

Package `mathfont` v. 2.2b Implementation

Conrad Kosowsky

August 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}[2023/08/21 v. 2.2b Package mathfont]
```

We specify conditionals that we will use later in handling options and setup.

```
3 \newif\ifM@XeTeXLuaTeX % is engine one of xetex or luatex?
```

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.

```

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?

```

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.

```

8 \long\def\@gobbletwo@brackets[#1]#2{}
9 \def\M@NoMathfontError#1{\PackageError{mathfont}
10  {\MessageBreak Invalid command\MessageBreak
11  \string#1 on line \the\inputlineno}
12  {Your command was ignored. I couldn't\MessageBreak
13  load mathfont, so I never defined this\MessageBreak
14  control sequence.}}
15 \protected\def\mathfont{\M@NoMathfontError\mathfont
16  \@ifnextchar[\@gobbletwo@brackets\@gobble}
17 \protected\def\setfont{\M@NoMathfontError\setfont\@gobble}
18 \protected\def\mathconstantsfont{\M@NoMathfontError\mathconstantsfont
19  \@ifnextchar[\@gobbletwo@brackets\@gobble}
20 \protected\def\newmathrm{\M@NoMathfontError\newmathrm\@gobbletwo}
21 \protected\def\newmathit{\M@NoMathfontError\newmathit\@gobbletwo}
22 \protected\def\newmathbf{\M@NoMathfontError\newmathbf\@gobbletwo}
23 \protected\def\newmathbfi{\M@NoMathfontError\newmathbf\@gobbletwo}
24 \protected\def\newmathbold{\M@NoMathfontError\newmathbold\@gobbletwo}
25 \protected\def\newmathboldit{\M@NoMathfontError\newmathbold\@gobbletwo}
26 \protected\def\newmathsc{\M@NoMathfontError\newmathsc\@gobbletwo}
27 \protected\def\newmathscit{\M@NoMathfontError\newmathscit\@gobbletwo}
28 \protected\def\newmathbfsc{\M@NoMathfontError\newmathbfsc\@gobbletwo}
29 \protected\def\newmathbfscit{\M@NoMathfontError\newmathbfscit\@gobbletwo}
30 \protected\def\newmathfontcommand{%
31  \M@NoMathfontError\newmathfontcommand\@gobblefour}
32 \protected\def\RuleThicknessFactor{%
33  \M@NoMathfontError\RuleThicknessFactor\@gobble}
34 \protected\def\IntegralItalicFactor{%
35  \M@NoMathfontError\IntegralItalicFactor\@gobble}
36 \protected\def\SurdVerticalFactor{%
37  \M@NoMathfontError\SurdVerticalFactor\@gobble}
38 \protected\def\SurdHorizontalFactor{%
39  \M@NoMathfontError\SurdHorizontalFactor\@gobble}
40 \protected\def\CharmLine{\M@NoMathfontError\CharmLine\@gobble}
41 \protected\def\CharmFile{\M@NoMathfontError\CharmFile\@gobble}

```

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

```

42 \ifdefined\directlua

```

```

43 \M@XeTeXLuaTeXtrue
44 \fi
45 \ifdefined\XeTeXrevision
46 \M@XeTeXLuaTeXtrue
47 \fi

```

The package can raise two fatal errors: one if the engine is not XeTeX or LuaTeX (and cannot load OpenType fonts) and one if TeX 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 $\text{L}^{\text{A}}\text{TeX}$ adds to the error message. Notice that we expand the error before the `\endgroup`—this is because we need to switch `\M@NoFontspecError` 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.

```

48 \ifM@XeTeXLuaTeX\else
49 \begingroup
50 \catcode\ =12\relax
51 \def\M@XeTeXLuaTeXError{\GenericError{}}%
52 {\MessageBreak\MessageBreak
53 Package mathfont error:%
54 \MessageBreak\MessageBreak
55 *****\MessageBreak
56 *
57 *     UNABLE TO     *\MessageBreak
58 *     LOAD MATHFONT *\MessageBreak
59 *
60 *     Missing XeTeX *\MessageBreak
61 *     or LuaTeX    *\MessageBreak
62 *
63 *****\MessageBreak\@gobbletwo}%
64 {See the mathfont package documentation for explanation.}%
65 {I need XeTeX or LuaTeX to use mathfont. It\MessageBreak
66 looks like the current engine is something\MessageBreak
67 else, so I'm going to stop reading in the\MessageBreak
68 package file now. (You won't be able to use\MessageBreak
69 commands from mathfont in your document.) To\MessageBreak
70 load mathfont correctly, please retypeset your\MessageBreak
71 document with one of those two engines.^^J}}%
72 \expandafter\endgroup
73 \M@XeTeXLuaTeXError
74 \AtEndOfPackage{\typeout{:: mathfont :: Failed to load\on@line.}}
75 \expandafter\endinput % we should \endinput with a balanced conditional
76 \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 \XeTeX , issue a warning.

```

77 \ifdefined\directlua
78 \M@adjust@fonttrue % if engine is LuaTeX, adjust font by default
79 \IfFileExists{luaotfload.sty}
80   {\M@Noluaotfloadfalse\RequirePackage{luaotfload}}{\M@Noluaotfloadtrue}
81 \else
82   \PackageWarningNoLine{mathfont}{%
83     The current engine is XeTeX, but as\MessageBreak
84     of mathfont version 2.0, LuaTeX is\MessageBreak
85     recommended. Consider compiling with\MessageBreak
86     LuaLaTeX. Certain features will not\MessageBreak
87     work with XeTeX}
88 \fi

```

If the engine is $\text{Lua}\TeX$, we absolutely must have `luaotfload` because $\text{Lua}\TeX$ 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 $\text{L}\TeX$ versions load `luaotfload` as part of the format, but it never hurts to double check.

```

89 \ifM@Noluaotfload % false by default; true if LuaTeX AND no luaotfload.sty
90 \begingroup
91 \catcode`\ =12\relax
92 \def\M@NoluaotfloadError{\GenericError{}}%
93 {\MessageBreak\MessageBreak
94 Package mathfont error:%
95 \MessageBreak\MessageBreak
96 *****\MessageBreak
97 *
98 *     UNABLE TO     *\MessageBreak
99 *     LOAD MATHFONT *\MessageBreak
100 *
101 *     Cannot find the *\MessageBreak
102 *     file luaotfload.sty *\MessageBreak
103 *
104 *****\MessageBreak\@gobbletwo}%
105 {You are likely seeing this message because you haven't^^J%
106 installed luaotfload. Check your TeX distribution for a^^J%
107 list of the packages on your system.^^J^^J%
108 See the mathfont documentation for further explanation.}%
109 {You're in trouble here. It looks like the current\MessageBreak
110 engine is LuaTeX, so I need the luaotfload package\MessageBreak
111 to make mathfont work properly. However, I can't\MessageBreak
112 find luaotfload, which likely means something is\MessageBreak
113 wrong with your TeX installation. I'm going to stop\MessageBreak
114 reading in the mathfont package file. (You won't be\MessageBreak
115 able to use commands from mathfont in your document.)\MessageBreak
116 To load mathfont work correctly, make sure you have\MessageBreak

```

```

117 installed luaotfload.sty in a directory searchable\MessageBreak
118 by TeX or compile with XeLaTeX.^^J}}%
119 \expandafter\endgroup
120 \M@NoluaotfloadError
121 \AtEndOfPackage{\typeout{:: mathfont :: Failed to load\on@line.}}
122 \expandafter\endinput % we should \endinput with a balanced conditional
123 \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.

```

124 \def\M@Optiondeprecated#1#2{\PackageError{mathfont}
125   {Option "#1" deprecated}
126   {Your option was ignored. Please\MessageBreak
127   use #2\MessageBreak
128   instead. For more information,\MessageBreak
129   see the mathfont documentation.}}

```

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

```

130 \DeclareOption{packages}{%
131   \M@Optiondeprecated{packages}
132   {the macro \string\restoremathinternals}}
133 \DeclareOption{operators}{%
134   \M@Optiondeprecated{operators}
135   {the bigops keyword with \string\mathfont}}
136 \DeclareOption{no-operators}{%
137   \M@Optiondeprecated{no-operators}
138   {the bigops keyword with \string\mathfont}}

```

Easter egg!

```

139 \DeclareOption{easter-egg}{%
140   \ifE@sterEggDecl@red\else
141     \E@sterEggDecl@redtrue
142     \def\EasterEggUpdate{\show\E@sterEggUpd@te}
143     \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
144       Okay, opening your Easter egg.^^J^^J}
145     \EasterEggUpdate
146     \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
147       Uh oh. It looks like^^J%
148       your Easter egg flew^^J%
149       out the window. I don't^^J%
150       suppose you know the^^J%
151       best kind of bait to^^J%
152       lure an egg?^^J^^J}
153     \EasterEggUpdate
154     \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
155       Still wrangling. Try back later.^^J^^J}
156   \AtBeginDocument{\bgroup

```

```

157     \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J:%
158     If we have zero eggs^^J%
159     and zero bunnies, how^^J%
160     many gnats does it take^^J%
161     to change a lightbulb??^^J^^J}
162     \EasterEggUpdate
163     \egroup}
164 \AtEndDocument{%
165     \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
166     Happy, happy day! Happy,^^J%
167     happy day! Clap your hands,^^J%
168     and be glad your hovercraft^^J%
169     isn't full of eels!^^J^^J}
170     \EasterEggUpdate
171     \let\E@sterEggUpd@te\relax
172     \let\EasterEggUpdate\relax}
173 \fi}% my easter egg :)

```

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

```

174 \DeclareOption{adjust}{\M@adjust@fonttrue}
175 \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`.

```

176 \DeclareOption*{\M@font@loadedtrue\edef\M@font@load{\CurrentOption}}
177 \ProcessOptions*

```

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

```

178 \ifdefined\directlua
179   \ifM@adjust@font
180     \AtEndOfPackage{%
181       \typeout{:: mathfont :: Lua-based font adjustments enabled.}}
182   \else
183     \AtEndOfPackage{%
184       \typeout{:: mathfont :: Lua-based font adjustments disabled.}}
185   \fi
186 \else

```

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

```

187 \AtEndOfPackage{%
188   \typeout{:: mathfont :: Lua-based font adjustments disabled.}}
189 \ifM@adjust@font
190   \AtEndOfPackage{%
191     \PackageError{mathfont}{Option^^J"adjust" ignored with XeTeX}
192     {Your package option "adjust" was ignored.\MessageBreak
193     This option works only with LuaTeX, and it\MessageBreak

```

```

194     looks like the current engine is XeTeX. To\MessageBreak
195     enable Lua-based font adjustments, typeset\MessageBreak
196     with LuaLaTeX.^^J}}
197     \M@adjust@fontfalse
198     \fi
199 \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.

```

200 \def\M@SymbolFontInfo#1#2#3#4{\wlog{^^JPackage mathfont Info:
201   Declaring new symbol font from #1!^^J%

```

Table 1: Various Messages and Errors and Their Uses

Command	Use
<code>\M@FontChangeInfo</code>	Use a symbol font for some characters
<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@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>\mathcontantsfont</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@NoFontspecFamilyError</code>	Improper option <code>fontspec</code> for <code>\mathfont</code>
<code>\M@NoFontspecError</code>	Option <code>fontspec</code> for <code>\mathfont</code> declared without having loaded <code>fontspec</code>
<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

```

202 NFSS Family Name: \space#2^^J%
203 Series/Shape Info: #3^^J%
204 Symbol Font Name: \space#4^^J}}
205 \def\M@FontChangeInfo#1#2{\wlog{Package mathfont Info:
206 Setting #1 chars to #2!}}
207 \def\M@NewFontCommandInfo#1#2#3#4#5{\wlog{^^JPackage mathfont Info:
208 Creating \string#1 using #2!^^J%
209 NFSS Family Name: \space#3^^J%
210 Series/Shape Info: #4/#5^^J}}
211 \def\M@CharsSetWarning#1{\PackageWarning{mathfont}
212 {I already set the font for\MessageBreak
213 #1 chars, so I'm ignoring\MessageBreak
214 this option for \string\mathfont\space
215 on line \the\inputlineno\@gobble}}

```

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

```

216 \def\M@DeprecatedWarning#1#2{\PackageWarning{mathfont}
217 {Your \string#1\space command on\MessageBreak
218 line \the\inputlineno\space is deprecated, and I\MessageBreak
219 replaced it with \string#2\@gobble}}

```

Error messages associated with `\mathfont`.

```

220 \def\M@InvalidOptionError#1{\PackageError{mathfont}
221 {Invalid^^Joption "#1" for \string\mathfont\on@line}
222 {Hm. You used a keyword that isn't actually an optional\MessageBreak
223 argument for \string\mathfont. Check
224 that you spelled the keyword\MessageBreak
225 correctly. Otherwise, I'm not sure what's wrong. Is this\MessageBreak
226 option listed in the package documentation? In any event,\MessageBreak
227 I'm going to ignore it.^^J}}
228 \def\M@InvalidSuboptionError#1{\PackageError{mathfont}
229 {Invalid^^Jsuboption "#1" for \string\mathfont\on@line}
230 {Hm. You used a keyword that isn't actually a suboption\MessageBreak
231 for \string\mathfont. Check that you
232 spelled the keyword correctly.\MessageBreak
233 Otherwise, I'm not sure what's wrong. Is this suboption\MessageBreak
234 listed in the package documentation? In any event, I'm\MessageBreak
235 going to ignore it.^^J}}
236 \def\M@MissingOptionError{\PackageError{mathfont}
237 {Missing^^Joption for \string\mathfont\on@line}
238 {It looks like you included a , or = in\MessageBreak
239 the optional argument of \string\mathfont\space but\MessageBreak
240 didn't put anything before it.^^J}}
241 \def\M@MissingSuboptionError{\PackageError{mathfont}
242 {Missing^^Jsuboption for \string\mathfont\on@line}
243 {It looks like you included an = somewhere\MessageBreak
244 but didn't put the suboption after it. Either\MessageBreak
245 that or you typed == instead of = in the\MessageBreak
246 optional argument of \string\mathfont.^^J}}

```



```

247 \def\M@InternalsRestoredError{\PackageError{mathfont}
248   {Internal^^Jcommands restored}
249   {This package slightly changes two LaTeX\MessageBreak
250   internal commands, and you really shouldn't\MessageBreak
251   be loading new math fonts without those\MessageBreak
252   adjustments. What happened here is that you\MessageBreak
253   used \string\mathfont\space in a situation where those\MessageBreak
254   two commands retain their original defini-\MessageBreak
255   tions. Presumably you used \string\mathfont\space after\MessageBreak
256   calling the \string\restoremathinternals\space command.\MessageBreak
257   I'm going to ignore this call to \string\mathfont.\MessageBreak
258   Try typesetting this document with all\MessageBreak
259   \string\mathfont\space commands placed before you call\MessageBreak
260   \string\restoremathinternals.^^J}}
261 \def\M@NoFontspecFamilyError{\PackageError{mathfont}
262   {No previous^^Jfont loaded by fontspec}
263   {You called \string\mathfont\space
264   with the argument "fontspec" \MessageBreak
265   on line \the\inputlineno,
266   and that tells me to use the previous \MessageBreak
267   font loaded by the fontspec package. However, it \MessageBreak
268   looks like you haven't loaded any fonts yet with \MessageBreak
269   fontspec. To resolve this error, try using for \MessageBreak
270   example \string\setmainfont\space
271   before calling \string\mathfont.^^J}}
272 \def\M@NoFontspecError{\PackageError{mathfont}
273   {Missing^^Jpackage fontspec}
274   {You called \string\mathfont\space
275   with the argument "fontspec" \MessageBreak
276   on line \the\inputlineno,
277   and that tells me to use the previous \MessageBreak
278   font loaded by the fontspec package. However, you\MessageBreak
279   haven't loaded fontspec, so some things are about\MessageBreak
280   to get messed up. To resolve this error, load\MessageBreak
281   fontspec before calling \string\mathfont.^^J}}
Error messages for \mathconstantsfont.
282 \def\M@BadMathConstantsFontError#1{\PackageError{mathfont}
283   {Invalid\MessageBreak font specifier for
284   \string\mathconstantsfont:\MessageBreak"#1"}
285   {Your command was ignored--I can't parse your argument.\MessageBreak
286   Please make sure to use text that you have previously\MessageBreak
287   fed to \string\mathfont\space for the argument of
288   \string\mathconstantsfont.^^J}}
289 \def\M@BadMathConstantsFontTypeError#1{\PackageError{mathfont}
290   {Invalid\MessageBreak font specifier for
291   \string\mathconstantsfont:\MessageBreak"#1"}
292   {The optional argument of \string\mathconstantsfont\MessageBreak

```

```

293 should be "upright" or "italic." Right now,\MessageBreak
294 it's "#1."^^J}}
295 \def\M@LuaTeXOnlyWarning{\PackageWarning{mathfont}
296   {Your \string\mathconstantsfont\space
297   on line \the\inputlineno\space is\MessageBreak
298   for LuaTeX only, and I'm ignoring it@\gobble}}

```

Error messages for the `\newmathrm`, etc. commands.

