\mathcal
2970       $%
2971       \fi
2972       \begingroup
2973         \M@cal@mathcodes
2974         ##1%
2975       \endgroup}
2976 \edef\M@cal@num{\number\csname symM\M@calshape\@tempa\endcsname}
2977 \protected\edef\M@cal@mathcodes{%
2978 \Umathcode`A=0+\M@cal@num"1D49C\relax
2979 \Umathcode`B=0+\M@cal@num"212C\relax
2980 \Umathcode`C=0+\M@cal@num"1D49E\relax
2981 \Umathcode`D=0+\M@cal@num"1D49F\relax
2982 \Umathcode`E=0+\M@cal@num"2130\relax
2983 \Umathcode`F=0+\M@cal@num"2131\relax
2984 \Umathcode`G=0+\M@cal@num"1D4A2\relax
2985 \Umathcode`H=0+\M@cal@num"210B\relax
2986 \Umathcode`I=0+\M@cal@num"2110\relax
2987 \Umathcode`J=0+\M@cal@num"1D4A5\relax
2988 \Umathcode`K=0+\M@cal@num"1D4A6\relax
2989 \Umathcode`L=0+\M@cal@num"2112\relax
2990 \Umathcode`M=0+\M@cal@num"2133\relax

```

```

2991 \Umathcode`N=0+\M@cal@num"1D4A9\relax
2992 \Umathcode`O=0+\M@cal@num"1D4AA\relax
2993 \Umathcode`P=0+\M@cal@num"1D4AB\relax
2994 \Umathcode`Q=0+\M@cal@num"1D4AC\relax
2995 \Umathcode`R=0+\M@cal@num"211B\relax
2996 \Umathcode`S=0+\M@cal@num"1D4AE\relax
2997 \Umathcode`T=0+\M@cal@num"1D4AF\relax
2998 \Umathcode`U=0+\M@cal@num"1D4B0\relax
2999 \Umathcode`V=0+\M@cal@num"1D4B1\relax
3000 \Umathcode`W=0+\M@cal@num"1D4B2\relax
3001 \Umathcode`X=0+\M@cal@num"1D4B3\relax
3002 \Umathcode`Y=0+\M@cal@num"1D4B4\relax
3003 \Umathcode`Z=0+\M@cal@num"1D4B5\relax
3004 \Umathcode`a=0+\M@cal@num"1D4B6\relax
3005 \Umathcode`b=0+\M@cal@num"1D4B7\relax
3006 \Umathcode`c=0+\M@cal@num"1D4B8\relax
3007 \Umathcode`d=0+\M@cal@num"1D4B9\relax
3008 \Umathcode`e=0+\M@cal@num"212F\relax
3009 \Umathcode`f=0+\M@cal@num"1D4BB\relax
3010 \Umathcode`g=0+\M@cal@num"210A\relax
3011 \Umathcode`h=0+\M@cal@num"1D4BD\relax
3012 \Umathcode`i=0+\M@cal@num"1D4BE\relax
3013 \Umathcode`j=0+\M@cal@num"1D4BF\relax
3014 \Umathcode`k=0+\M@cal@num"1D4C0\relax
3015 \Umathcode`l=0+\M@cal@num"1D4C1\relax
3016 \Umathcode`m=0+\M@cal@num"1D4C2\relax
3017 \Umathcode`n=0+\M@cal@num"1D4C3\relax
3018 \Umathcode`o=0+\M@cal@num"2134\relax
3019 \Umathcode`p=0+\M@cal@num"1D4C5\relax
3020 \Umathcode`q=0+\M@cal@num"1D4C6\relax
3021 \Umathcode`r=0+\M@cal@num"1D4C7\relax
3022 \Umathcode`s=0+\M@cal@num"1D4C8\relax
3023 \Umathcode`t=0+\M@cal@num"1D4C9\relax
3024 \Umathcode`u=0+\M@cal@num"1D4CA\relax
3025 \Umathcode`v=0+\M@cal@num"1D4CB\relax
3026 \Umathcode`w=0+\M@cal@num"1D4CC\relax
3027 \Umathcode`x=0+\M@cal@num"1D4CD\relax
3028 \Umathcode`y=0+\M@cal@num"1D4CE\relax
3029 \Umathcode`z=0+\M@cal@num"1D4CF\relax}}

```

Set fraktur letters.

```

3030 \def\M@frak@set{%
3031 \protected\def\mathfrak##1{\relax
3032 \ifmmode\else
3033 \M@HModeError\mathfrak
3034 $%
3035 \fi
3036 \begingroup

```

```

3037     \M@frac@mathcodes
3038     ##1%
3039     \endgroup}
3040     \edef\M@frac@num{\number\csname symM\M@fracshape\@tempa\endcsname}
3041     \protected\edef\M@frac@mathcodes{%
3042     \Umathcode`A=0+\M@frac@num"1D504\relax
3043     \Umathcode`B=0+\M@frac@num"1D505\relax
3044     \Umathcode`C=0+\M@frac@num"212D\relax
3045     \Umathcode`D=0+\M@frac@num"1D507\relax
3046     \Umathcode`E=0+\M@frac@num"1D508\relax
3047     \Umathcode`F=0+\M@frac@num"1D509\relax
3048     \Umathcode`G=0+\M@frac@num"1D50A\relax
3049     \Umathcode`H=0+\M@frac@num"210C\relax
3050     \Umathcode`I=0+\M@frac@num"2111\relax
3051     \Umathcode`J=0+\M@frac@num"1D50D\relax
3052     \Umathcode`K=0+\M@frac@num"1D50E\relax
3053     \Umathcode`L=0+\M@frac@num"1D50F\relax
3054     \Umathcode`M=0+\M@frac@num"1D510\relax
3055     \Umathcode`N=0+\M@frac@num"1D511\relax
3056     \Umathcode`O=0+\M@frac@num"1D512\relax
3057     \Umathcode`P=0+\M@frac@num"1D513\relax
3058     \Umathcode`Q=0+\M@frac@num"1D514\relax
3059     \Umathcode`R=0+\M@frac@num"211C\relax
3060     \Umathcode`S=0+\M@frac@num"1D516\relax
3061     \Umathcode`T=0+\M@frac@num"1D517\relax
3062     \Umathcode`U=0+\M@frac@num"1D518\relax
3063     \Umathcode`V=0+\M@frac@num"1D519\relax
3064     \Umathcode`W=0+\M@frac@num"1D51A\relax
3065     \Umathcode`X=0+\M@frac@num"1D51B\relax
3066     \Umathcode`Y=0+\M@frac@num"1D51C\relax
3067     \Umathcode`Z=0+\M@frac@num"2128\relax
3068     \Umathcode`a=0+\M@frac@num"1D51E\relax
3069     \Umathcode`b=0+\M@frac@num"1D51F\relax
3070     \Umathcode`c=0+\M@frac@num"1D520\relax
3071     \Umathcode`d=0+\M@frac@num"1D521\relax
3072     \Umathcode`e=0+\M@frac@num"1D522\relax
3073     \Umathcode`f=0+\M@frac@num"1D523\relax
3074     \Umathcode`g=0+\M@frac@num"1D524\relax
3075     \Umathcode`h=0+\M@frac@num"1D525\relax
3076     \Umathcode`i=0+\M@frac@num"1D526\relax
3077     \Umathcode`j=0+\M@frac@num"1D527\relax
3078     \Umathcode`k=0+\M@frac@num"1D528\relax
3079     \Umathcode`l=0+\M@frac@num"1D529\relax
3080     \Umathcode`m=0+\M@frac@num"1D52A\relax
3081     \Umathcode`n=0+\M@frac@num"1D52B\relax
3082     \Umathcode`o=0+\M@frac@num"1D52C\relax
3083     \Umathcode`p=0+\M@frac@num"1D52D\relax

```

```

3084 \Umathcode`q=0+\M@frac@num"1D52E\relax
3085 \Umathcode`r=0+\M@frac@num"1D52F\relax
3086 \Umathcode`s=0+\M@frac@num"1D530\relax
3087 \Umathcode`t=0+\M@frac@num"1D531\relax
3088 \Umathcode`u=0+\M@frac@num"1D532\relax
3089 \Umathcode`v=0+\M@frac@num"1D533\relax
3090 \Umathcode`w=0+\M@frac@num"1D534\relax
3091 \Umathcode`x=0+\M@frac@num"1D535\relax
3092 \Umathcode`y=0+\M@frac@num"1D536\relax
3093 \Umathcode`z=0+\M@frac@num"1D537\relax}}

```

Set bold caligraphic letters.