```

299 \def\M@MissingControlSequenceError#1#2{\PackageError{mathfont}
300   {Missing control sequence\MessageBreak
301   for\string#1\MessageBreak on input line \the\inputlineno}
302   {Your command was ignored. Right now the\MessageBreak
303   first argument of \string#1\space is "#2."\MessageBreak
304   Please use a control sequence instead.^^J}}
305 \def\M@DoubleArgError#1#2{\PackageError{mathfont}
306   {Multiple characters in\MessageBreak
307   first argument of \string#1\MessageBreak
308   on input line \the\inputlineno}
309   {Your command was ignored. Right now the\MessageBreak
310   first argument of \string#1\space is "#2,"MessageBreak
311   which is multiple characters. Please use\MessageBreak
312   a single character instead.^^J}}
313 \def\M@HModeError#1{\PackageError{mathfont}
314   {Missing \string$ inserted\MessageBreak
315   on input line line \the\inputlineno}
316   {I generated an error because
317   you used \string#1\space outside of\MessageBreak
318   math mode. I inserted a \string$
319   before your \string#1, so we\MessageBreak
320   should be all good now.^^J}}

```

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

```

321 \def\M@ForbiddenCharmLine#1{\PackageError{mathfont}
322   {Forbidden charm info contains #1}
323   {The argument of your \string\CharmLine\space
324   macro on line \the\inputlineno\MessageBreak
325   contains the character #1, which will mess me up\MessageBreak
326   if I try to read it, so I'm ignoring this call\MessageBreak
327   to \string\CharmLine. To resolve this error, make sure\MessageBreak
328   your charm information contains only integers,\MessageBreak
329   floats, asterisks, commas, and spaces.^^J}}
330 \def\M@ForbiddenCharmFile#1{\PackageError{mathfont}
331   {Forbidden charm info contains #1}
332   {One of the lines in your \string\CharmFile\space
333   from line \the\inputlineno\MessageBreak
334   contains the character #1, which will mess me up\MessageBreak
335   if I try to read it, so I'm ignoring this line\MessageBreak
336   from your file. To resolve this error, make sure\MessageBreak
337   your charm information contains only integers,\MessageBreak

```

```

338 floats, asterisks, commas, and spaces.^^J}}
339 \def\M@NoFontAdjustError#1{\PackageError{mathfont}
340 {Your command \MessageBreak\string#1 is invalid\MessageBreak
341 without Lua-based font adjustments}
342 {You haven't enabled Lua-based font adjustments,\MessageBreak
343 but the macro you called won't do anything without\MessageBreak
344 them. I'm going to ignore your command for now. To\MessageBreak
345 resolve this error, load mathfont with the package\MessageBreak
346 option "adjust" or compile with LuaLaTeX.^^J}}
347 \def\M@BadIntegerError#1#2{\PackageError{mathfont}
348 {Bad argument for\MessageBreak\string#1}
349 {Your command was ignored. Please make sure\MessageBreak
350 that your argument of \string#1\space\MessageBreak
351 is a nonnegative integer. Right now it's\MessageBreak
352 "#2".^^J}}

```

3 Default Settings

We do not want fontspec making changes to mathematics. If the user has loaded the package, we set `\g__fontspec_math_bool` to false. Otherwise, we pass the `no-math` option to the package in case the user loads it later.

```

353 \@ifpackageloaded{fontspec}
354 {\wlog{Package mathfont Info: Package fontspec detected.}
355  \wlog{Package mathfont Info: Setting \string\g__fontspec_math_bool
356   to false.}
357  \csname bool_set_false:N\expandafter\endcsname
358  \csname g__fontspec_math_bool\endcsname}
359 {\wlog{Package mathfont Info: Package fontspec not detected.}
360  \wlog{Package mathfont Info: Will pass no-math option to fontspec
361   if it gets loaded.}
362  \PassOptionsToPackage{no-math}{fontspec}}

```

We save four macros from the L^AT_EX 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.

```

363 \let\@@set@mathchar\set@mathchar
364 \let\@@set@mathsymbol\set@mathsymbol
365 \let\@@set@mathaccent\set@mathaccent
366 \let\@@DeclareSymbolFont\DeclareSymbolFont
367 \@onlypreamble\@@set@mathchar
368 \@onlypreamble\@@set@mathsymbol
369 \@onlypreamble\@@set@mathaccent
370 \@onlypreamble\@@DeclareSymbolFont
371 \wlog{Package mathfont Info: Adapting \noexpand\set@mathchar for unicode.}
372 \wlog{Package mathfont Info: Adapting \noexpand\set@mathsymbol for unicode.}
373 \wlog{Package mathfont Info: Adapting \noexpand\set@mathaccent for unicode.}

```

```
374 \wlog{Package mathfont Info: Increasing upper bound on
375 \noexpand\DeclareSymbolFont to 256.}
```

Kernel command to set math characters from keystrokes.

```
376 \def\set@mathchar#1#2#3#4{%
377 \multiply\count\z@ by 16\relax
378 \advance\count\z@\count\tw@
379 \global\Umathcode`#2=\mathchar@type#3+#1+\count\z@\relax}
```

Kernel command to set math characters from control sequences.

```
380 \def\set@mathsymbol#1#2#3#4{%
381 \multiply\count\z@ by 16\relax
382 \advance\count\z@\count\tw@
383 \global\Umathchardef#2=\mathchar@type#3+#1+\count\z@\relax}
```

Kernel command to set accents.

```
384 \def\set@mathaccent#1#2#3#4{%
385 \multiply\count\z@ by 16\relax
386 \advance\count\z@\count\tw@
387 \protected\xdef#2{%
388 \Umathaccent\mathchar@type#3+\number#1+\the\count\z@\relax}}
```

We increase the upper bound on the number of symbol fonts to be 256. Lua \TeX and X \LaTeX allow up to 256 math families, but the 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 \LaTeX and Lua \TeX . 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 the way \TeX stores and expands parameter symbols inside macros.

As of November 2022, the L \TeX kernel has redefined `\DeclareSymbolFont@m@dropped` to have the same definition as the old `\DeclareSymbolFont`, and now `\DeclareSymbolFont` is a wrapper around this 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`.

```
389 \ifx\DeclareSymbolFont@m@dropped\@undefined
390 \edef\@tempa{\expandafter\strip@prefix\meaning\DeclareSymbolFont}
391 \def\@tempb{\def\DeclareSymbolFont##1##2##3##4##5}
392 \else
393 \edef\@tempa{\expandafter\strip@prefix\meaning\DeclareSymbolFont@m@dropped}
394 \def\@tempb{\def\DeclareSymbolFont@m@dropped##1##2##3##4##5}
395 \fi
396 \def\M@p@tch@decl@re#1<15#2\@nil{#1<\e@mathgroup@top#2}
397 \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 Lua \TeX and a safe version of `\scantokens` in X \LaTeX . 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`.

```
398 \ifdefined\directlua
399   \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 \TeX 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`.

```
400 \else
401   \begingroup
402   \everyeof{\noexpand}
403   \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`.

```
404   \edef\M@retokenize{\noexpand\scantokens{\noexpand\unexpanded{\M@DecSymDef}}}
405   \edef\M@retokenize{\M@retokenize}
406   \expandafter\endgroup
407   \expandafter\@tempb\expandafter{\M@retokenize}
408 \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.

```
409 \newbox\surdbox
410 \newcount\M@count
411 \newcount\M@rule@thickness@factor
412 \newcount\M@integral@italic@factor
413 \newcount\M@surd@vertical@factor
414 \newcount\M@surd@horizontal@factor
415 \newmuskip\radicandoffset
416 \newread\M@Charm
417 \newtoks\M@toks
418 \M@count\z@
419 \M@rule@thickness@factor@m
420 \M@integral@italic@factor=400\relax
421 \M@surd@horizontal@factor@m
422 \M@surd@vertical@factor@m
423 \radicandoffset=3mu\relax
```

Necessary booleans and default math font shapes.

```

424 \newif\ifM@upper
425 \newif\ifM@lower
426 \newif\ifM@diacritics
427 \newif\ifM@greekupper
428 \newif\ifM@greeklower
429 \newif\ifM@agreekupper
430 \newif\ifM@agreeklower
431 \newif\ifM@cyrillicupper
432 \newif\ifM@cyrilliclower
433 \newif\ifM@hebrew
434 \newif\ifM@digits
435 \newif\ifM@operator
436 \newif\ifM@symbols
437 \newif\ifM@extsymbols
438 \newif\ifM@delimiters
439 \newif\ifM@radical
440 \newif\ifM@arrows
441 \newif\ifM@bigops
442 \newif\ifM@extbigops
443 \newif\ifM@bb
444 \newif\ifM@cal
445 \newif\ifM@frak
446 \newif\ifM@bcal
447 \newif\ifM@bfrak
448 \newif\if@optionpresent
449 \newif\if@suboptionpresent
450 \newif\ifM@arg@good
451 \newif\ifM@Decl@reF@mily
452 \newif\ifM@Decl@reF@milyB@se
453 \newif\ifM@fromCharmFile

```

Default shapes.

```

454 \def\M@uppershape{italic}           % latin upper
455 \def\M@lowershape{italic}          % latin lower
456 \def\M@diacriticssshape{upright}  % diacritics
457 \def\M@greekuppershape{upright}   % greek upper
458 \def\M@greeklowershape{italic}    % greek lower
459 \def\M@agreekuppershape{upright}  % ancient greek upper
460 \def\M@agreeklowershape{italic}   % ancient greek lower
461 \def\M@cyrillicuppershape{upright} % cyrillic upper
462 \def\M@cyrilliclowershape{italic} % cyrillic lower
463 \def\M@hebrewshape{upright}       % hebrew
464 \def\M@digitssshape{upright}      % numerals
465 \def\M@operatorshape{upright}     % operator font
466 \def\M@delimitersshape{upright}  % delimiters
467 \def\M@radicalshape{upright}      % surd
468 \def\M@bigopssshape{upright}      % big operators

```

```

469 \def\M@extbigopsshape{upright}      % extended big operators
470 \def\M@symbolsshape{upright}       % basic symbols
471 \def\M@extsymbolsshape{upright}    % extended symbols
472 \def\M@arrowsshape{upright}        % arrows
473 \def\M@bbshape{upright}            % blackboard bold
474 \def\M@calshape{upright}           % caligraphic
475 \def\M@frakshape{upright}          % fraktur
476 \def\M@bcalshape{upright}          % bold caligraphic
477 \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.

```

478 \def\M@keys{upper,lower,diacritics,greekupper,%
479   greeklower,agreekupper,agreeklower,cyrillicupper,%
480   cyrilliclower,hebrew,digits,operator,delimiters,%
481   radical,bigops,extbigops,symbols,extsymbols,arrows,%
482   bb,cal,frak,bcal,bfrak}
483 \def\M@defaultkeys{upper,lower,diacritics,greekupper,%
484   greeklower,digits,operator,symbols}

```

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

```

485 \ifM@adjust@font
486   \edef\M@defaultkeys{\M@defaultkeys,delimiters,radical,bigops}
487 \fi

```

Default OpenType features for loading fonts.

```

488 \def\M@otf@features{script=latin;language=DFLT;%
489   tlig=true;liga=true;smcp=false;lnum=true}
490 \def\M@otf@features@sc{script=latin;language=DFLT;%
491   tlig=true;liga=true;smcp=true;lnum=true}

```

4 Fontloader

We come to the fontloader. The main font-loading macro is `\M@newfont`, and it is basically a wrapper around code we would expect to see in a typical `fd` file. Advanced users: please do not call `\M@newfont` directly because it may change without warning. Instead call `\mathfont` with the `empty` keyword and extract the NFSS family name from `\M@f@ntn@me` or `\M@f@ntn@meb@se`. Our general approach is to feed the mandatory argument of `\mathfont` to `\M@newfont`, check if we have reason to believe that the font corresponds to an entry already in the NFSS, and declare the font family and font shapes as necessary. If `fontspec` is loaded, we pass the entire argument to `fontspec`. If not, `mathfont` handles the font declaration internally. When `mathfont` declares a font family in the NFSS, it does so twice, once using the information provided (which typically results in a font in node mode) and once using the information provided with `mode=base` (which results in a font in base mode). The first declaration uses the entire mandatory argument of `\mathfont` with spaces removed as the family name, and the second declaration uses this name with `-base` tacked onto the

end. However the font gets loaded, we store the NFSS family names in `\M@font@me` and `\M@font@me@base`.

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 previous section.

```
492 \def\M@split@colon#1:#2\@nil{\def\@tempbase{#1}
493   \def\@tempfeatures{#2}}
494 \def\M@strip@colon#1:{#1}
```

The macro `\M@fill@nfss@shapes` accepts two arguments and does the actual work of ensuring that the NFSS contains the appropriate series and shapes. The first argument should be the name of a font family in the NFSS, and the second should be a list of OpenType features. We check whether combinations of bold series and italic shape exist for that font in the NFSS, and if not, we add them with `\DeclareFontShape`.

```
495 \def\M@fill@nfss@shapes#1#2{%
```

Upright shape.

```
496   \ifcsname TU/#1/\mddefault/\shapedefault\endcsname
497   \else
498     \DeclareFontShape{TU}{#1}{\mddefault}{\shapedefault}
499     {<->"\@tempbase:\M@otf@features;#2"}{}
500   \fi
```

Italic shape.

```
501   \ifcsname TU/#1/\mddefault/\itdefault\endcsname
502   \else
503     \DeclareFontShape{TU}{#1}{\mddefault}{\itdefault}
504     {<->"\@tempbase/I:\M@otf@features;#2"}{}
505   \fi
```

Bold series with upright shape.

```
506   \ifcsname TU/#1/\bfdefault/\shapedefault\endcsname
507   \else
508     \DeclareFontShape{TU}{#1}{\bfdefault}{\shapedefault}
509     {<->"\@tempbase/B:\M@otf@features;#2"}{}
510   \fi
```

Bold series with italic shape.

```
511   \ifcsname TU/#1/\bfdefault/\itdefault\endcsname
512   \else
513     \DeclareFontShape{TU}{#1}{\bfdefault}{\itdefault}
514     {<->"\@tempbase/BI:\M@otf@features;#2"}{}
515   \fi
```

Now do the same thing for the small caps variants. I make no promises that this will work. 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.


```

516 \ifcsname TU/#1/\mddefault/\scdefault\endcsname
517 \else
518   \DeclareFontShape{TU}{#1}{\mddefault}{\scdefault}
519     {<->"\@tempbase:\M@otf@features@sc;#2"}{}
520 \fi
521 \ifcsname TU/#1/\mddefault/\scdefault\itdefault\endcsname
522 \else
523   \DeclareFontShape{TU}{#1}{\mddefault}{\scdefault\itdefault}
524     {<->"\@tempbase/I:\M@otf@features@sc;#2"}{}
525 \fi
526 \ifcsname TU/#1/\bfdefault/\scdefault\endcsname
527 \else
528   \DeclareFontShape{TU}{#1}{\bfdefault}{\scdefault}
529     {<->"\@tempbase/B:\M@otf@features@sc;#2"}{}
530 \fi
531 \ifcsname TU/#1/\bfdefault/\scdefault\itdefault\endcsname
532 \else
533   \DeclareFontShape{TU}{#1}{\bfdefault}{\scdefault\itdefault}
534     {<->"\@tempbase/BI:\M@otf@features@sc;#2"}{}
535 \fi}

```

The main font-loading macro. This macro takes a single argument, which should have the form $\langle font\ name \rangle : \langle optional\ features \rangle$, and `mathfont` handles the information in one of three ways if all goes well: interface with `fontspec`, possibly declare a few extra shapes for a font already in the NFSS, or declare and load the whole font. At a minimum, `mathfont` ensures that we have access to medium upright, medium italic, bold upright, and bold italic fonts after calling `\M@newfont`. If `mathfont` decides to declare a font itself, it will also try to load small caps versions. We begin by splitting the argument into `\@tempbase` and `\@tempfeatures`.

```

536 \def\M@newfont#1{%
537   \edef\@tempa{#1}
538   \expandafter\M@split@colon\@tempa:\@nil
539   \def\@tempb{fontspec}

```

If the argument is “fontspec,” we want to use the last font loaded by `fontspec`, which is stored in `\l_fontspeg_family_tl`. If this macro is not empty, we store its contents in `\M@f@ntn@me` and skip loading entirely because `fontspec` already took care of it. We issue an error if `\l_fontspeg_family_tl` is empty or if the user has not loaded `fontspec`. If we use `fontspec` to load the font, we don’t get a separate font in base mode.

```

540 \ifx\@tempa\@tempb
541   \ifpackageloaded{fontspec}{%
542     \expandafter\ifx\csname l_fontspeg_family_tl\endcsname\@empty
543       \M@NoFontspecFamilyError
544     \else
545       \expandafter
546         \let\expandafter\M@f@ntn@me\csname l_fontspeg_family_tl\endcsname
547         \def\@tempbase{\M@f@ntn@me\space(from fontspec)}
548         \let\M@f@ntn@meb@se\M@f@ntn@me % no separate font in base mode
549       \fi}{\M@NoFontspecError}

```

If the argument is something other than “fontspec,” we need to parse it. If the user loaded `fontspec`, we pass the entire argument to `\fontspec_set_family:Nnn` for loading and store the NFSS family name in `\M@f@ntn@me`. For Lua_T_E_X, this is not recommended—`fontspec` is designed to work with text, not math, fonts and typically loads fonts in node mode, which makes their OpenType features unusable in math mode.

```

550 \else
551   \@ifpackageloaded{fontspec}
552   {\csname fontspec_set_family:Nnn\endcsname\M@f@ntn@me}{\@tempa}
553   % no separate font in base mode
554   \let\M@f@ntn@meb@se\M@f@ntn@me}

```

If the user has not loaded `fontspec`, we split the argument into a name and features using `\M@split@colon`. The name goes in `\@tempbase`, and the features go in `\@tempfeatures`. We store the OpenType features for loading in base mode inside `\@basefeatures`. If we are typesetting in Lua_T_E_X, `\@basefeatures` will be the same as `\@tempfeatures` except with `mode=base` at the end, and if we are using X_Y_T_E_X, it will be exactly the same.

```

555   {\M@Decl@reF@milytrue
556     \M@Decl@reF@milyB@settrue
557     \ifx\@tempfeatures\@empty
558       \ifdefined\directlua
559         \edef\@basefeatures{mode=base}
560       \else
561         \let\@basefeatures\@tempfeatures
562       \fi
563     \else
564       \edef\@tempfeatures{\expandafter\M@strip@colon\@tempfeatures}
565       \ifdefined\directlua
566         \edef\@basefeatures{\@tempfeatures;mode=base}
567       \else
568         \let\@basefeatures\@tempfeatures
569       \fi
570     \fi

```

We remove the spaces from `#1` and store it in `\@tempa` and from the human-readable font name contained in `#1` and store it in `\@tempb`. We check whether either already exists as a family name in the NFSS, and if we do, we call `\M@fill@nfss@shapes` to ensure that we have declared all the shapes. In this case, we set `\ifM@Decl@reF@mily` to false and break out of the `\@tfor` loop.

```

571     \edef@nospace\@tempa{\@tempa}
572     \edef@nospace\@tempb{\@tempbase}
573     \M@Decl@reF@milytrue
574     \@tfor\@i:=\@tempa\@tempb\@tempbase\do{%
575       \ifcsname TU+\@i\endcsname
576         \expandafter\let\expandafter\M@f@ntn@me\@i
577         \M@Decl@reF@milyfalse
578         \M@fill@nfss@shapes\M@f@ntn@me\@tempfeatures
579       \@break@tfor
580     \fi}

```

If `\M@newfont` didn't find anything in the NFSS, we need to load the font. The name for the font family will be #1 with spaces removed, which we previously stored in `\@tempa`.

```
581     \ifM@Decl@reF@amily
582     \let\M@f@ntn@me\@tempa
583     \wlog{Package mathfont Info: Adding \M@f@ntn@me\space to the nfss!}
584     \DeclareFontFamily{TU}{\M@f@ntn@me}{}
```

Now load the four most common font faces with `\M@fill@nfss@shapes`.

```
585     \M@fill@nfss@shapes\M@f@ntn@me\@tempfeatures
586     \fi
```

At this point, there is an entry for the font in the NFSS, and we stored the family name in `\M@f@ntn@me`. Now we check if the NFSS contains a base-mode version with the family name ending in `-base`.

```
587     \ifdefined\directlua
588     \edef\M@f@ntn@meb@se{\M@f@ntn@me-base}
589     \else
590     \let\M@f@ntn@meb@se\M@f@ntn@me
591     \fi
592     \ifcsname TU+\M@f@ntn@meb@se\endcsname\else
593     \wlog{Package mathfont Info: Adding \M@f@ntn@meb@se\space
594     to the nfss!}
595     \DeclareFontFamily{TU}{\M@f@ntn@meb@se}{ }
596     \fi
597     \M@fill@nfss@shapes\M@f@ntn@meb@se\@basefeatures}
598 \fi}
```

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

```
599 \@onlypreamble\M@fill@nfss@shapes
600 \@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 played with the idea of forcing some fonts to load now, but I'm hesitant to change L^AT_EX's standard font-loading behavior. I may address this issue further in future versions of `mathfont`.

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.

```

601 \def\M@check@option@valid#1{%
602   \let\@temperror\M@InvalidOptionError % error by default
603   \@for\@j:=\M@keys\do{%
604     \ifx\@j#1
605       \let\@temperror\@gobble % eliminate error
606       \@optionpresenttrue      % set switch to true
607     \fi}
608 \def\@j{empty} % if option is "empty," we do nothing
609 \ifx\@j#1
610   \let\@temperror\@gobble
611   \@optionpresentfalse
612 \fi
613 \@temperror{#1}}

```

Do the same thing for the suboption.

```

614 \def\M@check@suboption@valid#1{%
615   \let\@temperror\M@InvalidSuboptionError % error by default
616   \@for\@j:=roman,upright,italic\do{%
617     \ifx\@j#1
618       \let\@temperror\@gobble % eliminate error
619       \@suboptionpresenttrue  % set switch to true
620     \fi}
621   \@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@b` contains `<suboption>=`, and we use `\M@strip@equals` to get rid of the extra `=`. If the user does not specify a suboption, `\@temp@b` will be empty.

```

622 \def\M@strip@equals#1={#1}
623 \def\M@parse@option#1=#2\@nil{%
624   \@optionpresentfalse      % set switch to false by default
625   \@suboptionpresentfalse  % set switch to false by default
626   \def\@temp@opt{#1}        % store option
627   \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@a` is empty, meaning the user did not specify an option; or (2) `\@temp@b` is `=`, meaning the user typed `=` but did not follow it with a suboption.

```

628   \ifx\@temp@opt\@empty
629     \M@MissingOptionError
630   \else

```

Check that the user specified a valid option. We redefine `\@temp@a` 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`.

```

631 \M@check@option@valid\@temp@opt
632 \bgroup\def\@tempa{=}
633 \ifx\@temp@sub\@tempa
634 \egroup % first branch \egroup
635 \M@MissingSuboptionError
636 \else
637 \egroup % second branch \egroup

```

If `\@temp@sub` is nonempty, strip the final = and check that it contains a valid suboption.

```

638 \ifx\@temp@sub\@empty
639 \else
640 \edef\@temp@sub{\expandafter\M@strip@equals\@temp@sub}
641 \M@check@suboption@valid\@temp@sub % check that suboption is valid
642 \fi
643 \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.

```

644 \bgroup\def\@tempa{roman}
645 \ifx\@temp@sub\@tempa
646 \egroup % first branch \egroup
647 \def\@temp@sub{upright}
648 \else
649 \egroup % second branch \egroup
650 \fi
651 \fi}

```

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

```

652 \long\def\edef@nospace#1#2{%
653 \edef#1{#2}%
654 \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`.

```

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

```

The internal font-changing command.

```

656 \def\@mathfont[#1]#2{%

```

```
657 \ifx\set@mathchar@@set@mathchar
658   \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.

```
659 \else
660   \M@toks{}
```

We immediately call `\M@newfont` on the mandatory argument of `\mathfont`. We store the NFSS family name in `\M@fontfamily@<argument>` and `M@fontfamily@base@<argument>`. 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.

```
661   \M@newfont{#2}
662   \expandafter\edef\csname M@fontfamily@#2\endcsname{\M@fontn@me}
663   \expandafter\edef\csname M@fontfamily@base@#2\endcsname{\M@fontn@meb@se}
664   \ifcsname M@fontid@\M@fontn@me\endcsname\else % need new \M@count value?
665     \expandafter\edef\csname M@fontid@\M@fontn@me\endcsname{\the\M@count}
666     \expandafter\let\csname M@fontid@\M@fontn@meb@se\expandafter\endcsname
667       \csname M@fontid@\M@fontn@me\endcsname
668     \advance\M@count\@ne
669   \fi
670   \edef\@tempa{\csname M@fontid@\M@fontn@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.

```
671   \edef@nospace\@tempb{#1}
672   \@for\@i:=\@tempb\do{\expandafter\M@parse@option\@i=\@nil}
673   \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!)

```
674     \expandafter\ifx % next line is two cs to be compared
675       \csname ifM@\@temp@opt\expandafter\endcsname\csname iftrue\endcsname
676       \M@CharsSetWarning{\@temp@opt}
677     \else
```

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

```
678       \edef\@tempc{\the\M@toks^^J\@temp@opt}
679       \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`.

```

680     \if@suboptionpresent
681         \expandafter\edef\csname M@\@temp@opt shape\endcsname{\@temp@sub}
682     \fi
683     \edef\@tempc{\the\M@toks\space
684         (\csname M@\@temp@opt shape\endcsname)}
685     \M@toks\expandafter{\@tempc}
686     \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.

```

687     \def\@tempb{upright}
688     \ifx\@tempb\@tempc
689         \let\@tempb\shapedefault
690     \else
691         \let\@tempb\itdefault
692     \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.

```

693     \ifcsname symM@\@tempc\@tempa\endcsname\else
694         \M@SymbolFontInfo{\@tempbase}{\M@fontname@se}
695         {\mddefault/\@tempb}{M@\@tempc\@tempa}
696     \DeclareSymbolFont
697         {M@\@tempc\@tempa}{TU}{\M@fontname@se}{\mddefault}{\@tempb}
698     \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.

```

699     \expandafter
700         \edef\csname M@\@temp@opt @fontinfo\endcsname{\@tempbase}
701     \M@FontChangeInfo{\@temp@opt}{\@tempbase}

```

And now the magic happens!

```

702     \csname M@\@temp@opt @set\endcsname % set default font
703     \csname M@\@temp@opt true\endcsname % set switch to true
704     \fi
705     \fi}

```

Display concluding messages for the user.

```

706     \edef\@tempa{\the\M@toks}
707     \ifx\@tempa\@empty

```

```

708     \wlog{The \string\mathfont\space command on line
709     \the\inputlineno\space did not change the font for any characters!}
710   \else
711     \wlog{}
712     \typeout{:: mathfont :: Using font \@tempbase\space
713     on line \the\inputlineno.}
714     \wlog{Character classes changed:\the\M@toks}
715   \fi
716 \fi}
717 \@onlypreamble\mathfont
718 \@onlypreamble\m@thfont
719 \@onlypreamble\@mathfont

```

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

```

720 \protected\def\setfont#1{%
721   \mathfont{#1}
722   \mathconstantsfont{#1}
723   \setmathfontcommands{#1}
724   \let\rmdefault\M@font@me}
725 \@onlypreamble\setfont

```

The macro `\mathconstantsfont` handles choosing the font for setting math parameters in Lua \TeX . It issues a warning if called in X \TeX . 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.

```

726 \ifdefined\directlua
727   \let\M@SetMathConstants\relax
728   \protected\def\mathconstantsfont{%
729     \ifnextchar[{\@mathconstantsfont}{\@mathconstantsfont[upright]}}
730   \def\@mathconstantsfont[#1]#2{%
731     \edef\@tempa{\csname M@fontfamily@base@#2\endcsname}
732     \expandafter\ifx\@tempa\relax
733       \M@BadMathConstantsFontError{#2}
734     \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`.

```

735     \def\@tempb{#1}
736     \def\@tempc{upright}
737     \ifx\@tempb\@tempc
738       \let\m@th@const@nts@font@sh@pe\shapedefault
739     \else
740       \def\@tempc{italic}
741       \ifx\@tempb\@tempc
742         \let\m@th@const@nts@font@sh@pe\itdefault
743       \else
744         \M@BadMathConstantsFontTypeError{#1}
745     \fi

```



```

746     \fi
747     \ifcsname symM#1\csname M@fontid@\@tempa\endcsname\endcsname\else
748         \DeclareSymbolFont{M#1\csname M@fontid@\@tempa\endcsname}
749             {TU}{\@tempa}{\mddefault}{\m@th@const@nts@font@sh@pe}
750     \fi
751     \let\m@th@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^AT_EX 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^AT_EX 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`.

```

752     \protected\def\M@SetMathConstants{%
753         \begingroup
754         \escapechar\m@ne
755         \expandafter\getanddefine@fonts
756             \csname symM#1\csname M@fontid@\m@th@const@nts@font\endcsname
757             \expandafter
758                 \endcsname % expands to \symMupright<id>
759                 \csname TU/\m@th@const@nts@font
760                     /\seriesdefault
761                     /\m@th@const@nts@font@sh@pe
762                 \endcsname % expands to \TU/<nfss family name>/m/<shape>
763         \globaldefs\@ne
764         \expandafter\@gobbletwo\math@fonts % gobble to avoid infinite loop
765         \endgroup}
766     \fi}
767 \def\math@fonts{\aftergroup\M@SetMathConstants}
768 \else
769 \protected\def\mathconstantsfont{\M@LuaTeXOnlyWarning
770     \@ifnextchar[\@gobbletwo@brackets\@gobble}
771 \fi
772 \@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.

```

773 \def\M@check@csarg#1#2{%
774   \expandafter\ifx\expandafter\@nnil\@gobble#2\@nnil % good
775   \ifcat\relax\noexpand#2 % good
776     \M@arg@goodtrue
777   \else % if #2 not a control sequence
778     \M@MissingControlSequenceError#1{#2}
779     \M@arg@goodfalse
780   \fi
781 \else % if #2 is multiple tokens
782   \M@DoubleArgError#1{#2}
783   \M@arg@goodfalse
784 \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`.

```

785 \protected\def\newmathfontcommand#1#2#3#4{%
786   \M@check@csarg\newmathfontcommand{#1}
787   \ifM@arg@good
788     \M@newfont{#2}
789     \M@NewFontCommandInfo{#1}{\@tempbase}{\M@f@ntn@m@b@se}{#3}{#4}
790     \DeclareMathAlphabet{#1}{TU}{\M@f@ntn@m@b@se}{#3}{#4}
791   \fi}
792 \@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`.