```

3094 \def\M@bcal@set{%
3095   \protected\def\mathbcal##1{\relax
3096     \ifmmode\else
3097       \M@HModeError\mathbcal
3098       $%
3099       \fi
3100       \begingroup
3101         \M@bcal@mathcodes
3102         ##1%
3103       \endgroup}
3104 \edef\M@bcal@num{\number\csname symM\M@bcalshape\@tempa\endcsname}
3105 \protected\edef\M@bcal@mathcodes{%
3106 \Umathcode`A=0+\M@bcal@num"1D4D0\relax
3107 \Umathcode`B=0+\M@bcal@num"1D4D1\relax
3108 \Umathcode`C=0+\M@bcal@num"1D4D2\relax
3109 \Umathcode`D=0+\M@bcal@num"1D4D3\relax
3110 \Umathcode`E=0+\M@bcal@num"1D4D4\relax
3111 \Umathcode`F=0+\M@bcal@num"1D4D5\relax
3112 \Umathcode`G=0+\M@bcal@num"1D4D6\relax
3113 \Umathcode`H=0+\M@bcal@num"1D4D7\relax
3114 \Umathcode`I=0+\M@bcal@num"1D4D8\relax
3115 \Umathcode`J=0+\M@bcal@num"1D4D9\relax
3116 \Umathcode`K=0+\M@bcal@num"1D4DA\relax
3117 \Umathcode`L=0+\M@bcal@num"1D4DB\relax
3118 \Umathcode`M=0+\M@bcal@num"1D4DC\relax
3119 \Umathcode`N=0+\M@bcal@num"1D4DD\relax
3120 \Umathcode`O=0+\M@bcal@num"1D4DE\relax
3121 \Umathcode`P=0+\M@bcal@num"1D4DF\relax
3122 \Umathcode`Q=0+\M@bcal@num"1D4E0\relax
3123 \Umathcode`R=0+\M@bcal@num"1D4E1\relax
3124 \Umathcode`S=0+\M@bcal@num"1D4E2\relax
3125 \Umathcode`T=0+\M@bcal@num"1D4E3\relax
3126 \Umathcode`U=0+\M@bcal@num"1D4E4\relax
3127 \Umathcode`V=0+\M@bcal@num"1D4E5\relax
3128 \Umathcode`W=0+\M@bcal@num"1D4E6\relax
3129 \Umathcode`X=0+\M@bcal@num"1D4E7\relax

```

```

3130 \Umathcode`Y=0+\M@bcal@num"1D4E8\relax
3131 \Umathcode`Z=0+\M@bcal@num"1D4E9\relax
3132 \Umathcode`a=0+\M@bcal@num"1D4EA\relax
3133 \Umathcode`b=0+\M@bcal@num"1D4EB\relax
3134 \Umathcode`c=0+\M@bcal@num"1D4EC\relax
3135 \Umathcode`d=0+\M@bcal@num"1D4ED\relax
3136 \Umathcode`e=0+\M@bcal@num"1D4EE\relax
3137 \Umathcode`f=0+\M@bcal@num"1D4EF\relax
3138 \Umathcode`g=0+\M@bcal@num"1D4F0\relax
3139 \Umathcode`h=0+\M@bcal@num"1D4F1\relax
3140 \Umathcode`i=0+\M@bcal@num"1D4F2\relax
3141 \Umathcode`j=0+\M@bcal@num"1D4F3\relax
3142 \Umathcode`k=0+\M@bcal@num"1D4F4\relax
3143 \Umathcode`l=0+\M@bcal@num"1D4F5\relax
3144 \Umathcode`m=0+\M@bcal@num"1D4F6\relax
3145 \Umathcode`n=0+\M@bcal@num"1D4F7\relax
3146 \Umathcode`o=0+\M@bcal@num"1D4F8\relax
3147 \Umathcode`p=0+\M@bcal@num"1D4F9\relax
3148 \Umathcode`q=0+\M@bcal@num"1D4FA\relax
3149 \Umathcode`r=0+\M@bcal@num"1D4FB\relax
3150 \Umathcode`s=0+\M@bcal@num"1D4FC\relax
3151 \Umathcode`t=0+\M@bcal@num"1D4FD\relax
3152 \Umathcode`u=0+\M@bcal@num"1D4FE\relax
3153 \Umathcode`v=0+\M@bcal@num"1D4FF\relax
3154 \Umathcode`w=0+\M@bcal@num"1D500\relax
3155 \Umathcode`x=0+\M@bcal@num"1D501\relax
3156 \Umathcode`y=0+\M@bcal@num"1D502\relax
3157 \Umathcode`z=0+\M@bcal@num"1D503\relax}]

```

Set bold fraktur letters.

```

3158 \def\M@bfrac@set{%
3159   \protected\def\mathbfrac##1{\relax
3160     \ifmmode\else
3161       \M@HModeError\mathbfrac
3162       $%
3163       \fi
3164       \begingroup
3165         \M@bfrac@mathcodes
3166         ##1%
3167       \endgroup}
3168 \edef\M@bfrac@num{\number\csname symM\M@bfracshape\@tempa\endcsname}
3169 \protected\edef\M@bfrac@mathcodes{%
3170 \Umathcode`A=0+\M@bfrac@num"1D56C\relax
3171 \Umathcode`B=0+\M@bfrac@num"1D56D\relax
3172 \Umathcode`C=0+\M@bfrac@num"1D56E\relax
3173 \Umathcode`D=0+\M@bfrac@num"1D56F\relax
3174 \Umathcode`E=0+\M@bfrac@num"1D570\relax
3175 \Umathcode`F=0+\M@bfrac@num"1D571\relax

```