```

793 \def\M@define@newmath@cmd#1#2#3{%
794   \protected\def#1##1##2{%
795     \M@check@csarg{#1}{##1}
796     \newmathfontcommand{##1}{##2}{#2}{#3}}
797 \def\M@default@newmath@cmds{%
798   \newmathrm{\mddefault}{\shapedefault},%
799   \newmathit{\mddefault}{\itdefault},%
800   \newmathbf{\bfdefault}{\shapedefault},%
801   \newmathbfit{\bfdefault}{\itdefault},%
802   \newmathsc{\mddefault}{\scdefault},%
803   \newmathscit{\mddefault}{\scdefault\itdefault},%
804   \newmathbfsc{\bfdefault}{\scdefault},%
805   \newmathbfscit{\bfdefault}{\scdefault\itdefault}}
806 \@for\@i:=\M@default@newmath@cmds\do{\expandafter\M@define@newmath@cmd\@i}
807 \@onlypreamble\newmathrm
808 \@onlypreamble\newmathit
809 \@onlypreamble\newmathbf
810 \@onlypreamble\newmathbfit
811 \@onlypreamble\newmathsc
812 \@onlypreamble\newmathscit
813 \@onlypreamble\newmathbfsc
814 \@onlypreamble\newmathbfscit
815 \@onlypreamble\M@define@newmath@cmd
816 \let\M@default@newmath@cmds\relax

```

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

```

817 \protected\def\setmathfontcommands#1{%
818   \newmathrm\mathrm{#1}
819   \newmathit\mathit{#1}
820   \newmathbf\mathbf{#1}
821   \newmathbfit\mathbfit{#1}
822   \newmathsc\mathsc{#1}
823   \newmathscit\mathscit{#1}
824   \newmathbfsc\mathbfsc{#1}
825   \newmathbfscit\mathbfscit{#1}}
826 \@onlypreamble\setmathfontcommands

```

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

```

827 \protected\def\newmathbold{%
828   \M@DeprecatedWarning\newmathbold\newmathbf\newmathbf}
829 \protected\def\newmathboldit{%
830   \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.

```

831 \ifM@adjust@font
832   \def\M@check@int#1{%
833     \M@arg@goodfalse
834     \begingroup
835     \edef\@tempa{\number0#1}
836     \edef\@tempa{\detokenize\expandafter{\@tempa}}
837     \@expandtwoargs\in@{"}\@tempa}

```

If `#1` contains a " or backslash, we set `\M@arg@good` to false and stop parsing the argument.

```

838   \ifin@ % is " in #1?
839     \endgroup % first branch \endgroup
840   \else
841     \@expandtwoargs\in@{\@backslashchar}\@tempa}
842     \ifin@ % is backslash in #1?
843       \endgroup % second branch \endgroup
844     \else
845       \directlua{
846         local num = tonumber("\@tempa")
847         local bool = 0 % keep track if \@tempa is (int >= 0)
848         if num then % if number?
849           if num == num - (num \@percentchar 1) then % if integer?
850             if num >= 0 then % if nonnegative?
851               bool = 1
852             end
853           end
854         end
855         tex.print("\@backslashchar\@backslashchar endgroup")
856         if bool == 1 then
857           tex.print("\@backslashchar\@backslashchar csname M@arg@goodtrue%
858             \@backslashchar\@backslashchar endcsname")
859         end}
860     \fi
861 \fi}

```

Define `\RuleThicknessFactor`.

```

862 \protected\def\RuleThicknessFactor#1{%
863   \M@check@int{#1}
864   \ifM@arg@good
865     \global\M@rule@thickness@factor=#1\relax
866   \else
867     \M@BadIntegerError\RuleThicknessFactor{#1}
868   \fi}

```

Define `\IntegralItalicFactor`.

```
869 \protected\def\IntegralItalicFactor#1{%
870   \M@check@int{#1}
871   \ifM@arg@good
872     \global\M@integral@italic@factor=#1\relax
873   \else
874     \M@BadIntegerError\IntegralItalicFactor{#1}
875   \fi}
```

Define `\SurdHorizontalFactor`.

```
876 \protected\def\SurdHorizontalFactor#1{%
877   \M@check@int{#1}
878   \ifM@arg@good
879     \global\M@surd@horizontal@factor=#1\relax
880   \else
881     \M@BadIntegerError\SurdHorizontalFactor{#1}
882   \fi}
```

Define `\SurdVerticalFactor`.

```
883 \protected\def\SurdVerticalFactor#1{%
884   \M@check@int{#1}
885   \ifM@arg@good
886     \global\M@surd@vertical@factor=#1\relax
887   \else
888     \M@BadIntegerError\SurdVerticalFactor{#1}
889   \fi}
```

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.

```
890 \else
891   \@tfor\@i:=\RuleThicknessFactor\IntegralItalicFactor\SurdHorizontalFactor
892     \SurdVerticalFactor\CharmLine\CharmFile
893     \do{%
894       \protected\expandafter\edef\@i{\noexpand\M@NoFontAdjustError
895         \expandafter\noexpand\@i
896         \noexpand\@gobble}}
897 \fi}
```

These commands should appear in the preamble only.

```
898 \@onlypreamble\RuleThicknessFactor
899 \@onlypreamble\IntegralItalicFactor
900 \@onlypreamble\SurdHorizontalFactor
901 \@onlypreamble\SurdVerticalFactor
902 \@onlypreamble\CharmLine
903 \@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.

```
904 \def\restoremathinternals{%
```

```

905 \ifx\set@mathchar\@@set@mathchar
906 \else
907   \wlog{Package mathfont Info: Restoring \string\set@mathchar.}
908   \wlog{Package mathfont Info: Restoring \string\set@mathsymbol.}
909   \wlog{Package mathfont Info: Restoring \string\set@mathaccent.}
910   \wlog{Package mathfont Info: Restoring \string\DeclareSymbolFont.}
911   \let\set@mathchar\@@set@mathchar
912   \let\set@mathsymbol\@@set@mathsymbol
913   \let\set@mathaccent\@@set@mathaccent
914   \let\DeclareSymbolFont\@@DeclareSymbolFont
915 \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_T_E_X—perhaps in a future version.

```

916 \protected\gdef\clap#1{\hb@xt@\z@{\hss#1\hss}}
917 \protected\def\stack@flatrel#1#2{\expandafter
918   \st@ck@fl@trel\expandafter#1\@firstofone#2}
919 \protected\gdef\st@ck@fl@trel#1#2#3{%
920   {\setbox0\hbox{\m@th#1#2$}% contains \mathrel symbol
921   \setbox1\hbox{\m@th#1#3$}% gets raised over \box0
922   \if\wd0>\wd1\relax
923     \hb@xt@\wd0{%
924       \hfil
925       \clap{\raise0.7\ht0\box1}%
926       \clap{\box0}\hfil}%
927   \else
928     \hb@xt@\wd1{%
929       \hfil
930       \clap{\raise0.7\ht0\box1}%
931       \clap{\box0}\hfil}%
932 \fi}}

```

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

```

933 \ifnum\tracinglostchars<\tw@
934   \tracinglostchars\tw@
935 \fi

```

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

```

936 \ifdefined\XeTeXrevision
937   \ifM@font@loaded
938     \AtEndOfPackage{%
939       \PackageWarningNoLine{mathfont}
940       {XeTeX detected. It looks like you\MessageBreak
941       specified a font when you loaded\MessageBreak

```

```

942 mathfont. If you run into problems\MessageBreak
943 with a font whose name is multiple\MessageBreak
944 words, try compiling with LuaLaTeX\MessageBreak
945 or call \string\setfont\space or \string\mathfont\MessageBreak
946 manually}}
947 \fi
948 \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.

```

949 \def\keyword@info@begindocument#1:#2\@nil{%
950 \expandafter\ifx % next line is two cs to be compared
951     \csname ifM@#1\expandafter\endcsname\csname iftrue\endcsname
952     \wlog{#1:#2@spaces Set to
953     \csname M@#1@fontinfo\endcsname,
954     \csname M@#1shape\endcsname\space shape.}
955 \else
956     \wlog{#1:#2@spaces No change.}
957 \fi}

```

Now print the messages.

```

958 \AtBeginDocument{%
959 \def\@tempa{% <-- everything should be 14 characters long
960 upper:\@spaces\@spaces,%
961 lower:\@spaces\@spaces,%
962 diacritics:\space\space\space,%
963 greekupper:\space\space\space,%
964 greeklower:\space\space\space,%
965 agreeekupper:\space\space,%
966 agreeeklower:\space\space,%
967 cyrillicupper:,%
968 cyrilliclower:,%
969 hebrew:\@spaces\space\space\space,%
970 digits:\@spaces\space\space\space,%
971 operator:\@spaces\space,%
972 delimiters:\space\space\space,%
973 radical:\@spaces\space\space,%
974 bigops:\@spaces\space\space\space,%
975 extbigops:\@spaces,%
976 symbols:\@spaces\space\space,%
977 extsymbols:\space\space\space,%
978 arrows:\@spaces\space\space\space,%
979 bb:\@spaces\@spaces\space\space\space,%
980 cal:\@spaces\@spaces\space\space,%
981 frak:\@spaces\@spaces\space,%
982 bcal:\@spaces\@spaces\space,%

```

```

983   bfrac:\@spaces\@spaces}
984   \wlog{^^JPackage mathfont Info: List of changes made in the preamble.}
985   \@for\@i:=\@tempa\do{%
986     \expandafter\keyword@info@begindocument\@i\@nil}
987   \wlog{}}

```

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

```

988 \ifM@font@loaded
989   \AtEndOfPackage{\setfont\M@font@load}
990 \fi

```

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

```

991 \@onlypreamble\M@upper@set
992 \@onlypreamble\M@lower@set
993 \@onlypreamble\M@diacritics@set
994 \@onlypreamble\M@greekupper@set
995 \@onlypreamble\M@greeklower@set
996 \@onlypreamble\M@agreekupper@set
997 \@onlypreamble\M@agreeklower@set
998 \@onlypreamble\M@cyrillicupper@set
999 \@onlypreamble\M@cyrilliclower@set
1000 \@onlypreamble\M@hebrew@set
1001 \@onlypreamble\M@digits@set
1002 \@onlypreamble\M@operator@set
1003 \@onlypreamble\M@symbols@set
1004 \@onlypreamble\M@extsymbols@set
1005 \@onlypreamble\M@delimiters@set
1006 \@onlypreamble\M@arrows@set
1007 \@onlypreamble\M@bigops@set
1008 \@onlypreamble\M@extbigops@set
1009 \@onlypreamble\M@bb@set
1010 \@onlypreamble\M@cal@set
1011 \@onlypreamble\M@frak@set
1012 \@onlypreamble\M@bcal@set
1013 \@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.

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

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 $\text{T}_{\text{E}}\text{X}$ 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 $\text{T}_{\text{E}}\text{X}$ math parameters based on the essential parameters of the font, 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 $\text{T}_{\text{E}}\text{X}$ 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

Table 3: Functions in mathfont

Function	Argument(s)	Used For
<code>new_type_a</code>	<code>index, next, data</code>	Add type <code>a</code> entry to <code>mathfont</code>
<code>new_type_e</code>	<code>index, smash, next, data</code>	Add type <code>e</code> entry to <code>mathfont</code>
<code>new_type_u</code>	<code>index, smash, next, data</code>	Add type <code>u</code> entry to <code>mathfont</code>
<code>add_to_charm</code>	string of new charm info	Add new charm into to <code>mathfont</code>
<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 <code>a</code>
<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 <code>e</code>
<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 <code>u</code>
<code>modify_e_base</code>	<code>index, offset</code>	Modify base glyph for type <code>e</code>
<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 <code>log</code> file
<code>math_constants</code>	fontdata	Creates a <code>MathConstants</code> table
<code>set_nomath_true</code>	fontdata	Set top-level font specs for <code>math</code>

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{}` 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.

```
1014 \ifM@adjust@font
1015 \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.

```

1016 \bgroup
1017  \catcode`\~=0
1018  ~catcode`~\=12
1019  ~@firstofone{
1020 ~egroup
1021 ~def~M@number@ssert{"\n%
1022  Package mathfont error: Nonnumeric charm value.\n\n%
1023  I'm having trouble with a character metric.\n%
1024  Your \\CharmLine or \\CharmFile contains \".temp_string.\".\n%
1025  which is not a number. Make sure that your\n%
1026  charm information is all integers, floats,\n%
1027  or asterisks separated by commas or spaces.\n"}
1028 ~def~M@index@ssert{"\n%
1029  Package mathfont error: Invalid unicode index.\n\n%
1030  The unicode index \".split_string[1].\" is invalid. Make sure\n%
1031  that the first number in your \\CharmLine and in each\n%
1032  line of your \\CharmFile is an integer between 0 and\n%
1033  1,114,111.\n"}
1034 ~def~M@entries@ssert{"\n%
1035  Package mathfont error: Charm values too short.\n\n%
1036  Your charm information for U+\".index.\" needs more\n%
1037  entries. Right now you have \".number_of_entries.\" entries, and\n%
1038  you need at least \".entries_needed.\". If you aren't sure what\n%
1039  to do, try adding asterisks to your \\CharmLine\n%
1040  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.

```

1041 \protected\def\CharmLine#1{%
1042  \begingroup
1043  \edef\@tempa{#1}
1044  \edef\@tempa{\detokenize\expandafter{\@tempa}}
1045  \@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`.

```

1046  \ifin@ % is " in #1?
1047    \ifM@fromCharmFile
1048      \M@ForbiddenCharmFile{"}
1049    \else
1050      \M@ForbiddenCharmLine{"}
1051    \fi
1052  \else
1053    \@expandtwoargs\in@{\@backslashchar}{\@tempa}

```

```

1054   \ifin@ % is backslash in #1?
1055   \ifM@fromCharmFile
1056     \M@ForbiddenCharmFile{\@backslashchar}
1057   \else
1058     \M@ForbiddenCharmLine{\@backslashchar}
1059   \fi
1060 \else

```

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

```

1061   \directlua{mathfont:add_to_charm("\@tempa")}
1062 \fi
1063 \fi
1064 \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.

```

1065 \protected\def\CharmFile#1{%
1066 \begingroup
1067 \M@fromCharmFiletrue
1068 \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.

```

1069 \def\@next{%
1070   \read\M@Charm to \@tempa
1071   \CharmLine\@tempa
1072   \ifeof\M@Charm\else % if file has more lines?
1073     \expandafter\@next
1074   \fi}

```

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

```

1075 \@next
1076 \immediate\closein\M@Charm
1077 \M@fromCharmFilefalse
1078 \endgroup}

```

This concludes the T_EX-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.

```

1079 \directlua{
1080 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 .

```

1081 function mathfont:new_type_a(index, next, data)
1082   self[index] = {}
1083   self[index].type = "a"
1084   self[index].next = next
1085   self[index].left_stretch = data[1] / 1000
1086   self[index].right_stretch = data[2] / 1000
1087   self[index].top_accent_stretch = data[3] / 1000
1088   self[index].bot_accent_stretch = data[4] / 1000
1089 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.

```

1090 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`.

```

1091   local v = \string# next
1092   self[index] = {}
1093   self[index].type = "e"
1094   self[index].smash = smash
1095   self[index].next = next
1096   self[index].total_variants = v
1097   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.

```

1098   for i = 1, v, 1 do
1099     self[index].data[i] = {}
1100     self[index].data[i].x = data[2*i-1] / 1000
1101     self[index].data[i].y = data[2*i] / 1000
1102   end
1103   self[index].top_accent_stretch = data[2*v+1] / 1000
1104   self[index].bot_accent_stretch = data[2*v+2] / 1000
1105 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.

```

1106 function mathfont:new_type_u(index, data)
1107   self[index] = {}

```

```

1108 self[index].type = "u"
1109 self[index].top_accent_stretch = data[1] / 1000
1110 self[index].bot_accent_stretch = data[2] / 1000
1111 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`.

```

1112 function mathfont.parse_charm(charm_input)
1113   local split_string = {}
1114   local charm_string = charm_input
1115   local temp_string = ""
1116   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.

```

1117   while string.find(charm_string, " ") or string.find(charm_string, ",") do
1118     local length = string.len(charm_string)
1119     local first_space = string.find(charm_string, " ") or length
1120     local first_comma = string.find(charm_string, ",") or length

```

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

```

1121     local sep = first_space
1122     if first_comma < first_space then
1123       sep = first_comma
1124     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`.

```

1125     temp_string = string.sub(charm_string, 1, sep-1)
1126     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.

```

1127     if temp_string \noexpand~= "" then
1128       if tonumber(temp_string) then % if a number, append number
1129         split_string[i] = tonumber(temp_string)
1130         i = i+1
1131       elseif temp_string == "*" then % if asterisk, append asterisk
1132         split_string[i] = temp_string
1133         i = i+1
1134       else % if neither, raise error
1135         error(\M@number@assert)
1136     end

```

```

1137     end
1138 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.

```

1139 temp_string = charm_string
1140 if temp_string \noexpand~="" then
1141     if tonumber(temp_string) then % if a number, append number
1142         split_string[i] = tonumber(temp_string)
1143     elseif temp_string == "*" then % if asterisk, append asterisk
1144         split_string[i] = temp_string
1145     else % if neither, raise error
1146         error(\M@number@assert)
1147     end
1148 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.

```

1149 local index = split_string[1]
1150 if index == "*" then
1151     error(\M@index@assert)
1152 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`.

```

1153 local rounded = index - (index \@percentchar 1) % subtract decimal portion
1154 local max = 1114111
1155 assert(index == rounded and index >= 0 and index <= max, \M@index@assert)
1156 return split_string
1157 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`.

```

1158 function mathfont:add_to_charm(charm_string)
1159     local charm_metrics = self.parse_charm(charm_string)
1160     local index = charm_metrics[1]
1161     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`.

```

1162     if not self[index] then
1163         self:new_type_u(index, {0, 0})
1164     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.

```

1165  if self[index].type == "a" then
1166    local entries_needed = 5
1167    assert(number_of_entries >= entries_needed, \M@entries@assert)
1168    if charm_metrics[2] \noexpand~="*" then
1169      self[index].left_stretch = charm_metrics[2] / 1000
1170    end
1171    if charm_metrics[3] \noexpand~="*" then
1172      self[index].right_stretch = charm_metrics[3] / 1000
1173    end
1174    if charm_metrics[4] \noexpand~="*" then
1175      self[index].top_accent_stretch = charm_metrics[4] / 1000
1176    end
1177    if charm_metrics[5] \noexpand~="*" then
1178      self[index].bot_accent_stretch = charm_metrics[5] / 1000
1179    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.

```

1180  elseif self[index].type == "e" then
1181    local tot_variants = self[index].total_variants
1182    local entries_needed = 2 * tot_variants + 3
1183    assert(number_of_entries >= entries_needed, \M@entries@assert)
1184    for i = 1, tot_variants, 1 do
1185      if charm_metrics[2*i] \noexpand~="*" then
1186        self[index].data[i].x = charm_metrics[2*i] / 1000
1187      end
1188      if charm_metrics[2*i+1] \noexpand~="*" then
1189        self[index].data[i].y = charm_metrics[2*i+1] / 1000
1190      end
1191    end

```

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

```

1192    if charm_metrics[2*tot_variants+2] \noexpand~="*" then
1193      self[index].top_accent_stretch = charm_metrics[2*tot_variants+2] / 1000
1194    end
1195    if charm_metrics[2*tot_variants+3] \noexpand~="*" then
1196      self[index].bot_accent_stretch = charm_metrics[2*tot_variants+3] / 1000
1197    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.


```

1198 elseif self[index].type == "u" then
1199     local entries_needed = 3
1200     assert(number_of_entries >= entries_needed, \M@entries@assert)
1201     if charm_metrics[2] \noexpand~="*" then
1202         self[index].top_accent_stretch = charm_metrics[2] / 1000
1203     end
1204     if charm_metrics[3] \noexpand~="*" then
1205         self[index].bot_accent_stretch = charm_metrics[3] / 1000
1206     end
1207 end
1208 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.

```

1209 function mathfont.make_hex_value(integer)
1210     if integer == 0 then
1211         return "0000"
1212     end
1213     if integer == 1 then
1214         return "0001"
1215     end
1216     local hex_digits = "0123456789ABCDEF" % for reference
1217     local hex_string = ""
1218     local curr_val = integer
1219     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`.

```

1220     local i = 0
1221     while 16i <= curr_val do
1222         i = i+1
1223     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`.

```

1224     for j = i-1, 0, -1 do
1225         remainder = curr_val \@percentchar (16j)
1226         curr_val = (curr_val - remainder) / (16j)
1227         hex_string = hex_string .. string.sub(hex_digits, curr_val+1, curr_val+1)
1228         curr_val = remainder
1229     end

```

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

```

1230     if \string# hex_string < 4 then

```

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

```

1231   for i = \string# hex_string, 4, 1 do
1232     hex_string = "0" .. hex_string
1233   end
1234 end
1235 return hex_string
1236 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.

```

1237 function mathfont.glyph_info(char)
1238   local glyph_width = char.width or 0
1239   local glyph_height = char.height or 0
1240   local glyph_depth = char.depth or 0
1241   local glyph_italic = char.italic or 0
1242   return glyph_width, glyph_height, glyph_depth, glyph_italic
1243 end

```

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.

```

1244 function mathfont:smash_glyph(index, fontdata)
1245   local smash_table = {}
1246   smash_table.width = 0
1247   smash_table.height = 0
1248   smash_table.depth = 0
1249   smash_table.commands = {"char", index}
1250   return smash_table
1251 end

```

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

```

1252 function mathfont.empty(arg)
1253 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.

```
1254 function mathfont.make_a_commands(index, offset)
1255   local c_1 = {"right", offset}
1256   local c_2 = {"char", index}
1257   return {c_1, c_2}
1258 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.

```
1259 function mathfont:make_a_table(index, charm_data, fontdata)
1260   local a_table = {}
1261   local char = fontdata.characters[index] or {}
1262   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.

```
1263   local left_stretch = charm_data.left_stretch
1264   local right_stretch = charm_data.right_stretch
```

```
1265 local width, height, depth, italic = self.glyph_info(char)
```

Incorporate the italic correction into the character width.

```
1266 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.

```
1267 local offset = width * left_stretch
```

```
1268 a_table.width = width * (1 + left_stretch + right_stretch)
```

```
1269 a_table.height = height
```

```
1270 a_table.depth = depth
```

```
1271 a_table.italic = italic
```

```
1272 a_table.unicode = index
```

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

```
1273 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 σ_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.

```
1274 local top_base = (0.5 + left_stretch) * width + 0.5 * slant * height
```

```
1275 local bot_base = (0.5 + left_stretch) * width - 0.5 * slant * height
```

```
1276 local top_accent_shift = charm_data.top_accent_stretch * width
```

```
1277 local bot_accent_shift = charm_data.bot_accent_stretch * width
```

```
1278 a_table.top_accent = top_base + top_accent_shift
```

```
1279 a_table.bot_accent = bot_base + bot_accent_shift
```

Add the commands to the table.

```
1280 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.

```
1281 a_table.mathkern = {}
1282 a_table.mathkern.top_right = {{height = 0, kern = -italic}}
1283 a_table.mathkern.bottom_right = {{height = 0, kern = -italic}}
1284 a_table.mathkern.top_left = {{height = 0, kern = 0}}
1285 a_table.mathkern.bottom_left = {{height = 0, kern = 0}}
1286 return a_table
1287 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.

```
1288 function mathfont:modify_e_base(index, fontdata)
1289   local char = fontdata.characters[index] or {}
1290   local width, height, depth, italic = self.glyph_info(char)
1291   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@surd@horizontal@factor` and `\M@surd@vertical@factor`.

```
1292 if index == 8730 then
```

Now get the scale factors from the TeX side of things and scale down (or up) the height and width of the surd.

```
1293   local horizontal_scale = tex.getcount("M@surd@horizontal@factor") / 1000
1294   local vertical_scale = tex.getcount("M@surd@vertical@factor") / 1000
1295   char.width = horizontal_scale * char.width
1296   char.height = vertical_scale * height
1297 end
```

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

```
1298 if index == 8747 then
1299   local scale_factor = tex.getcount("M@integral@italic@factor") / 1000
1300   char.italic = scale_factor * width
1301 end
1302 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.

```

1303 function mathfont.make_e_commands(index, h_stretch, v_stretch)
1304   local c_1 = {"pdf", "origin", string.format(
1305     "q \@percentchar s 0 0 \@percentchar s 0 0 cm", h_stretch, v_stretch)}
1306   local c_2 = {"char", index}
1307   local c_3 = {"pdf", "origin", "Q"}
1308   return {c_1, c_2, c_3}
1309 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.

```

1310 function mathfont:make_e_table(index, charm_data, fontdata)
1311   local e_table = {}
1312   local char = fontdata.characters[index] or {}
1313   local slant = fontdata.parameters.slant / 65536
1314   local tounicode = self.make_hex_value(index)
1315   local smash_index = charm_data.smash
1316   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.

```

1317   for i = 1, charm_data.total_variants, 1 do
1318     local h_stretch = charm_data.data[i].x
1319     local v_stretch = charm_data.data[i].y
1320     local new_width = width * h_stretch
1321     local new_height = height * v_stretch
1322     local new_depth = depth * v_stretch
1323     local new_italic = italic * h_stretch

```

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

```

1324     e_table[i] = {}
1325     e_table[i].width = new_width
1326     e_table[i].height = new_height
1327     e_table[i].depth = new_depth
1328     e_table[i].italic = new_italic

```

Add the unicode information.

```
1329     e_table[i].unicode = index
1330     e_table[i].tunicode = tunicode
```

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

```
1331     local base_top_accent = 0.5 * new_width + 0.5 * slant * new_height
1332     local base_bot_accent = 0.5 * new_width - 0.5 * slant * new_height
1333     local top_accent_shift = charm_data.top_accent_stretch * new_width
1334     local bot_accent_shift = charm_data.bot_accent_stretch * new_width
1335     e_table[i].top_accent = base_top_accent + top_accent_shift
1336     e_table[i].bot_accent = base_bot_accent + bot_accent_shift
```

Add the commands.

```
1337     e_table[i].commands =
1338         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* + 1st character, and we can extract the index from the `charm_information`.

```
1339     if i < charm_data.total_variants then
1340         e_table[i].next = charm_data.next[i+1]
1341     end
1342 end
1343 return e_table
1344 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.

```
1345 function mathfont:make_u_table(index, charm_data, fontdata)
1346     local u_table = fontdata.characters[index] or {}
1347     local slant = fontdata.parameters.slant / 65536 or 0
1348     local width, height, depth, italic = self.glyph_info(u_table)
1349     local new_width = width + italic
1350     u_table.width = new_width
```

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

```
1351     local base_top_accent = 0.5 * new_width + 0.5 * slant * height
1352     local base_bot_accent = 0.5 * new_width - 0.5 * slant * height
1353     local top_accent_shift = charm_data.top_accent_stretch * new_width
1354     local bot_accent_shift = charm_data.bot_accent_stretch * new_width
1355     u_table.top_accent = base_top_accent + top_accent_shift
1356     u_table.bot_accent = base_bot_accent + bot_accent_shift
```

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

```
1357     u_table.mathkern = {}
1358     u_table.mathkern.top_right = {{height = 0, kern = -italic}}
1359     u_table.mathkern.bottom_right = {{height = 0, kern = -italic}}
1360     u_table.mathkern.top_left = {{height = 0, kern = 0}}
```

```

1361 u_table.mathkern.bottom_left = {{height = 0, kern = 0}}
1362 return u_table
1363 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.

```

1364 function mathfont.make_fake_angle(index, smash, fontdata)
1365   local temp = {}
1366   local lparen = fontdata.characters[40] or {}
1367   local lparen_height = lparen.height or 0
1368   local lparen_depth = lparen.depth or 0
1369   local glyph = fontdata.characters[index] or {}
1370   local glyph_height = glyph.height or 0
1371   local base_height = 0.9 * glyph_height
1372   local factor = 0
1373   if glyph_height \noexpand~= 0 then
1374     factor = (lparen_height + lparen_depth) / base_height
1375   end
1376   local shift = 0.1 * glyph_height * factor + lparen_depth
1377   temp.height = lparen_height
1378   temp.depth = lparen_depth
1379   temp.width = glyph.width or 0
1380   temp.italic = glyph.italic or 0
1381   temp.top_accent = glyph.top_accent or 0.5 * temp.width
1382   temp.bot_accent = glyph.bot_accent or 0.5 * temp.width
1383   temp.commands = {
1384     {"down", shift},
1385     {"pdf", "origin", string.format("q 1 0 0 \@percentchar s 0 0 cm", factor)},
1386     {"char", smash},
1387     {"pdf", "origin", "Q"},
1388     {"down", -shift}}
1389   return temp
1390 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.

```

1391 function mathfont.set_nomath_true(fontdata)
1392   fontdata.nomath = false
1393   fontdata.oldmath = false
1394 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.

```
1395 function mathfont.apply_charm_info(fontdata)
1396   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.

```
1397   chars[1044538] = mathfont.smash_glyph(8249, fontdata) % \lguil
1398   chars[1044539] = mathfont.smash_glyph(8250, fontdata) % \rguil
1399   chars[1044540] = mathfont.smash_glyph(171, fontdata) % \llguil
1400   chars[1044541] = mathfont.smash_glyph(187, fontdata) % \rrguil
```

Now add the characters to the font.

```
1401   chars[1044508] = mathfont.make_fake_angle(8249, 1044538, fontdata)
1402   chars[1044509] = mathfont.make_fake_angle(8250, 1044539, fontdata)
1403   chars[1044510] = mathfont.make_fake_angle(171, 1044540, fontdata)
1404   chars[1044511] = mathfont.make_fake_angle(187, 1044541, fontdata)
```

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

```
1405   if not chars[8711] then
1406     chars[8710] = chars[8710] or {}
1407     chars[1044508] = mathfont.smash_glyph(8710, fontdata)
1408     chars[8711] = {}
1409     chars[8711].width = chars[8710].width or 0
1410     chars[8711].height = chars[8710].height or 0
1411     chars[8711].depth = chars[8710].depth or 0
1412     chars[8711].italic = chars[8710].italic or 0
1413     chars[8711].top_accent = chars[8710].top_accent or 0.5 * chars[8711].width
1414     chars[8711].bot_accent = chars[8710].bot_accent or 0.5 * chars[8711].width
1415     chars[8711].unicode = 8711
1416     chars[8711].tounicode = mathfont.make_hex_value(8711)
1417     chars[8711].commands = {
1418       {"down", -chars[8711].height},
1419       {"pdf", "origin", "q 1 0 0 -1 0 0 cm"},
1420       {"char", 1044508},
1421       {"pdf", "origin", "Q"},
1422       {"down", chars[8711].height}}
1423   end
```

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

```
1424   for index, info in pairs(mathfont) do
1425     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.

```
1426       if info.type == "a" then
1427         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.

```
1428     elseif info.type == "e" then
1429         local smash = info.smash
1430         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.

```
1431         chars[index].next = info.next[1]
1432         mathfont:modify_e_base(index, fontdata)
1433         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`.

```
1434         local variants_table = mathfont:make_e_table(index, info, fontdata)
1435         for i = 1, info.total_variants, 1 do
1436             chars[info.next[i]] = variants_table[i]
1437         end
```

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

```
1438     elseif info.type == "u" then
1439         chars[index] = mathfont:make_u_table(index, info, fontdata)
1440     end
1441 end
1442 end
1443 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.

```
1444 function mathfont.math_constants(fontdata)
1445     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.

```
1446     local size = fontdata.size or 0
1447     local ex = fontdata.parameters.x_height or 0
1448     local em = fontdata.parameters.quad or 0
1449     local A_height = 0
1450     local y_depth = 0
1451     if fontdata.characters[65] then
```

```

1452   A_height = fontdata.characters[65].height or 0 % A
1453 end
1454 if fontdata.characters[121] then
1455   y_depth = fontdata.characters[121].depth or 0 % y
1456 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.

```

1457 local axis = 0
1458 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@rule@thickness@factor`, which is the value of that `\count` divided by 1000.

```

1459 local dim = "FractionRuleThickness"
1460 if not fontdata.MathConstants[dim] then
1461   local scale_factor = tex.getcount("M@rule@thickness@factor") / 1000
1462   rule_thickness = (size / 18) * scale_factor
1463   fontdata.MathConstants[dim] = rule_thickness
1464 else
1465   rule_thickness = fontdata.MathConstants[dim]
1466 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`.

```

1467 local dim = "AxisHeight"
1468 if fontdata.MathConstants[dim] then
1469   axis = fontdata.MathConstants[dim]
1470 else
1471   if fontdata.characters[45] then
1472     axis = fontdata.characters[45].height - 0.5 * rule_thickness
1473   else
1474     axis = 0.8 * ex
1475   end
1476   fontdata.MathConstants[dim] = axis
1477 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 ξ_{13} and the seventh script parameter σ_{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 ξ_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 ξ_9 away from the operator character, \TeX attempts to position the baseline of the limit at ξ_{10} distance above the operator character, and we set ξ_{10} to be slightly larger than ξ_9 . If the upper limit has no descender, \TeX will raise its baseline by ξ_{10} , and if it has a descender, \TeX will position the bottom of the descender to be ξ_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 ξ_{11} for the lower limit to be equal to the attempted clearance for the upper limit, and the attempted clearance ξ_{12} for the lower limit will be the minimum clearance plus the average of the `\scriptfont` x-height and `\scriptfont` A-height.