```

3176 \Umathcode`G=0+\M@bfrac@num"1D572\relax
3177 \Umathcode`H=0+\M@bfrac@num"1D573\relax
3178 \Umathcode`I=0+\M@bfrac@num"1D574\relax
3179 \Umathcode`J=0+\M@bfrac@num"1D575\relax
3180 \Umathcode`K=0+\M@bfrac@num"1D576\relax
3181 \Umathcode`L=0+\M@bfrac@num"1D577\relax
3182 \Umathcode`M=0+\M@bfrac@num"1D578\relax
3183 \Umathcode`N=0+\M@bfrac@num"1D579\relax
3184 \Umathcode`O=0+\M@bfrac@num"1D57A\relax
3185 \Umathcode`P=0+\M@bfrac@num"1D57B\relax
3186 \Umathcode`Q=0+\M@bfrac@num"1D57C\relax
3187 \Umathcode`R=0+\M@bfrac@num"1D57D\relax
3188 \Umathcode`S=0+\M@bfrac@num"1D57E\relax
3189 \Umathcode`T=0+\M@bfrac@num"1D57F\relax
3190 \Umathcode`U=0+\M@bfrac@num"1D580\relax
3191 \Umathcode`V=0+\M@bfrac@num"1D581\relax
3192 \Umathcode`W=0+\M@bfrac@num"1D582\relax
3193 \Umathcode`X=0+\M@bfrac@num"1D583\relax
3194 \Umathcode`Y=0+\M@bfrac@num"1D584\relax
3195 \Umathcode`Z=0+\M@bfrac@num"1D585\relax
3196 \Umathcode`a=0+\M@bfrac@num"1D586\relax
3197 \Umathcode`b=0+\M@bfrac@num"1D587\relax
3198 \Umathcode`c=0+\M@bfrac@num"1D588\relax
3199 \Umathcode`d=0+\M@bfrac@num"1D589\relax
3200 \Umathcode`e=0+\M@bfrac@num"1D58A\relax
3201 \Umathcode`f=0+\M@bfrac@num"1D58B\relax
3202 \Umathcode`g=0+\M@bfrac@num"1D58C\relax
3203 \Umathcode`h=0+\M@bfrac@num"1D58D\relax
3204 \Umathcode`i=0+\M@bfrac@num"1D58E\relax
3205 \Umathcode`j=0+\M@bfrac@num"1D58F\relax
3206 \Umathcode`k=0+\M@bfrac@num"1D590\relax
3207 \Umathcode`l=0+\M@bfrac@num"1D591\relax
3208 \Umathcode`m=0+\M@bfrac@num"1D592\relax
3209 \Umathcode`n=0+\M@bfrac@num"1D593\relax
3210 \Umathcode`o=0+\M@bfrac@num"1D594\relax
3211 \Umathcode`p=0+\M@bfrac@num"1D595\relax
3212 \Umathcode`q=0+\M@bfrac@num"1D596\relax
3213 \Umathcode`r=0+\M@bfrac@num"1D597\relax
3214 \Umathcode`s=0+\M@bfrac@num"1D598\relax
3215 \Umathcode`t=0+\M@bfrac@num"1D599\relax
3216 \Umathcode`u=0+\M@bfrac@num"1D59A\relax
3217 \Umathcode`v=0+\M@bfrac@num"1D59B\relax
3218 \Umathcode`w=0+\M@bfrac@num"1D59C\relax
3219 \Umathcode`x=0+\M@bfrac@num"1D59D\relax
3220 \Umathcode`y=0+\M@bfrac@num"1D59E\relax
3221 \Umathcode`z=0+\M@bfrac@num"1D59F\relax}}

```

And that's everything!

## Version History

New features and updates with each version. Listed in no particular order.

- |   |   |
|---|---|
| <p><b>1.1b</b> ..... July 2018<br/>—initial release</p> <p><b>1.2</b> ..... August 2018<br/>—minor bug fix for <code>\mathfrak</code><br/>—eliminated redundant batchfile</p> <p><b>1.3</b> ..... January 2019<br/>—added <code>symbols</code> keyword<br/>—created <code>mathfont_example.pdf</code><br/>—corrected the description of the <code>mathastext</code> package<br/>—font-change <code>\message</code> added to <code>\mathfont</code></p> <p><b>1.4</b> ..... April 2019<br/>—<code>\setfont</code> command added<br/>—<code>\mathfont</code> optional argument can parse spaces<br/>—<code>no-operators</code> now default package optional argument<br/>—added <code>\comma</code> command<br/>—new fancy fatal error message<br/>—improved messaging for <code>\mathfont</code><br/>—internal command <code>\mathpound</code> changed to <code>\mathhash</code><br/>—added a missing <code>#1</code> after <code>\char`\"</code> in the example code redefining <code>"</code> in the user guide</p> <p><b>1.5</b> ..... April 2019<br/>—separated <code>\increment</code> and <code>\Delta</code><br/>—version history added<br/>—initial off-the-shelf use insert added</p> <p><b>1.6</b> ..... December 2019<br/>—separated implementation and user documentation<br/>—created <code>mathfont_heading.tex</code><br/>—created <code>mathfont_doc_patch.tex</code> for use with the index<br/>—changed <code>mathfont_greek.pdf</code> to <code>mathfont_symbol_list.pdf</code></p> | <p>—eliminated <code>mathfont_example.pdf</code><br/>—eliminated <code>operators</code> package option<br/>—eliminated <code>packages</code> package option<br/>—font name can be package option<br/>—added Hebrew and Cyrillic characters<br/>—separated ancient Greek from modern Greek characters<br/>—created new keywords: <code>extsymbols</code>, <code>delimiters</code>, <code>arrows</code>, <code>diacritics</code>, <code>bigops</code>, <code>extbigops</code><br/>—improved messaging<br/>—improved internal code for local font-change commands<br/>—improved space parsing for the optional argument of <code>\mathfont</code><br/>—bug fix for <code>\#</code>, etc. commands<br/>—bad input for <code>\mathbb</code>, etc. now gives a warning<br/>—improved error checking for <code>\newmathrm</code>, etc. commands<br/>—<code>\mathfont</code> now ignores bad options (on top of issuing an error)<br/>—internal commands now begin with <code>\M@...</code><br/>—added Easter Egg!<br/>—improved indexing<br/>—<code>mathfont.dtx</code> renamed as <code>mathfont_code.dtx</code><br/>—<code>\newmathbold</code> renamed as <code>\newmathbf</code><br/>—default local font changes now use <code>\updefault</code>, etc.<br/>—added fatal error for missing <code>fontspec</code><br/>—fatal errors result in <code>\endinput</code> rather than <code>\@@end</code></p> <p><b>2.0</b> ..... December 2021</p> <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <p><b>Big Change:</b> Font adjustments for LuaTeX: new glyph boundaries for Latin letters in math mode, resizable delimiters, big operators, MathConstants table based on font metrics.</p> </div> <p>—added <code>\CharmLine</code> and <code>\CharmFile</code></p> |
|---|---|

- added `\mathconstantsfont`
  - certain dimensions in equations are now adjustable when typesetting with Lua $\TeX$
  - added `adjust` and `no-adjust` package options
  - automatic generation of `ind` file
  - fixed symbols for `\leftharpoonup`, `\leftharpoondown`, and fraktur R
  - cleaned up internal code and documentation
  - font names for `\mathfont` stored to avoid multiple symbol font declarations with the same font