```

1478 local xi_9 = 2 * rule_thickness           % upper limit minimum clearance
1479 local xi_10 = xi_9 + 0.35 * y_depth      % upper limit attempt placement
1480 local xi_11 = xi_10                      % lower limit minimum clearance
1481 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 σ_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 * rule_thickness + 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.

```

1482 local sigma_8 = axis + 1.5 * rule_thickness + y_depth + 0.1 * A_height
1483 local sigma_9 = (axis + 1.35 * rule_thickness + 0.7 * y_depth +
1484   0.07 * A_height)
1485 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.

```

1486 local sigma_11 = (-axis + 1.5 * rule_thickness + 0.5 * y_depth +
1487   1.1 * A_height)
1488 local sigma_12 = (-axis + 1.35 * rule_thickness + 0.35 * y_depth +
1489   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*A_height` for regular styles and `1.2*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 σ_{18} and σ_{19} was another design choice that worked well.

```

1490 local sigma_13 = 0.6 * A_height      % attempted superscript height
1491 local sigma_15 = 0.5 * A_height      % attempted superscript for \cramped
1492 local sigma_16 = 1.1 * y_depth       % attempted subscript lower
1493 if sigma_16 < 0.2 * A_height then
1494   sigma_16 = 0.2 * A_height
1495 end
1496 local sigma_17 = sigma_16             % sigma_16 when superscript present
1497 local sigma_18 = 0.5 * A_height      % superscript lower for boxed subformula
1498 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.

```

1499 local dim = "DisplayOperatorMinHeight"
1500 if not fontdata.MathConstants[dim] then
1501   fontdata.MathConstants[dim] = 1.8 * A_height
1502 end
1503 local dim = "FractionDelimiterDisplayStyleSize"
1504 if not fontdata.MathConstants[dim] then
1505   fontdata.MathConstants[dim] = 2 * size
1506 end
1507 local dim = "FractionDelimiterSize"
1508 if not fontdata.MathConstants[dim] then
1509   fontdata.MathConstants[dim] = 1.3 * size
1510 end
1511 local dim = "FractionDenominatorDisplayStyleShiftDown"
1512 if not fontdata.MathConstants[dim] then
1513   fontdata.MathConstants[dim] = sigma_11
1514 end
1515 local dim = "FractionDenominatorShiftDown"
1516 if not fontdata.MathConstants[dim] then
1517   fontdata.MathConstants[dim] = sigma_12
1518 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 (σ_8 and σ_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 ξ_{11} larger than ξ_9 .

```

1519 local dim = "FractionDenominatorDisplayStyleGapMin"
1520 if not fontdata.MathConstants[dim] then
1521   fontdata.MathConstants[dim] = rule_thickness + 0.5 * y_depth
1522 end
1523 local dim = "FractionDenominatorGapMin"

```

```

1524 if not fontdata.MathConstants[dim] then
1525   fontdata.MathConstants[dim] = rule_thickness + 0.35 * y_depth
1526 end
1527 local dim = "FractionNumeratorDisplayStyleShiftUp"
1528 if not fontdata.MathConstants[dim] then
1529   fontdata.MathConstants[dim] = sigma_8
1530 end
1531 local dim = "FractionNumeratorShiftUp"
1532 if not fontdata.MathConstants[dim] then
1533   fontdata.MathConstants[dim] = sigma_9
1534 end
1535 local dim = "FractionNumeratorDisplayStyleGapMin"
1536 if not fontdata.MathConstants[dim] then
1537   fontdata.MathConstants[dim] = rule_thickness
1538 end
1539 local dim = "FractionNumeratorGapMin"
1540 if not fontdata.MathConstants[dim] then
1541   fontdata.MathConstants[dim] = rule_thickness
1542 end

```

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

```

1543 local dim = "SkewedFractionHorizontalGap"
1544 if not fontdata.MathConstants[dim] then
1545   fontdata.MathConstants[dim] = 0.5 * em
1546 end
1547 local dim = "SkewedFractionVerticalGap"
1548 if not fontdata.MathConstants[dim] then
1549   fontdata.MathConstants[dim] = ex
1550 end

```

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

```

1551 local dim = "UpperLimitBaselineRiseMin"
1552 if not fontdata.MathConstants[dim] then
1553   fontdata.MathConstants[dim] = xi_11
1554 end
1555 local dim = "UpperLimitGapMin"
1556 if not fontdata.MathConstants[dim] then
1557   fontdata.MathConstants[dim] = xi_9
1558 end
1559 local dim = "LowerLimitBaselineDropMin"
1560 if not fontdata.MathConstants[dim] then
1561   fontdata.MathConstants[dim] = xi_12
1562 end
1563 local dim = "LowerLimitGapMin"
1564 if not fontdata.MathConstants[dim] then
1565   fontdata.MathConstants[dim] = xi_10
1566 end

```

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

```

1567 local dim = "StretchStackGapBelowMin"
1568 if not fontdata.MathConstants[dim] then
1569   fontdata.MathConstants[dim] = xi_10
1570 end
1571 local dim = "StretchStackTopShiftUp"
1572 if not fontdata.MathConstants[dim] then
1573   fontdata.MathConstants[dim] = xi_11
1574 end
1575 local dim = "StretchStackGapAboveMin"
1576 if not fontdata.MathConstants[dim] then
1577   fontdata.MathConstants[dim] = xi_9
1578 end
1579 local dim = "StretchStackBottomShiftDown"
1580 if not fontdata.MathConstants[dim] then
1581   fontdata.MathConstants[dim] = xi_12
1582 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.

```

1583 local dim = "OverbarExtraAscender"
1584 if not fontdata.MathConstants[dim] then
1585   fontdata.MathConstants[dim] = rule_thickness
1586 end
1587 local dim = "OverbarRuleThickness"
1588 if not fontdata.MathConstants[dim] then
1589   fontdata.MathConstants[dim] = rule_thickness
1590 end
1591 local dim = "OverbarVerticalGap"
1592 if not fontdata.MathConstants[dim] then
1593   fontdata.MathConstants[dim] = 2 * rule_thickness
1594 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.

```

1595 local dim = "RadicalExtraAscender"
1596 if not fontdata.MathConstants[dim] then
1597   fontdata.MathConstants[dim] = rule_thickness
1598 end
1599 local dim = "RadicalRuleThickness"
1600 if not fontdata.MathConstants[dim] then
1601   fontdata.MathConstants[dim] = rule_thickness
1602 end
1603 local dim = "RadicalDisplayStyleVerticalGap"
1604 if not fontdata.MathConstants[dim] then

```

```

1605     fontdata.MathConstants[dim] = 2 * rule_thickness
1606 end
1607 local dim = "RadicalVerticalGap"
1608 if not fontdata.MathConstants[dim] then
1609     fontdata.MathConstants[dim] = rule_thickness
1610 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 LuaTeX uses for traditional tfm fonts.

```

1611 local dim = "RadicalKernBeforeDegree"
1612 if not fontdata.MathConstants[dim] then
1613     fontdata.MathConstants[dim] = (5/18) * em
1614 end
1615 local dim = "RadicalKernAfterDegree"
1616 if not fontdata.MathConstants[dim] then
1617     fontdata.MathConstants[dim] = (10/18) * em
1618 end
1619 local dim = "RadicalDegreeBottomRaisePercent"
1620 if not fontdata.MathConstants[dim] then
1621     fontdata.MathConstants[dim] = 60
1622 end

```

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

```

1623 local dim = "SpaceAfterScript"
1624 if not fontdata.MathConstants[dim] then
1625     fontdata.MathConstants[dim] = 0.1 * em
1626 end

```

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

```

1627 local dim = "StackBottomDisplayStyleShiftDown"
1628 if not fontdata.MathConstants[dim] then
1629     fontdata.MathConstants[dim] = sigma_11
1630 end
1631 local dim = "StackBottomShiftDown"
1632 if not fontdata.MathConstants[dim] then
1633     fontdata.MathConstants[dim] = sigma_12
1634 end
1635 local dim = "StackTopDisplayStyleShiftUp"
1636 if not fontdata.MathConstants[dim] then
1637     fontdata.MathConstants[dim] = sigma_8
1638 end
1639 local dim = "StackTopShiftUp"
1640 if not fontdata.MathConstants[dim] then
1641     fontdata.MathConstants[dim] = sigma_10
1642 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.

```

1643 local dim = "StackDisplayStyleGapMin"
1644 if not fontdata.MathConstants[dim] then
1645   fontdata.MathConstants[dim] = 5 * rule_thickness + 0.5 * y_depth
1646 end
1647 local dim = "StackGapMin"
1648 if not fontdata.MathConstants[dim] then
1649   fontdata.MathConstants[dim] = 3 * rule_thickness + 0.35 * y_depth
1650 end

```

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

```

1651 local dim = "SubscriptShiftDown"
1652 if not fontdata.MathConstants[dim] then
1653   fontdata.MathConstants[dim] = sigma_16
1654 end
1655 local dim = "SubscriptBaselineDropMin"
1656 if not fontdata.MathConstants[dim] then
1657   fontdata.MathConstants[dim] = sigma_19
1658 end
1659 local dim = "SubscriptShiftDownWithSuperscript"
1660 if not fontdata.MathConstants[dim] then
1661   fontdata.MathConstants[dim] = sigma_17
1662 end

```

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

```

1663 local dim = "SubscriptTopMax"
1664 if not fontdata.MathConstants[dim] then
1665   fontdata.MathConstants[dim] = 0.5 * A_height
1666 end

```

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

```

1667 local dim = "SubSuperscriptGapMin"
1668 if not fontdata.MathConstants[dim] then
1669   fontdata.MathConstants[dim] = rule_thickness
1670 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.

```

1671 local dim = "SuperscriptBottomMin"
1672 if not fontdata.MathConstants[dim] then
1673   fontdata.MathConstants[dim] = sigma_15 - 0.7 * y_depth

```

```

1674 end
1675 local dim = "SuperscriptBaselineDropMax"
1676 if not fontdata.MathConstants[dim] then
1677   fontdata.MathConstants[dim] = sigma_18
1678 end
1679 local dim = "SuperscriptShiftUp"
1680 if not fontdata.MathConstants[dim] then
1681   fontdata.MathConstants[dim] = sigma_13
1682 end
1683 local dim = "SuperscriptShiftUpCramped"
1684 if not fontdata.MathConstants[dim] then
1685   fontdata.MathConstants[dim] = sigma_15
1686 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.

```

1687 local dim = "SuperscriptBottomMaxWithSubscript"
1688 if not fontdata.MathConstants[dim] then
1689   fontdata.MathConstants[dim] = -sigma_16 + 0.7 * A_height + rule_thickness
1690 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.

```

1691 local dim = "UnderbarExtraDescender"
1692 if not fontdata.MathConstants[dim] then
1693   fontdata.MathConstants[dim] = rule_thickness
1694 end
1695 local dim = "UnderbarRuleThickness"
1696 if not fontdata.MathConstants[dim] then
1697   fontdata.MathConstants[dim] = rule_thickness
1698 end
1699 local dim = "UnderbarVerticalGap"
1700 if not fontdata.MathConstants[dim] then
1701   fontdata.MathConstants[dim] = 2 * rule_thickness
1702 end

```

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

```

1703 local dim = "MinConnectorOverlap"
1704 if not fontdata.MathConstants[dim] then
1705   fontdata.MathConstants[dim] = 0
1706 end
1707 end

```

Time for callbacks! We create six of them.

```

1708 luatexbase.create_callback("mathfont.inspect_font", "simple", mathfont.empty)
1709 luatexbase.create_callback("mathfont.pre_adjust", "simple", mathfont.empty)
1710 luatexbase.create_callback("mathfont.disable_nomath", "simple",
1711   mathfont.set_nomath_true)
1712 luatexbase.create_callback("mathfont.add_math_constants", "simple",
1713   mathfont.math_constants)
1714 luatexbase.create_callback("mathfont.fix_character_metrics", "simple",
1715   mathfont.apply_charm_info)
1716 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.

```

1717 function mathfont.info(msg)
1718   texio.write_nl("log", "Package mathfont Info: " .. msg)
1719 end
1720 function mathfont.get_font_name(fontdata)
1721   return fontdata.fullname or fontdata.psname or fontdata.name or "<??"
1722 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.

```

1723 function mathfont.adjust_font(fontdata)
1724   luatexbase.call_callback("mathfont.inspect_font", fontdata)
1725   if fontdata.nomath then
1726     mathfont.info("Adjusting font " .. mathfont.get_font_name(fontdata) .. ".")
1727     luatexbase.call_callback("mathfont.pre_adjust", fontdata)
1728     luatexbase.call_callback("mathfont.disable_nomath", fontdata)
1729     luatexbase.call_callback("mathfont.add_math_constants", fontdata)
1730     luatexbase.call_callback("mathfont.fix_character_metrics", fontdata)
1731     luatexbase.call_callback("mathfont.post_adjust", fontdata)
1732   else
1733     mathfont.info("No changes made to " ..
1734       mathfont.get_font_name(fontdata) .. ".")
1735   end
1736 end

```

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

```

1737 luatexbase.add_to_callback("luaotfload.patch_font", mathfont.adjust_font,
1738   "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.

```

1739 mathfont:new_type_a(97, 1044480, {50, 50, -50, 0}) % a
1740 mathfont:new_type_a(98, 1044481, {50, 50, -50, 0}) % b
1741 mathfont:new_type_a(99, 1044482, {50, 50, 0, 0}) % c
1742 mathfont:new_type_a(100, 1044483, {50, -50, -50, 0}) % d
1743 mathfont:new_type_a(101, 1044484, {50, 50, 0, 0}) % e
1744 mathfont:new_type_a(102, 1044485, {200, 0, 0, 0}) % f
1745 mathfont:new_type_a(103, 1044486, {100, 50, -50, 0}) % g
1746 mathfont:new_type_a(104, 1044487, {50, 0, -50, 0}) % h
1747 mathfont:new_type_a(105, 1044488, {50, 100, -100, 0}) % i
1748 mathfont:new_type_a(106, 1044489, {400, 50, -50, 0}) % j
1749 mathfont:new_type_a(107, 1044490, {50, 50, -100, 0}) % k
1750 mathfont:new_type_a(108, 1044491, {100, 150, -100, 0}) % l
1751 mathfont:new_type_a(109, 1044492, {50, 0, 0, 0}) % m
1752 mathfont:new_type_a(110, 1044493, {50, 0, 0, 0}) % n
1753 mathfont:new_type_a(111, 1044494, {50, 0, 0, 0}) % o
1754 mathfont:new_type_a(112, 1044495, {200, 50, -50, 0}) % p
1755 mathfont:new_type_a(113, 1044496, {50, 0, -50, 0}) % q
1756 mathfont:new_type_a(114, 1044497, {100, 100, -50, 0}) % r
1757 mathfont:new_type_a(115, 1044498, {50, 50, -50, 0}) % s
1758 mathfont:new_type_a(116, 1044499, {50, 50, -50, 0}) % t
1759 mathfont:new_type_a(117, 1044500, {0, 50, 0, 0}) % u
1760 mathfont:new_type_a(118, 1044501, {0, 50, -50, 0}) % v
1761 mathfont:new_type_a(119, 1044502, {0, 50, 0, 0}) % w
1762 mathfont:new_type_a(120, 1044503, {50, 0, -50, 0}) % x
1763 mathfont:new_type_a(121, 1044504, {150, 50, -50, 0}) % y
1764 mathfont:new_type_a(122, 1044505, {100, 50, -100, 0}) % z
1765 mathfont:new_type_a(305, 1044506, {100, 100, -150, 0}) % \imath
1766 mathfont:new_type_a(567, 1044507, {700, 50, -150, 0}) % \jmath

```

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

```

1767 mathfont:new_type_a(65, 1044512, {50, 0, 150, 0}) % A
1768 mathfont:new_type_a(66, 1044513, {50, 0, 0, 0}) % B
1769 mathfont:new_type_a(67, 1044514, {0, 0, 0, 0}) % C
1770 mathfont:new_type_a(68, 1044515, {50, 0, -50, 0}) % D
1771 mathfont:new_type_a(69, 1044516, {50, 0, 0, 0}) % E
1772 mathfont:new_type_a(70, 1044517, {50, 0, 0, 0}) % F
1773 mathfont:new_type_a(71, 1044518, {0, 0, 0, 0}) % G
1774 mathfont:new_type_a(72, 1044519, {50, 0, -50, 0}) % H
1775 mathfont:new_type_a(73, 1044520, {100, 0, 0, 0}) % I
1776 mathfont:new_type_a(74, 1044521, {50, 0, 100, 0}) % J
1777 mathfont:new_type_a(75, 1044522, {50, 0, 0, 0}) % K
1778 mathfont:new_type_a(76, 1044523, {50, 0, -180, 0}) % L
1779 mathfont:new_type_a(77, 1044524, {50, 0, -50, 0}) % M
1780 mathfont:new_type_a(78, 1044525, {50, 0, -50, 0}) % N
1781 mathfont:new_type_a(79, 1044526, {0, 0, 0, 0}) % O
1782 mathfont:new_type_a(80, 1044527, {0, 0, -50, 0}) % P
1783 mathfont:new_type_a(81, 1044528, {0, 50, 0, 0}) % Q
1784 mathfont:new_type_a(82, 1044529, {50, 0, -50, 0}) % R

```

```

1785 mathfont:new_type_a(83, 1044530, {0, 0, -50, 0}) % S
1786 mathfont:new_type_a(84, 1044531, {0, 0, -50, 0}) % T
1787 mathfont:new_type_a(85, 1044532, {0, 0, -50, 0}) % U
1788 mathfont:new_type_a(86, 1044533, {0, 50, 0, 0}) % V
1789 mathfont:new_type_a(87, 1044534, {0, 50, -50, 0}) % W
1790 mathfont:new_type_a(88, 1044535, {50, 0, 0, 0}) % X
1791 mathfont:new_type_a(89, 1044536, {0, 0, -50, 0}) % Y
1792 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.