  - more information about `nfss` family names stored and provided
  - added option `empty`
  - raised upper bound on `\DeclareSymbolFont` to 256
  - reintroduced `mathfont_example.tex` with different contents
  - changed several symbol-commands to `\protected` rather than robust macros
  - many user-level commands are now `\protected`
  - `\updefault` changed to `\shapedefault`
  - eliminated `\catcode` change for space characters when scanning optional argument of `\mathfont`
  - improved messaging for `\mathfont`
  - removed dependence on `fontspec` and added internal font-loader
  - switched `\epsilon` and `\varepsilon`
  - switched `\phi` and `\varphi`
  - changed `/` to produce a solidus in math mode and added `\fractionslash`
  - removed `\restoremathinternals` from the user guide
  - `\setfont` now sets `\mathrm`, etc.
  - added `\newmathsc`, other math alphabet commands for small caps
- 2.1** ..... November 2022
- `\mathbb`, etc. commands change `\Umathcodes` of letters instead of `\M@bb,etc.@letter` commands
  - removed warnings about non-letter contents of `\mathbb`, etc.
  - fonts loaded twice, once with default settings (for text) and once in base mode (for math)
  - `mathconstantsfont` accepts “upright” or “italic” as optional argument
- 2.2** ..... December 2022
- changed the easter egg text
  - updated patch for `\DeclareSymbolFont` to work with changes to the kernel (eliminated `\M@p@tch@decl@re` error message)
  - calling Plain  $\TeX$  on `mathfont_code.dtx` produces `sty` file and no `pdf` file
- 2.2a** ..... December 2022
- bug fix for `\mathconstantsfont`
  - bug fix for `\M@check@int`
  - added `doc2` option to `ltxdoc` in `mathfont_code.dtx`
- 2.2b** ..... August 2023
- minor changes to code and documentation
  - `\ng` now works in math (as not greater than symbol) and text (as pronunciation symbol)
- 2.3** ..... September 2023
- solidus and `\fractionslash` are `\mathord` instead of `\mathbin`
  - removed `\mathfont{fontspec}` functionality
  - redesigned font-loader
  - added package options `default-loader` and `fontspec-loader`



# Index

Upright entries refer to lines in the code, and italic entries indicate pages in the document. Bold means a definition.

Symbols	
<code>\#</code> .....	2551
<code>\%</code> .....	2552
<code>\&amp;</code> .....	2553
<code>\=</code> .....	1039
<code>\@Relbar</code> .....	2555, 2562
<code>\@backslashchar</code> .....	
... 873, 887, 889, 890, 1074, 1077, 1079	
<code>\@expandtwoargs</code> ....	869, 873, 1066, 1074
<code>\@mathconstantsfont</code> .....	761, <b>762</b>
<code>\@mathfont</code> .....	689, <b>690</b> , 752
<code>\@optionpresentfalse</code> .....	645, 658
<code>\@optionpresenttrue</code> .....	640
<code>\@percentchar</code> .	881, 1174, 1246, 1326, 1406
<code>\@relbar</code> .....	2554, 2561
<code>\@sqrts@gn</code> .....	2436, 2440, 2454
<code>\@tempbase</code> .....	19
<code>\@tempfeatures</code> .....	19
<code>\@tempwarning</code> .....	<b>548</b> , <b>554</b> , 557
<code>\@verticalbar</code> .....	2556, 2563
<code>\~</code> .....	1038
<code>\_</code> .....	60, 101
<b>A</b>	
<code>\aacute</code> .....	2085
<code>\acute</code> .....	2084
<code>\aftergroup</code> .....	803
<code>\aleph</code> .....	2262
<code>\Alpha</code> .....	1844, 2097
<code>\amalg</code> .....	2625
<code>\approx</code> .....	2540
<code>\approxeq</code> .....	2675
<code>\arceq</code> .....	2682
<code>\asympt</code> .....	2703
<code>\AtEndDocument</code> .....	174
<code>\ayin</code> .....	2277
<b>B</b>	
<code>\backslash</code> .....	2383
Bad argument for .....	12
<code>\bar</code> .....	2092
<code>\bclubsuit</code> .....	2607, 2618
<code>\bdiamondsuit</code> .....	2608
<code>\because</code> .....	2660
<code>\Beta</code> .....	1845, 2098
<code>\beta</code> .....	1815, 2138
<code>\beth</code> .....	2263
<code>\bheartsuit</code> .....	2609
<code>\bigand</code> .....	2569
<code>\bigat</code> .....	2565
<code>\bigdiv</code> .....	2575
<code>\bigdollar</code> .....	2567
<code>\bighash</code> .....	2566
<code>\bigp</code> .....	2571
<code>\bigpercent</code> .....	2568
<code>\bigplus</code> .....	2570
<code>\bigq</code> .....	2572
<code>\bigS</code> .....	2573
<code>\bigsqcap</code> .....	1906, 2496
<code>\bigtimes</code> .....	2574
<code>\bot</code> .....	2599
<code>\breve</code> .....	2089
<code>\bspadesuit</code> .....	2610, 2615
<code>\bullet</code> .....	2529
<b>C</b>	
cannot find the file <code>luaotfload</code> .....	4
catcode changes .....	3
<code>\cdot</code> .....	2532
<code>\CharmLine</code> 21, 377, 381, 912, 922, <b>1062</b> , 1092	
<code>\check</code> .....	2091
<code>\Chi</code> .....	1865, 2118
<code>\chi</code> .....	1835, 2158
<code>\circlearrowleft</code> .....	2832
<code>\circlearrowright</code> .....	2801
<code>\coloneq</code> .....	2679
<code>\comma</code> .....	2520
<code>\cramped</code> .....	1512
<code>\curvearrowleft</code> .....	2831
<code>\curvearrowright</code> .....	2800
<b>D</b>	
<code>\dagger</code> .....	2530
<code>\daleth</code> .....	2265

- `\dashv` ..... 2601  
`\ddagger` ..... 2531  
`\ddot` ..... 2087  
`\Ddownarrow` ..... 2863  
`\DeclareFontFamily` ..... 598, 605  
`\DeclareFontShape` ..... 534  
`\DeclareMathAlphabet` ..... 822  
`\DeclareSymbolFont@m@dropped` .....  
       ..... 433, 437, **438**  
`\defeq` ..... 2687  
`\degree` ..... 2518  
`\Delta` ..... 1847, 2100  
`\delta` ..... 1817, 2140  
`deprecated` ..... 5, 10  
`\Digamma` ..... 2172  
`\digamma` ..... 2184  
`\directlua` ..... 52, 87,  
       235, 442, 570, 600, 616, 877, 1082, 1100  
`\div` ..... 1895, 2527  
`\downarrowto` ..... 2871  
`\downbararrow` ..... 2864  
`\downdasharrow` ..... 2865  
`\downdownarrows` ..... 2873  
`\downharpoonleft` ..... 2868  
`\downharpoonright` ..... 2869  
`\downuparrows` ..... 2877  
`\downupharpoons` ..... 2879  
`\downwhitearrow` ..... 2872
- E**
- `\edef@nospace` .... 588, 593, **686**, 704, 763  
`\emptyset` ..... 2593  
`\endinput` ..... 85, 132  
`\Epsilon` ..... 1848, 2101  
`\epsilon` ..... 1818, 2141  
`\eqcolon` ..... 2680  
`\eqsim` ..... 2672  
`\equiv` ..... 2541  
`\Eta` ..... 1850, 2103  
`\eta` ..... 1820, 2143  
`\exists` ..... 2592
- F**
- `\fakelangle` ..... 1881, 2405  
`\fakellangle` ..... 1883, 2411  
`\fakerrangle` ..... 1882, 2408  
`\fakerrangle` ..... 1884, 2414  
`\flat` ..... 2605
- `\flat` ..... 2602  
`\forall` ..... 2591  
`Forbidden charm info` ..... 12  
`\fractionslash` ..... 2526
- G**
- `\Gamma` ..... 1846, 2099  
`\gamma` ..... 1816, 2139  
`\gapprox` ..... 2670  
`\geq` ..... 2538  
`\geqq` ..... 2668  
`\getanddefine@fonts` ..... 788  
`\ggg` ..... 2666  
`\gimel` ..... 2264  
`\gnapprox` ..... 2738  
`\gneq` ..... 2721  
`\gneqq` ..... 2723  
`\gnsim` ..... 2736  
`\grave` ..... 2088  
`\gsim` ..... 2690
- H**
- `\hat` ..... 2090  
`\hb@xt@` ..... 936, 943, 948  
`\hbar` ..... 2048, 2080  
`\hbox` ..... 940, 941, 2439–2441, 2451  
`\het` ..... 2269  
`\Heta` ..... 2170  
`\heta` ..... 2182  
`\hfil` ..... 944, 946, 949, 951  
`\hourglass` ..... 2658
- I**
- `I already set the font` ..... 10, 24  
`if-parser` ..... 24  
`\ifdim` ..... 2446  
`\ifE@sterEggDecl@red` ..... 7, 150  
`\ifin@` ..... 870, 874, 1067, 1075  
`\ifM@adjust@font` ..... 5, 236,  
       246, 528, 800, 863, 1035, 1957, 1958,  
       2017, 2122, 2302, 2364, 2432, 2564, 2756  
`\ifM@arg@good` ..... 494, 819, 902  
`\ifM@Decl@ref@family` ..... 495  
`\ifM@font@loaded` ..... 6, 996, 1008  
`\ifM@fromCharmFile` ..... 496, 1068, 1076  
`\ifM@Noluaotfload` ..... 4, 99  
`\ifM@radical` ..... 483  
`\ifM@XeTeXLuaTeX` ..... 3, 58

- `\iiintop` ..... 2485, 2486  
`\iintop` ..... 2483, 2484  
`\Im` ..... 2589  
`\imath` ..... 1786, 2046, 2078, 2357  
`\in@` ..... 869, 873, 1066, 1074  
`\increment` ..... 2124, 2129, 2519  
`\infty` ..... 2516  
`\IntegralItalicFactor` .. 18, 898, 911, 919  
Internal commands restored ..... 10, 31  
`\intop` ..... 1909, 2466  
invalid command ..... 2  
Invalid font specifier ..... 11  
Invalid Option for `\mathfont` ..... 10  
Invalid Suboption for `\mathfont` ..... 10  
`\Iota` ..... 1852, 2105  
`\iota` ..... 1822, 2145
- J**
- `\jmath` ..... 1787, 2047, 2079, 2358
- K**
- `\kaf` ..... 2272  
`\Kappa` ..... 1853, 2106  
`\kappa` ..... 1823, 2146  
`\kern` ..... 2447, 2449, 2452  
keyword options for `\mathfont` 16, 21, 23–25  
`\keyword@info@begindocument` .. **956**, 993  
keyword `agreeklower` ..... 71  
keyword `agreekupper` ..... 71  
keyword `arrows` ..... 84  
keyword `bb` ..... 87  
keyword `bcal` ..... 92  
keyword `bfrac` ..... 93  
keyword `bigops` ..... 78  
keyword `cal` ..... 89  
keyword `cyrilliclower` ..... 72  
keyword `cyrillicupper` ..... 71  
keyword `delimiters` ..... 75  
keyword `diacritics` ..... 69  
keyword `digits` ..... 74  
keyword `extbigops` ..... 78  
keyword `extsymbols` ..... 80  
keyword `frac` ..... 90  
keyword `greeklower` ..... 70  
keyword `greekupper` ..... 69  
keyword `hebrew` ..... 73  
keyword `lower` ..... 68  
keyword `operator` ..... 74  
keyword `radical` ..... 77  
keyword `symbols` ..... 78  
keyword `upper` ..... 66  
`\Koppa` ..... 2173  
`\koppa` ..... 2185
- L**
- `\Lambda` ..... 1854, 2107  
`\lambda` ..... 2147  
`\lamed` ..... 2273  
`\lapprox` ..... 2669  
L<sup>A</sup>T<sub>E</sub>X kernel ..... 13, 27, 77  
`\lbrace` ..... 2387  
`\lcirclearrow` ..... 2890  
`\Leftarrow` ..... 2810  
`\leftarrowtail` ..... 2825  
`\leftarrowto` ..... 2834  
`\Leftbararrow` ..... 2817  
`\leftbararrow` ..... 2815, 2816  
`\leftbrace` ..... 2374, 2429  
`\leftdasharrow` ..... 2822  
`\leftharpoondown` ..... 2824  
`\leftharpoonup` ..... 2823  
`\leftleftarrows` ..... 2836  
`\leftleftleftarrows` ..... 2837  
`\leftplusarrow` ..... 2826  
`\Leftrightarrow` ..... 2839  
`\leftrightarrows` ..... 2844  
`\leftrightharpoons` ..... 2846  
`\leftrightharpoons` ..... 2845  
`\leftrightharpoonup` ..... 2843  
`\leftsquigarrow` ..... 2828  
`\leftwavearrow` ..... 2827  
`\leftwhitearrow` ..... 2835  
`\leq` ..... 2537  
`\leqq` ..... 2667  
`\lguil` ..... 1418, 1877, 2393, 2425  
`\llap` ..... 2451  
`\Lleftarrow` ..... 2812  
`\llguil` ..... 1420, 1879, 2399, 2427  
`\lll` ..... 2665  
`\lnapprox` ..... 2737  
`\lneq` ..... 2720  
`\lneqq` ..... 2722  
`\lnsim` ..... 2735  
local font changes ..... 28  
log file ..... 20, 25, 32  
`\Longleftbararrow` ..... 2820

- `\longleftbararrow` ..... 2818, 2819
  - `\longleftsquigarrow` ..... 2829
  - `\longrightbararrow` ..... 2789
  - `\longrightbararrow` ..... 2787, 2788
  - `\longrightsquigarrow` ..... 2798
  - `\looparrowleft` ..... 2830
  - `\looparrowright` ..... 2799
  - `\lsim` ..... 2689
- M**
- `\M@agreeklower@set` ..... 25, 1017, **2180**
  - `\M@agreeklowershape` ..... **503**, 2181
  - `\M@agreekupper@set` ..... 25, 1016, **2168**
  - `\M@agreekuppershape` ..... **502**, 2169
  - `\M@arrows@set` ..... 25, 1029, **2759**
  - `\M@arrowsshape` ..... **515**, 2760
  - `\M@BadIntegerError` ..... **401**, 906
  - `\M@BadMathConstantsFontError` .. **332**, 766
  - `\M@BadMathConstantsFontTypeError` ..  
..... **340**, 777
  - `\M@bb@mathcodes` ..... 2899, **2903**
  - `\M@bb@num` ..... **2902**, 2904–2965
  - `\M@bb@set` ..... 25, 1030, **2892**
  - `\M@bbshape` ..... **516**, 2902
  - `\M@bcal@mathcodes` ..... 3101, **3105**
  - `\M@bcal@num` ..... **3104**, 3106–3157
  - `\M@bcal@set` ..... 25, 1033, **3094**
  - `\M@bcalshape` ..... **519**, 3104
  - `\M@bfrac@mathcodes` ..... 3165, **3169**
  - `\M@bfrac@num` ..... **3168**, 3170–3221
  - `\M@bfrac@set` ..... 25, 1034, **3158**
  - `\M@bfracshape` ..... **520**, 3168
  - `\M@bigops@set` ..... 25, 1025, **2460**