```

1793 mathfont:new_type_u(945, {0, 0}) % \alpha
1794 mathfont:new_type_u(946, {0, 0}) % \beta
1795 mathfont:new_type_u(947, {-50, 0}) % \gamma
1796 mathfont:new_type_u(948, {0, 0}) % \delta
1797 mathfont:new_type_u(1013, {50, 0}) % \epsilon
1798 mathfont:new_type_u(950, {0, 0}) % \zeta
1799 mathfont:new_type_u(951, {-50, 0}) % \eta
1800 mathfont:new_type_u(952, {0, 0}) % \theta
1801 mathfont:new_type_u(953, {-50, 0}) % \iota
1802 mathfont:new_type_u(954, {0, 0}) % \kappa
1803 mathfont:new_type_u(955, {-150, 0}) % \lambda
1804 mathfont:new_type_u(956, {0, 0}) % \mu
1805 mathfont:new_type_u(957, {-50, 0}) % \nu
1806 mathfont:new_type_u(958, {0, 0}) % \xi
1807 mathfont:new_type_u(959, {0, 0}) % \omicron
1808 mathfont:new_type_u(960, {-100, 0}) % \pi
1809 mathfont:new_type_u(961, {-50, 0}) % \rho
1810 mathfont:new_type_u(963, {-100, 0}) % \sigma
1811 mathfont:new_type_u(964, {-100, 0}) % \tau
1812 mathfont:new_type_u(965, {-50, 0}) % \upsilon
1813 mathfont:new_type_u(981, {0, 0}) % \phi
1814 mathfont:new_type_u(967, {-50, 0}) % \chi
1815 mathfont:new_type_u(968, {-50, 0}) % \psi
1816 mathfont:new_type_u(969, {0, 0}) % \omega
1817 mathfont:new_type_u(976, {0, 0}) % \varbeta
1818 mathfont:new_type_u(949, {-50, 0}) % \varepsilon
1819 mathfont:new_type_u(977, {50, 0}) % \vartheta
1820 mathfont:new_type_u(1009, {-50, 0}) % \varrho
1821 mathfont:new_type_u(962, {-50, 0}) % \varsigma
1822 mathfont:new_type_u(966, {0, 0}) % \varphi

```

Upper-case Greek characters. Same as previously.

```

1823 mathfont:new_type_u(913, {0, 0}) % \Alpha
1824 mathfont:new_type_u(914, {0, 0}) % \Beta
1825 mathfont:new_type_u(915, {0, 0}) % \Gamma
1826 mathfont:new_type_u(916, {0, 0}) % \Delta
1827 mathfont:new_type_u(917, {0, 0}) % \Epsilon

```

```

1828 mathfont:new_type_u(918, {0, 0}) % \Zeta
1829 mathfont:new_type_u(919, {0, 0}) % \Eta
1830 mathfont:new_type_u(920, {0, 0}) % \Theta
1831 mathfont:new_type_u(921, {0, 0}) % \Iota
1832 mathfont:new_type_u(922, {0, 0}) % \Kappa
1833 mathfont:new_type_u(923, {0, 0}) % \Lambda
1834 mathfont:new_type_u(924, {0, 0}) % \Mu
1835 mathfont:new_type_u(925, {0, 0}) % \Nu
1836 mathfont:new_type_u(926, {0, 0}) % \Xi
1837 mathfont:new_type_u(927, {0, 0}) % \Omicron
1838 mathfont:new_type_u(928, {0, 0}) % \Pi
1839 mathfont:new_type_u(929, {0, 0}) % \Rho
1840 mathfont:new_type_u(931, {0, 0}) % \Sigma
1841 mathfont:new_type_u(932, {0, 0}) % \Tau
1842 mathfont:new_type_u(933, {0, 0}) % \Upsilon
1843 mathfont:new_type_u(934, {0, 0}) % \Phi
1844 mathfont:new_type_u(935, {0, 0}) % \Chi
1845 mathfont:new_type_u(936, {0, 0}) % \Psi
1846 mathfont:new_type_u(937, {0, 0}) % \Omega
1847 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`.

```

1848 local delim_glyphs = {40, % (
1849   41,      % )
1850   47,      % /
1851   91,      % [
1852   92,      % backslash
1853   93,      % ]
1854  123,     % {
1855  125,     % }
1856  8249,    % \lguil
1857  8250,    % \rguil
1858  171,     % \llguil
1859  187,     % \rrguil
1860  1044508, % \fakelangle
1861  1044509, % \fakerangle
1862  1044510, % \fakellangle
1863  1044511} % \fakerrangle
1864 local big_op_glyphs = {33, % !
1865   35,     % #
1866   36,     % $
1867   37,     % %
1868   38,     % &
1869   43,     % +

```

```

1870 63,      % ?
1871 64,      % @
1872 167,     % \S
1873 215,     % \times
1874 247,     % \div
1875 8719,    % \prod
1876 8721,    % \sum
1877 8720,    % \coprod
1878 8897,    % \bigvee
1879 8896,    % \bigwedge
1880 8899,    % \bigcup
1881 8898,    % \bigcap
1882 10753,   % \bigoplus
1883 10754,   % \bigotimes
1884 10752,   % \bigodot
1885 10757,   % \bigsqcap
1886 10758}   % \bigsqcup
1887 local vert_glyphs = {124, 8730} % | and \surd
1888 local int_glyphs = {8747, % \intop
1889 8748,     % \iint
1890 8749,     % \iiint
1891 8750,     % \oint
1892 8751,     % \oiint
1893 8752}     % \oiiint

```

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

```
1894 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.

```

1895 local delim_scale = {}
1896 local big_op_scale = {}
1897 local vert_scale = {}
1898 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.

```

1899 for i = 1, 15, 1 do
1900   delim_scale[2*i-1] = 1000 + 100*i % horizontal - delimiters
1901   delim_scale[2*i] = 1000 + 500*i % vertical - delimiters
1902   vert_scale[2*i-1] = 1000
1903   vert_scale[2*i] = 1000 + 500*i % vertical - vertically scaled chars
1904   big_op_scale[2*i-1] = 1000 + 100*i % horizontal - big operators
1905   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 be-

tween the font's value of `\Umathoperator` 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 `\Umathoperator` setting in `populate_math_constants`. Effectively this means that the user should take the size of the integral operator as fixed and should set `\Umathoperator` to make all other big operators the desired size.

```
1906 int_scale[2*i-1] = 1000 + 500*i    % horizontal - integral sign
1907 int_scale[2*i]   = 1000 + 1500*i   % vertical - integral sign
1908 end
```

We do not modify accent placement.

```
1909 delim_scale[31] = 0
1910 delim_scale[32] = 0
1911 big_op_scale[31] = 0
1912 big_op_scale[32] = 0
1913 vert_scale[31]  = 0
1914 vert_scale[32]  = 0
1915 int_scale[31]   = 0
1916 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.

```
1917 function mathfont:add_extensible_variants(first_smash, glyph_list, scale_list)
1918     local variants = (\string# scale_list - 2) / 2
1919     local curr_smash = first_smash
1920     for i = 1, \string# glyph_list, 1 do
1921         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`.

```
1922     local curr_slots = {}
1923     for j = 1, variants, 1 do
1924         curr_slots[j] = curr_smash + j
1925     end
```

Add the charm information and increment `smash`.

```
1926     self:new_type_e(curr_char, curr_smash, curr_slots, scale_list)
1927     smash = smash + variants + 1
1928     curr_smash = smash
1929     end
1930 end
```

Add the charm information for the type e characters.

```
1931 mathfont:add_extensible_variants(smash, delim_glyphs, delim_scale)
1932 mathfont:add_extensible_variants(smash, big_op_glyphs, big_op_scale)
1933 mathfont:add_extensible_variants(smash, vert_glyphs, vert_scale)
1934 mathfont:add_extensible_variants(smash, int_glyphs, int_scale)
```


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

```
1935 }
1936 \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 L^AT_EX 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.

```
1937 \ifM@adjust@font
1938   \def\M@upper@set{%
1939     \edef\M@upper@font{M\M@uppershape\@tempa}
1940     \DeclareMathSymbol{A}{\mathalpha}{\M@upper@font}{1044512}
1941     \DeclareMathSymbol{B}{\mathalpha}{\M@upper@font}{1044513}
1942     \DeclareMathSymbol{C}{\mathalpha}{\M@upper@font}{1044514}
1943     \DeclareMathSymbol{D}{\mathalpha}{\M@upper@font}{1044515}
1944     \DeclareMathSymbol{E}{\mathalpha}{\M@upper@font}{1044516}
1945     \DeclareMathSymbol{F}{\mathalpha}{\M@upper@font}{1044517}
1946     \DeclareMathSymbol{G}{\mathalpha}{\M@upper@font}{1044518}
1947     \DeclareMathSymbol{H}{\mathalpha}{\M@upper@font}{1044519}
1948     \DeclareMathSymbol{I}{\mathalpha}{\M@upper@font}{1044520}
1949     \DeclareMathSymbol{J}{\mathalpha}{\M@upper@font}{1044521}
1950     \DeclareMathSymbol{K}{\mathalpha}{\M@upper@font}{1044522}
1951     \DeclareMathSymbol{L}{\mathalpha}{\M@upper@font}{1044523}
1952     \DeclareMathSymbol{M}{\mathalpha}{\M@upper@font}{1044524}
1953     \DeclareMathSymbol{N}{\mathalpha}{\M@upper@font}{1044525}
1954     \DeclareMathSymbol{O}{\mathalpha}{\M@upper@font}{1044526}
1955     \DeclareMathSymbol{P}{\mathalpha}{\M@upper@font}{1044527}
1956     \DeclareMathSymbol{Q}{\mathalpha}{\M@upper@font}{1044528}
1957     \DeclareMathSymbol{R}{\mathalpha}{\M@upper@font}{1044529}
1958     \DeclareMathSymbol{S}{\mathalpha}{\M@upper@font}{1044530}
1959     \DeclareMathSymbol{T}{\mathalpha}{\M@upper@font}{1044531}
1960     \DeclareMathSymbol{U}{\mathalpha}{\M@upper@font}{1044532}
1961     \DeclareMathSymbol{V}{\mathalpha}{\M@upper@font}{1044533}
1962     \DeclareMathSymbol{W}{\mathalpha}{\M@upper@font}{1044534}
1963     \DeclareMathSymbol{X}{\mathalpha}{\M@upper@font}{1044535}
1964     \DeclareMathSymbol{Y}{\mathalpha}{\M@upper@font}{1044536}
1965     \DeclareMathSymbol{Z}{\mathalpha}{\M@upper@font}{1044537}}
1966 \else
1967   \def\M@upper@set{%
1968     \edef\M@upper@font{M\M@uppershape\@tempa}
1969     \DeclareMathSymbol{A}{\mathalpha}{\M@upper@font}{`A}
1970     \DeclareMathSymbol{B}{\mathalpha}{\M@upper@font}{`B}
1971     \DeclareMathSymbol{C}{\mathalpha}{\M@upper@font}{`C}
1972     \DeclareMathSymbol{D}{\mathalpha}{\M@upper@font}{`D}
```

```

1973 \DeclareMathSymbol{E}{\mathalpha}{\M@upper@font}{`E}
1974 \DeclareMathSymbol{F}{\mathalpha}{\M@upper@font}{`F}
1975 \DeclareMathSymbol{G}{\mathalpha}{\M@upper@font}{`G}
1976 \DeclareMathSymbol{H}{\mathalpha}{\M@upper@font}{`H}
1977 \DeclareMathSymbol{I}{\mathalpha}{\M@upper@font}{`I}
1978 \DeclareMathSymbol{J}{\mathalpha}{\M@upper@font}{`J}
1979 \DeclareMathSymbol{K}{\mathalpha}{\M@upper@font}{`K}
1980 \DeclareMathSymbol{L}{\mathalpha}{\M@upper@font}{`L}
1981 \DeclareMathSymbol{M}{\mathalpha}{\M@upper@font}{`M}
1982 \DeclareMathSymbol{N}{\mathalpha}{\M@upper@font}{`N}
1983 \DeclareMathSymbol{O}{\mathalpha}{\M@upper@font}{`O}
1984 \DeclareMathSymbol{P}{\mathalpha}{\M@upper@font}{`P}
1985 \DeclareMathSymbol{Q}{\mathalpha}{\M@upper@font}{`Q}
1986 \DeclareMathSymbol{R}{\mathalpha}{\M@upper@font}{`R}
1987 \DeclareMathSymbol{S}{\mathalpha}{\M@upper@font}{`S}
1988 \DeclareMathSymbol{T}{\mathalpha}{\M@upper@font}{`T}
1989 \DeclareMathSymbol{U}{\mathalpha}{\M@upper@font}{`U}
1990 \DeclareMathSymbol{V}{\mathalpha}{\M@upper@font}{`V}
1991 \DeclareMathSymbol{W}{\mathalpha}{\M@upper@font}{`W}
1992 \DeclareMathSymbol{X}{\mathalpha}{\M@upper@font}{`X}
1993 \DeclareMathSymbol{Y}{\mathalpha}{\M@upper@font}{`Y}
1994 \DeclareMathSymbol{Z}{\mathalpha}{\M@upper@font}{`Z}}
1995 \fi