  - `\M@bigopsshape` ..... **511**, 2461
  - `\M@cal@mathcodes` ..... 2973, **2977**
  - `\M@cal@num` ..... **2976**, 2978–3029
  - `\M@cal@set` ..... 25, 1031, **2966**
  - `\M@calshape` ..... **517**, 2976
  - `\M@Charm` ..... 460, 1089, 1091, 1093, 1097
  - `\M@CharsSetWarning` ..... **270**, 709
  - `\M@check@csarg` ..... **805**, 818, **827**
  - `\M@check@int` ..... **864**, 901
  - `\M@check@nfss@shapes` ..... **546**, 569, 576
  - `\M@check@option@valid` ..... **635**, 665
  - `\M@check@suboption@valid` ..... **648**, 675
  - `\M@cyrilliclower@set` .... 25, 1019, **2226**
  - `\M@cyrilliclowershape` ..... **505**, 2227
  - `\M@cyrillicupper@set` .... 25, 1018, **2192**
  - `\M@cyrillicuppershape` ..... **504**, 2193
  - `\M@declare@shape` ..... **531**, 545, 632
  - `\M@DecSymDef` ..... **441**, 443, 448
  - `\M@default@newmath@cmds` .. **829**, 838, **848**
  - `\M@defaultkeys` ..... **526**, 529, **529**, 689
  - `\M@define@newmath@cmd` .... **825**, 838, 847
  - `\M@delimiters@set` . 25, 1023, **2365**, **2419**
  - `\M@delimitersshape` ..... **509**, 2366, 2420
  - `\M@DeprecatedWarning` ..... **287**, 860, 862
  - `\M@diacritics@set` ..... 25, 1013, **2082**
  - `\M@diacriticsshape` ..... **499**, 2083
  - `\M@digits@font` ..... **2290**, 2291–2300
  - `\M@digits@set` ..... 25, 1021, **2289**
  - `\M@digitsshape` ..... **507**, 2290
  - `\M@DoubleArgError` ..... **359**, 814
  - `\M@entries@assert` ..... 1188, 1204, 1221
  - `\M@extbigops@set` ..... 25, 1026, **2467**
  - `\M@extbigopsshape` ..... **512**, 2468
  - `\M@extsymbols@set` ..... 25, 1028, **2578**
  - `\M@extsymbolsshape` ..... **514**, 2579
  - `\M@fill@nfss@shapes` .. **536**, 599, 606, 633
  - `\M@FontChangeInfo` ..... **264**, 734
  - `\M@ForbiddenCharmFile` ... **384**, 1069, 1077
  - `\M@ForbiddenCharmLine` ... **375**, 1071, 1079
  - `\M@frak@mathcodes` ..... 3037, **3041**
  - `\M@frak@num` ..... **3040**, 3042–3093
  - `\M@frak@set` ..... 25, 1032, **3030**
  - `\M@frakshape` ..... **518**, 3040
  - `\M@greeklower@set` ..... 25, 1015, **2135**
  - `\M@greeklowershape` ..... **501**, 2136
  - `\M@greekupper@set` ..... 25, 1014, **2095**
  - `\M@greekuppershape` ..... **500**, 2096
  - `\M@hebrew@set` ..... 25, 1020, **2260**
  - `\M@hebrewshape` ..... **506**, 2261
  - `\M@HModeError` .....  
..... **367**, 2895, 2969, 3033, 3097, 3161
  - `\M@index@assert` ..... 1172, 1176
  - `\M@IntegralItalicFactor` ..... 456, 464
  - `\M@InternalsRestoredError` .... **318**, 692
  - `\M@InvalidOptionError` ..... **291**, 636
  - `\M@InvalidSuboptionError` ..... **299**, 649
  - `\M@keys` ..... **521**, 637
  - `\M@loader` .. 8, 184, 185, 190, 221, 223, 594
  - `\M@lower@set` ..... 25, 1012, **2018**, **2050**
  - `\M@lowershape` ..... **498**, 2019, 2051
  - `\M@MathConstantsNoFontAdjustWarning`  
..... **346**, 801

- $\backslash$ M@MissingControlSequenceError **353**, 810  
 $\backslash$ M@MissingNFSSShapesWarning .. **275**, 554  
 $\backslash$ M@MissingOptionError ..... **307**, 663  
 $\backslash$ M@MissingSuboptionError ..... **312**, 669  
 $\backslash$ M@newfont ..... **561**, 634, 695, 820  
 $\backslash$ M@NewFontCommandInfo ..... **266**, 821  
 $\backslash$ M@NoBaseModeDetectedWarning .. **280**, 578  
 $\backslash$ M@NoFontAdjustError ..... **393**, 914  
 $\backslash$ M@NoLuaotfloadError ..... **102**, 130  
 $\backslash$ M@NoMathfontError .... **10**, 24, 39, 45, 51  
 $\backslash$ M@number@assert ..... 1156, 1167  
 $\backslash$ M@operator@mathcodes . **2304**, **2360**, 2363  
 $\backslash$ M@operator@num ..... **2303**, 2305–2358  
 $\backslash$ M@operator@set ..... 25, 1022, **2301**  
 $\backslash$ M@operatorshape ..... **508**, 2303, 2363  
 $\backslash$ M@Optiondeprecated .. **134**, 141, 144, 147  
 $\backslash$ M@otf@features **226**, **230**, 599, 607, 615, 620  
 $\backslash$ M@p@tch@decl@re ..... **440**, 441  
 $\backslash$ M@parse@option ..... **657**, 705  
 $\backslash$ M@radical@set .... 25, 1024, **2433**, **2456**  
 $\backslash$ M@radicalshape ..... **510**, 2434, 2457  
 $\backslash$ M@retokenize ..... **448**, 449, **449**, 451  
 $\backslash$ M@RuleThicknessFactor ..... 455, 463  
 $\backslash$ M@SetMathConstants ..... **759**, **785**, 803  
 $\backslash$ M@split@colon ..... **558**, 563  
 $\backslash$ M@strip@colon ..... **560**, 565  
 $\backslash$ M@strip@equals ..... **656**, 674  
 $\backslash$ M@SurdHorizontalFactor ..... 458, 465  
 $\backslash$ M@SurdVerticalFactor ..... 457, 466  
 $\backslash$ M@SymbolFontInfo ..... **259**, 727  
 $\backslash$ M@symbols@set ..... 25, 1027, **2498**  
 $\backslash$ M@symbolsshape ..... **513**, 2499  
 $\backslash$ m@th@const@nts@font ..... **784**, 789, 792  
 $\backslash$ m@th@const@nts@font@sh@pe .....  
 ..... **771**, **775**, 782, 794  
 $\backslash$ m@thf@nt ..... 751  
 $\backslash$ M@upper@set ..... 25, 1011, **1959**, **1988**  
 $\backslash$ M@uppershape ..... **497**, 1960, 1989  
 $\backslash$ M@XeTeXLuaTeXError ..... **61**, 83  
 $\backslash$ math@fonts ..... 797, **803**  
 $\backslash$ mathand ..... 2510, 2553  
 $\backslash$ mathbackslash ..... 2383, 2384  
 $\backslash$ mathbb ..... **2893**, 2895  
 $\backslash$ mathbcal ..... **3095**, 3097  
 $\backslash$ mathbf ..... 852  
 $\backslash$ mathbfit ..... 853  
 $\backslash$ mathbfrak ..... **3159**, 3161  
 $\backslash$ mathbfsc ..... 856  
 $\backslash$ mathbfscit ..... 857  
 $\backslash$ mathcal ..... **2967**, 2969  
 $\backslash$ mathconstantsfont ..... 43,  
 334, 339, 342, 343, 347, 755, **760**, 804  
 $\backslash$ mathdollar ..... 2508  
 $\backslash$ mathfont .. 43, 145, 148, 273, 292, 294,  
 300, 302, 308, 310, 313, 315, 324, 326,  
 328, 330, 338, **689**, 741, 750, 754, 1004  
 $\backslash$ mathfrak ..... **3031**, 3033  
 $\backslash$ mathhash ..... 2507, 2551  
 $\backslash$ mathit ..... 851  
 $\backslash$ mathng ..... 2754, 2755  
 $\backslash$ mathnolimitsmode ..... 1036  
 $\backslash$ mathpalette ..... 2558, 2559  
 $\backslash$ mathparagraph ..... 2511  
 $\backslash$ mathpercent ..... 2509, 2552  
 $\backslash$ mathring ..... 2093  
 $\backslash$ mathrm ..... 850  
 $\backslash$ mathsc ..... 854  
 $\backslash$ mathscit ..... 855  
 $\backslash$ mathsection ..... 2512  
 $\backslash$ mathsterling ..... 2513  
 $\backslash$ meaning ..... 434, 437  
 $\backslash$ mem ..... 2274  
 $\backslash$ mid ..... 2542  
 Missing \$ inserted ..... 11  
 Missing control sequence ..... 11  
 Missing Option for  $\backslash$ mathfont ..... 10  
 Missing package fontspec ..... 10  
 Missing Suboption for  $\backslash$ mathfont ..... 10  
 missing X $\TeX$  or Lua $\TeX$  ..... 3  
 $\backslash$ mkern ..... 2454  
 $\backslash$ models ..... 2563  
 $\backslash$ Mu ..... 1855, 2108  
 Multiple characters in argument ..... 11
- N**
- $\backslash$ nabla ..... 2125, 2132, 2576, 2757  
 $\backslash$ napprox ..... 2729  
 $\backslash$ natural ..... 2603  
 $\backslash$ Narrow ..... 2881  
 $\backslash$ nearrow ..... 2880  
 $\backslash$ neg ..... 2514  
 $\backslash$ nequiv ..... 2751  
 $\backslash$ neswarrow ..... 2889  
 $\backslash$ newmathbf ..... 30, 832, 841, 852, 860  
 $\backslash$ newmathbfit ..... 31, 833, 842, 853, 862

<code>\newmathbfsc</code> .....	36, 836, 845, 856
<code>\newmathbfscit</code> .....	37, 837, 846, 857
<code>\newmathbold</code> .....	32, <b>859</b> , 860
<code>\newmathboldit</code> .....	33, <b>861</b> , 862
<code>\newmathfontcommand</code> .....	<b>50</b> , 51, <b>817</b> , 818, 824, 828
<code>\newmathit</code> .....	29, 831, 840, 851
<code>\newmathrm</code> .....	28, 830, 839, 850
<code>\newmathsc</code> .....	34, 834, 843, 854
<code>\newmathscit</code> .....	35, 835, 844, 855
<code>\ngeq</code> .....	2719
<code>\ngsim</code> .....	2734
<code>\nLeftarrow</code> .....	2811
<code>\nleftarrow</code> .....	2809
<code>\nLeftrightarrow</code> .....	2840
<code>\nleq</code> .....	2718
<code>\nlsim</code> .....	2733
no previous font .....	10
<code>\nprec</code> .....	2739
<code>\npreceq</code> .....	2741
<code>\nRrightarrow</code> .....	2780
<code>\nrightarrow</code> .....	2778
<code>\nsim</code> .....	2728
<code>\nsimeq</code> .....	2730
<code>\nsimeqq</code> .....	2731
<code>\nsqsubsetq</code> .....	2712
<code>\nsqsupsetq</code> .....	2713
<code>\nsubset</code> .....	2706
<code>\nsubsetq</code> .....	2708
<code>\nsucc</code> .....	2740
<code>\nsucceq</code> .....	2742
<code>\nsupset</code> .....	2707
<code>\nsupsetq</code> .....	2709
<code>\ntriangleleft</code> .....	2724
<code>\ntrianglelefteq</code> .....	2726
<code>\ntriangleright</code> .....	2725
<code>\ntrianglerighteq</code> .....	2727
<code>\Nu</code> .....	1856, 2109
<code>\nun</code> .....	2275
<code>\Nwarrow</code> .....	2883
<code>\nwarrow</code> .....	2882
<code>\nwsearrow</code> .....	2888
<b>O</b>	
<code>\odiv</code> .....	2635
<code>\oiiintop</code> .....	2491, 2492
<code>\oiintop</code> .....	2489, 2490
<code>\ointop</code> .....	2487, 2488

<code>\Omega</code> .....	1867, 2120
<code>\omega</code> .....	1837, 2160
<code>\Omicron</code> .....	1858, 2111
<code>\omicron</code> .....	1828, 2151
<code>\ominus</code> .....	2634
<code>\operator@font</code> .....	2362
<code>\oslash</code> .....	2636

**P**

Package mathfont Info .....	9
<code>\parallel</code> .....	2543
parse <code>\mathfont</code> arguments .....	22
<code>\partial</code> .....	2517
<code>\Phi</code> .....	1864, 2117
<code>\phi</code> .....	1834, 2157
<code>\Pi</code> .....	1859, 2112
<code>\pi</code> .....	1829, 2152
<code>\pm</code> .....	2528
<code>\prec</code> .....	2691
<code>\precapprox</code> .....	2699
<code>\preceq</code> .....	2693
<code>\preceqq</code> .....	2695
<code>\precnapprox</code> .....	2749
<code>\precneq</code> .....	2743
<code>\precneqq</code> .....	2745
<code>\precnsim</code> .....	2747
<code>\precprec</code> .....	2701
<code>\precsim</code> .....	2697
<code>\prime</code> .....	2505
<code>\proportion</code> .....	2662
<code>\propto</code> .....	2656
<code>\Psi</code> .....	1866, 2119

**Q**

<code>\qeq</code> .....	2688
<code>\qof</code> .....	2280

**R**

<code>\r@@t</code> .....	2438
<code>\radicandoffset</code> .....	459, 467, 2454
<code>\ratio</code> .....	2661
<code>\rbrace</code> .....	2390
<code>\rcirculararrow</code> .....	2891
<code>\Relbar</code> .....	2562, 2563
<code>\relbar</code> .....	2561
<code>\resh</code> .....	2281
<code>\restoremathinternals</code> .	142, 327, 331, <b>924</b>
<code>\rguil</code> .....	1419, 1878, 2396, 2426

<code>\Rho</code> .....	1860, 2113	<code>\simeqq</code> .....	2673, 2674
<code>\rho</code> .....	1830, 2153	<code>\simneqq</code> .....	2732
<code>\Rightarrow</code> .....	2779	<code>\sqcap</code> .....	2623
<code>\rightarrowtail</code> .....	2794	<code>\sqdot</code> .....	2641
<code>\rightarrowtoobar</code> .....	2803	<code>\sqminus</code> .....	2640
<code>\Rightbararrow</code> .....	2786	<code>\sqplus</code> .....	2638
<code>\rightbararrow</code> .....	2784, 2785	<code>\sqrtsign</code> .....	2439, 2454
<code>\rightbrace</code> .....	2375, 2430	<code>\sqsubseteq</code> .....	2650
<code>\rightrightarrow</code> .....	2791	<code>\sqsubsetneq</code> .....	2714
<code>\rightharpoonowdown</code> .....	2793	<code>\sqsupseteq</code> .....	2651
<code>\rightharpoonup</code> .....	2792	<code>\sqsupsetneq</code> .....	2715
<code>\rightleftarrows</code> .....	2847	<code>\sqtimes</code> .....	2639
<code>\rightoplusarrow</code> .....	2795	<code>\ssharp</code> .....	2606
<code>\rightrightarrows</code> .....	2805	<code>\sssim</code> .....	2676
<code>\rightrightarrows</code> .....	2806	<code>\st@ck@fl@trel</code> .....	938, 939
<code>\rightsquigarrow</code> .....	2797	<code>\stack@flatrel</code> .....	<b>937</b> , 2558, 2559
<code>\rightwvearrow</code> .....	2796	<code>\star</code> .....	2628
<code>\rightwhitearrow</code> .....	2804	<code>\stareq</code> .....	2685
<code>\ringeq</code> .....	2681	<code>\Stigma</code> .....	2174
robust commands .....	80	<code>\stigma</code> .....	2186
<code>\rootbox</code> .....	2446, 2449, 2451	<code>\strip@prefix</code> .....	434, 437
<code>\rrguil</code> .....	1421, 1880, 2402, 2428	suboption <i>italic</i> .....	22, 25
<code>\Rrightarrow</code> .....	2781	suboption <i>roman</i> .....	22, 25
<code>\RuleThicknessFactor</code> ...	17, 898, 911, 918	suboption <i>upright</i> .....	22
<b>S</b>			
<code>\samekh</code> .....	2276	<code>\subseteq</code> .....	2646
<code>\Sampi</code> .....	2171	<code>\subsetneq</code> .....	2710
<code>\sampi</code> .....	2183	<code>\succ</code> .....	2692
<code>\San</code> .....	2176	<code>\succapprox</code> .....	2700
<code>\san</code> .....	2188	<code>\succeq</code> .....	2694
<code>\scantextokens</code> .....	443	<code>\succeqq</code> .....	2696
<code>\scantokens</code> .....	448	<code>\succnapprox</code> .....	2750
<code>\scdefault</code> .....	541, 542, 544, 834–837	<code>\succneq</code> .....	2744
<code>\Searrow</code> .....	2885	<code>\succneqq</code> .....	2746
<code>\searrow</code> .....	2884	<code>\succnsim</code> .....	2748
<code>\seq</code> .....	2677	<code>\succsim</code> .....	2698
<code>\seriesdefault</code> .....	793	<code>\succsucc</code> .....	2702
<code>\setfont</code> .....	16, <b>753</b> , 758, 1004, 1009	<code>\supseteq</code> .....	2647
<code>\setmathfontcommands</code> .....	756, <b>849</b> , 858	<code>\supsetneq</code> .....	2711
<code>\setminus</code> .....	2533	<code>\surd</code> .....	1908, 2435, 2458
<code>\sharp</code> .....	2604	<code>\surdbox</code> .....	
<code>\shin</code> .....	2282		453, 2440, 2442, 2443, 2445–2447, 2452
<code>\Sho</code> .....	2175	<code>\SurdHorizontalFactor</code> ..	20, 899, 911, 920
<code>\Sigma</code> .....	1861, 2114	<code>\SurdVerticalFactor</code> ...	19, 899, 912, 921
<code>\sigma</code> .....	1831, 2154	<code>\Svarrow</code> .....	2887
<code>\sim</code> .....	2539, 2558, 2559	<code>\swarrow</code> .....	2886
<code>\simeq</code> .....	2558, <b>2581</b> , 2671	<code>\symMupright</code> .....	791

<b>T</b>		<code>\vardot</code> . . . . . 2630	
<code>\Tau</code> . . . . . 1862, 2115		<code>\varDigamma</code> . . . . . 2178	
<code>\tau</code> . . . . . 1832, 2155		<code>\vardigamma</code> . . . . . 2190	
<code>\tav</code> . . . . . 2283		<code>\varepsilon</code> . . . . . 1839, 2162	
terminal . . . . . 25, 32		<code>\varkaf</code> . . . . . 2284	
<code>\tet</code> . . . . . 2270		<code>\varkappa</code> . . . . . 2163	
<code>\textbackslash</code> . . . . . 2383		<code>\varKoppa</code> . . . . . 2179	
<code>\textng</code> . . . . . <b>2752</b> , 2755		<code>\varkoppa</code> . . . . . 2191	
<code>\therefore</code> . . . . . 2659		<code>\varmem</code> . . . . . 2285	
<code>\Theta</code> . . . . . 1851, 2104		<code>\varnun</code> . . . . . 2286	
<code>\theta</code> . . . . . 1821, 2144		<code>\varpe</code> . . . . . 2287	
<code>\tilde</code> . . . . . 2094		<code>\varphi</code> . . . . . 1843, 2167	
<code>\tracinglostchars</code> . . . . . 953, 954		<code>\varrho</code> . . . . . 1841, 2165	
<code>\triangleeq</code> . . . . . 2686		<code>\varSampi</code> . . . . . 2177	
<code>\triangleleft</code> . . . . . 2652		<code>\varsampi</code> . . . . . 2189	
<code>\trianglelefteq</code> . . . . . 2654		<code>\varsetminus</code> . . . . . 2631	
<code>\triangleright</code> . . . . . 2653		<code>\varsigma</code> . . . . . 1842, 2166	
<code>\trianglerighteq</code> . . . . . 2655		<code>\varTheta</code> . . . . . 1868, 2121	
<code>\tsadi</code> . . . . . 2279		<code>\vartheta</code> . . . . . 1840, 2164	
<code>\twoheadrightarrow</code> . . . . . 2870		<code>\vartsadi</code> . . . . . 2288	
<code>\twoheadleftarrow</code> . . . . . 2833		<code>\vav</code> . . . . . 2267	
<code>\twoheadrightarrow</code> . . . . . 2802		<code>\vdash</code> . . . . . 2600	
<code>\twoheaduparrow</code> . . . . . 2856		<code>\veeeq</code> . . . . . 2684	
<b>U</b>		<code>\vert</code> . . . . . <b>2382</b>	
<code>\Udelcode</code> . . . . . 2376–2381		<code>\vphantom</code> . . . . . 2441	
<code>\Udelimitter</code> . . . . . 2385, 2388, 2391, 2394, 2397, 2400, 2403, 2406, 2409, 2412, 2415		<b>W</b>	
unable to load . . . . . 3, 4		<code>\wclubsuit</code> . . . . . 2611	
<code>\unexpanded</code> . . . . . 448		<code>\wdiamondsuit</code> . . . . . 2612, 2617	
<code>\uparrows</code> . . . . . 2857		<code>\wedgreek</code> . . . . . 2683	
<code>\upbararrow</code> . . . . . 2852		<code>\wheartsuit</code> . . . . . 2613, 2616	
<code>\updasharrow</code> . . . . . 2853		<code>\wp</code> . . . . . 2587	
<code>\updownarrows</code> . . . . . 2876		<code>\wspadesuit</code> . . . . . 2614	
<code>\updownharpoons</code> . . . . . 2878		<b>X</b>	
<code>\upharpoonleft</code> . . . . . 2854		<code>\XeTeXrevision</code> . . . . . 55, 995	
<code>\upharpoonright</code> . . . . . 2855		<code>\Xi</code> . . . . . 1857, 2110	
<code>\Upsilon</code> . . . . . 1863, 2116		<b>Y</b>	
<code>\upsilon</code> . . . . . 1833, 2156		<code>\yod</code> . . . . . 2271	
<code>\upuparrows</code> . . . . . 2860		Your command is invalid without Lua-based <i>12</i>	
<code>\upwhitearrow</code> . . . . . 2858		Your <code>\mathconstants</code> on line . . . . . <i>11</i>	
<code>\upwhitebararrow</code> . . . . . 2859		<b>Z</b>	
<code>\Uradical</code> . . . . . 2437		<code>\zayin</code> . . . . . 2268	
<code>\Uparrow</code> . . . . . 2851		<code>\Zeta</code> . . . . . 1849, 2102	
<b>V</b>		<code>\zeta</code> . . . . . 1819, 2142	
<code>\varbeta</code> . . . . . 1838, 2161		<code>\zigzagarrow</code> . . . . . 2866, 2867	