```

Set lower-case Latin characters.

```

1996 \ifM@adjust@font
1997 \def\M@lower@set{%
1998 \edef\M@lower@font{M\M@lowershape\@tempa}
1999 \DeclareMathSymbol{a}{\mathalpha}{\M@lower@font}{1044480}
2000 \DeclareMathSymbol{b}{\mathalpha}{\M@lower@font}{1044481}
2001 \DeclareMathSymbol{c}{\mathalpha}{\M@lower@font}{1044482}
2002 \DeclareMathSymbol{d}{\mathalpha}{\M@lower@font}{1044483}
2003 \DeclareMathSymbol{e}{\mathalpha}{\M@lower@font}{1044484}
2004 \DeclareMathSymbol{f}{\mathalpha}{\M@lower@font}{1044485}
2005 \DeclareMathSymbol{g}{\mathalpha}{\M@lower@font}{1044486}
2006 \DeclareMathSymbol{h}{\mathalpha}{\M@lower@font}{1044487}
2007 \DeclareMathSymbol{i}{\mathalpha}{\M@lower@font}{1044488}
2008 \DeclareMathSymbol{j}{\mathalpha}{\M@lower@font}{1044489}
2009 \DeclareMathSymbol{k}{\mathalpha}{\M@lower@font}{1044490}
2010 \DeclareMathSymbol{l}{\mathalpha}{\M@lower@font}{1044491}
2011 \DeclareMathSymbol{m}{\mathalpha}{\M@lower@font}{1044492}
2012 \DeclareMathSymbol{n}{\mathalpha}{\M@lower@font}{1044493}
2013 \DeclareMathSymbol{o}{\mathalpha}{\M@lower@font}{1044494}
2014 \DeclareMathSymbol{p}{\mathalpha}{\M@lower@font}{1044495}
2015 \DeclareMathSymbol{q}{\mathalpha}{\M@lower@font}{1044496}
2016 \DeclareMathSymbol{r}{\mathalpha}{\M@lower@font}{1044497}
2017 \DeclareMathSymbol{s}{\mathalpha}{\M@lower@font}{1044498}
2018 \DeclareMathSymbol{t}{\mathalpha}{\M@lower@font}{1044499}

```

```

2019 \DeclareMathSymbol{u}{\mathalpha}{\M@lower@font}{1044500}
2020 \DeclareMathSymbol{v}{\mathalpha}{\M@lower@font}{1044501}
2021 \DeclareMathSymbol{w}{\mathalpha}{\M@lower@font}{1044502}
2022 \DeclareMathSymbol{x}{\mathalpha}{\M@lower@font}{1044503}
2023 \DeclareMathSymbol{y}{\mathalpha}{\M@lower@font}{1044504}
2024 \DeclareMathSymbol{z}{\mathalpha}{\M@lower@font}{1044505}
2025 \DeclareMathSymbol{\imath}{\mathalpha}{\M@lower@font}{1044506}
2026 \DeclareMathSymbol{\jmath}{\mathalpha}{\M@lower@font}{1044507}
2027 \DeclareMathSymbol{\hbar}{\mathord}{\M@lower@font}{"127}
2028 \else
2029 \def\M@lower@set{%
2030 \edef\M@lower@font{M\M@lower@shape\@tempa}
2031 \DeclareMathSymbol{a}{\mathalpha}{\M@lower@font}{`a}
2032 \DeclareMathSymbol{b}{\mathalpha}{\M@lower@font}{`b}
2033 \DeclareMathSymbol{c}{\mathalpha}{\M@lower@font}{`c}
2034 \DeclareMathSymbol{d}{\mathalpha}{\M@lower@font}{`d}
2035 \DeclareMathSymbol{e}{\mathalpha}{\M@lower@font}{`e}
2036 \DeclareMathSymbol{f}{\mathalpha}{\M@lower@font}{`f}
2037 \DeclareMathSymbol{g}{\mathalpha}{\M@lower@font}{`g}
2038 \DeclareMathSymbol{h}{\mathalpha}{\M@lower@font}{`h}
2039 \DeclareMathSymbol{i}{\mathalpha}{\M@lower@font}{`i}
2040 \DeclareMathSymbol{j}{\mathalpha}{\M@lower@font}{`j}
2041 \DeclareMathSymbol{k}{\mathalpha}{\M@lower@font}{`k}
2042 \DeclareMathSymbol{l}{\mathalpha}{\M@lower@font}{`l}
2043 \DeclareMathSymbol{m}{\mathalpha}{\M@lower@font}{`m}
2044 \DeclareMathSymbol{n}{\mathalpha}{\M@lower@font}{`n}
2045 \DeclareMathSymbol{o}{\mathalpha}{\M@lower@font}{`o}
2046 \DeclareMathSymbol{p}{\mathalpha}{\M@lower@font}{`p}
2047 \DeclareMathSymbol{q}{\mathalpha}{\M@lower@font}{`q}
2048 \DeclareMathSymbol{r}{\mathalpha}{\M@lower@font}{`r}
2049 \DeclareMathSymbol{s}{\mathalpha}{\M@lower@font}{`s}
2050 \DeclareMathSymbol{t}{\mathalpha}{\M@lower@font}{`t}
2051 \DeclareMathSymbol{u}{\mathalpha}{\M@lower@font}{`u}
2052 \DeclareMathSymbol{v}{\mathalpha}{\M@lower@font}{`v}
2053 \DeclareMathSymbol{w}{\mathalpha}{\M@lower@font}{`w}
2054 \DeclareMathSymbol{x}{\mathalpha}{\M@lower@font}{`x}
2055 \DeclareMathSymbol{y}{\mathalpha}{\M@lower@font}{`y}
2056 \DeclareMathSymbol{z}{\mathalpha}{\M@lower@font}{`z}
2057 \DeclareMathSymbol{\imath}{\mathalpha}{\M@lower@font}{"131}
2058 \DeclareMathSymbol{\jmath}{\mathalpha}{\M@lower@font}{"237}
2059 \DeclareMathSymbol{\hbar}{\mathord}{\M@lower@font}{"127}
2060 \fi
Set diacritics.
2061 \def\M@diacritics@set{%
2062 \edef\M@diacritics@font{M\M@diacritics@shape\@tempa}
2063 \DeclareMathAccent{\acute}{\mathalpha}{\M@diacritics@font}{"B4}
2064 \DeclareMathAccent{\aacute}{\mathalpha}{\M@diacritics@font}{"2DD}

```

```

2065 \DeclareMathAccent{\dot}{\mathalpha}{\M@diacritics@font}{"2D9}
2066 \DeclareMathAccent{\ddot}{\mathalpha}{\M@diacritics@font}{"A8}
2067 \DeclareMathAccent{\grave}{\mathalpha}{\M@diacritics@font}{"60}
2068 \DeclareMathAccent{\breve}{\mathalpha}{\M@diacritics@font}{"2D8}
2069 \DeclareMathAccent{\hat}{\mathalpha}{\M@diacritics@font}{"2C6}
2070 \DeclareMathAccent{\check}{\mathalpha}{\M@diacritics@font}{"2C7}
2071 \DeclareMathAccent{\bar}{\mathalpha}{\M@diacritics@font}{"2C9}
2072 \DeclareMathAccent{\mathring}{\mathalpha}{\M@diacritics@font}{"2DA}
2073 \DeclareMathAccent{\tilde}{\mathalpha}{\M@diacritics@font}{"2DC}

```

Set capital Greek characters.

```

2074 \def\M@greekupper@set{%
2075   \edef\M@greekupper@font{M\M@greekupper@shape\@tempa}
2076   \DeclareMathSymbol{\Alpha}{\mathalpha}{\M@greekupper@font}{"391}
2077   \DeclareMathSymbol{\Beta}{\mathalpha}{\M@greekupper@font}{"392}
2078   \DeclareMathSymbol{\Gamma}{\mathalpha}{\M@greekupper@font}{"393}
2079   \DeclareMathSymbol{\Delta}{\mathalpha}{\M@greekupper@font}{"394}
2080   \DeclareMathSymbol{\Epsilon}{\mathalpha}{\M@greekupper@font}{"395}
2081   \DeclareMathSymbol{\Zeta}{\mathalpha}{\M@greekupper@font}{"396}
2082   \DeclareMathSymbol{\Eta}{\mathalpha}{\M@greekupper@font}{"397}
2083   \DeclareMathSymbol{\Theta}{\mathalpha}{\M@greekupper@font}{"398}
2084   \DeclareMathSymbol{\Iota}{\mathalpha}{\M@greekupper@font}{"399}
2085   \DeclareMathSymbol{\Kappa}{\mathalpha}{\M@greekupper@font}{"39A}
2086   \DeclareMathSymbol{\Lambda}{\mathalpha}{\M@greekupper@font}{"39B}
2087   \DeclareMathSymbol{\Mu}{\mathalpha}{\M@greekupper@font}{"39C}
2088   \DeclareMathSymbol{\Nu}{\mathalpha}{\M@greekupper@font}{"39D}
2089   \DeclareMathSymbol{\Xi}{\mathalpha}{\M@greekupper@font}{"39E}
2090   \DeclareMathSymbol{\Omicron}{\mathalpha}{\M@greekupper@font}{"39F}
2091   \DeclareMathSymbol{\Pi}{\mathalpha}{\M@greekupper@font}{"3A0}
2092   \DeclareMathSymbol{\Rho}{\mathalpha}{\M@greekupper@font}{"3A1}
2093   \DeclareMathSymbol{\Sigma}{\mathalpha}{\M@greekupper@font}{"3A3}
2094   \DeclareMathSymbol{\Tau}{\mathalpha}{\M@greekupper@font}{"3A4}
2095   \DeclareMathSymbol{\Upsilon}{\mathalpha}{\M@greekupper@font}{"3A5}
2096   \DeclareMathSymbol{\Phi}{\mathalpha}{\M@greekupper@font}{"3A6}
2097   \DeclareMathSymbol{\Chi}{\mathalpha}{\M@greekupper@font}{"3A7}
2098   \DeclareMathSymbol{\Psi}{\mathalpha}{\M@greekupper@font}{"3A8}
2099   \DeclareMathSymbol{\Omega}{\mathalpha}{\M@greekupper@font}{"3A9}
2100   \DeclareMathSymbol{\varTheta}{\mathalpha}{\M@greekupper@font}{"3F4}

```

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

```

2101 \ifM@adjust@font
2102   \ifM@symbols\else
2103     \DeclareMathSymbol{\increment}{\mathord}{\M@greekupper@font}{"2206}
2104     \DeclareMathSymbol{\nabla}{\mathord}{\M@greekupper@font}{"2207}
2105   \fi
2106 \else
2107   \ifM@symbols\else
2108     \DeclareMathSymbol{\increment}{\mathord}{\M@greekupper@font}{"2206}

```

```

2109 \fi
2110 \ifM@extsymbols\else
2111 \DeclareMathSymbol{\nabla}{\mathord}{\M@greekupper@font}{"2207}
2112 \fi
2113 \fi}

```

Set minuscule Greek characters.

```

2114 \def\M@greeklower@set{%
2115 \edef\M@greeklower@font{M\M@greeklower@shape\@tempa}
2116 \DeclareMathSymbol{\alpha}{\mathalpha}{\M@greeklower@font}{"3B1}
2117 \DeclareMathSymbol{\beta}{\mathalpha}{\M@greeklower@font}{"3B2}
2118 \DeclareMathSymbol{\gamma}{\mathalpha}{\M@greeklower@font}{"3B3}
2119 \DeclareMathSymbol{\delta}{\mathalpha}{\M@greeklower@font}{"3B4}
2120 \DeclareMathSymbol{\epsilon}{\mathalpha}{\M@greeklower@font}{"3B5}
2121 \DeclareMathSymbol{\zeta}{\mathalpha}{\M@greeklower@font}{"3B6}
2122 \DeclareMathSymbol{\eta}{\mathalpha}{\M@greeklower@font}{"3B7}
2123 \DeclareMathSymbol{\theta}{\mathalpha}{\M@greeklower@font}{"3B8}
2124 \DeclareMathSymbol{\iota}{\mathalpha}{\M@greeklower@font}{"3B9}
2125 \DeclareMathSymbol{\kappa}{\mathalpha}{\M@greeklower@font}{"3BA}
2126 \DeclareMathSymbol{\lambda}{\mathalpha}{\M@greeklower@font}{"3BB}
2127 \DeclareMathSymbol{\mu}{\mathalpha}{\M@greeklower@font}{"3BC}
2128 \DeclareMathSymbol{\nu}{\mathalpha}{\M@greeklower@font}{"3BD}
2129 \DeclareMathSymbol{\xi}{\mathalpha}{\M@greeklower@font}{"3BE}
2130 \DeclareMathSymbol{\omicron}{\mathalpha}{\M@greeklower@font}{"3BF}
2131 \DeclareMathSymbol{\pi}{\mathalpha}{\M@greeklower@font}{"3C0}
2132 \DeclareMathSymbol{\rho}{\mathalpha}{\M@greeklower@font}{"3C1}
2133 \DeclareMathSymbol{\sigma}{\mathalpha}{\M@greeklower@font}{"3C3}
2134 \DeclareMathSymbol{\tau}{\mathalpha}{\M@greeklower@font}{"3C4}
2135 \DeclareMathSymbol{\upsilon}{\mathalpha}{\M@greeklower@font}{"3C5}
2136 \DeclareMathSymbol{\phi}{\mathalpha}{\M@greeklower@font}{"3C6}
2137 \DeclareMathSymbol{\chi}{\mathalpha}{\M@greeklower@font}{"3C7}
2138 \DeclareMathSymbol{\psi}{\mathalpha}{\M@greeklower@font}{"3C8}
2139 \DeclareMathSymbol{\omega}{\mathalpha}{\M@greeklower@font}{"3C9}
2140 \DeclareMathSymbol{\varbeta}{\mathalpha}{\M@greeklower@font}{"3D0}
2141 \DeclareMathSymbol{\varepsilon}{\mathalpha}{\M@greeklower@font}{"3F5}
2142 \DeclareMathSymbol{\varkappa}{\mathalpha}{\M@greeklower@font}{"3F0}
2143 \DeclareMathSymbol{\vartheta}{\mathalpha}{\M@greeklower@font}{"3D1}
2144 \DeclareMathSymbol{\varrho}{\mathalpha}{\M@greeklower@font}{"3F1}
2145 \DeclareMathSymbol{\varsigma}{\mathalpha}{\M@greeklower@font}{"3C2}
2146 \DeclareMathSymbol{\varphi}{\mathalpha}{\M@greeklower@font}{"3D5}}

```

Set capital ancient Greek characters.

```

2147 \def\M@agreekupper@set{%
2148 \edef\M@agreekupper@font{M\M@agreekupper@shape\@tempa}
2149 \DeclareMathSymbol{\Heta}{\mathalpha}{\M@agreekupper@font}{"370}
2150 \DeclareMathSymbol{\Sampi}{\mathalpha}{\M@agreekupper@font}{"3E0}
2151 \DeclareMathSymbol{\Digamma}{\mathalpha}{\M@agreekupper@font}{"3DC}
2152 \DeclareMathSymbol{\Koppa}{\mathalpha}{\M@agreekupper@font}{"3D8}
2153 \DeclareMathSymbol{\Stigma}{\mathalpha}{\M@agreekupper@font}{"3DA}

```

```

2154 \DeclareMathSymbol{\Sho}{\mathalpha}{\M@agreekupper@font}{"3F7}
2155 \DeclareMathSymbol{\San}{\mathalpha}{\M@agreekupper@font}{"3FA}
2156 \DeclareMathSymbol{\varSampi}{\mathalpha}{\M@agreekupper@font}{"372}
2157 \DeclareMathSymbol{\varDigamma}{\mathalpha}{\M@agreekupper@font}{"376}
2158 \DeclareMathSymbol{\varKoppa}{\mathalpha}{\M@agreekupper@font}{"3DE}}

```

Set minuscule ancient Greek characters.

```

2159 \def\M@agreeklower@set{%
2160 \edef\M@agreeklower@font{M\M@agreeklower@shape\@tempa}
2161 \DeclareMathSymbol{\heta}{\mathalpha}{\M@agreeklower@font}{"371}
2162 \DeclareMathSymbol{\sampi}{\mathalpha}{\M@agreeklower@font}{"3E1}
2163 \DeclareMathSymbol{\digamma}{\mathalpha}{\M@agreeklower@font}{"3DD}
2164 \DeclareMathSymbol{\koppa}{\mathalpha}{\M@agreeklower@font}{"3D9}
2165 \DeclareMathSymbol{\stigma}{\mathalpha}{\M@agreeklower@font}{"3DB}
2166 \DeclareMathSymbol{\sho}{\mathalpha}{\M@agreeklower@font}{"3F8}
2167 \DeclareMathSymbol{\san}{\mathalpha}{\M@agreeklower@font}{"3FB}
2168 \DeclareMathSymbol{\varsampi}{\mathalpha}{\M@agreeklower@font}{"373}
2169 \DeclareMathSymbol{\vardigamma}{\mathalpha}{\M@agreeklower@font}{"377}
2170 \DeclareMathSymbol{\varkoppa}{\mathalpha}{\M@agreeklower@font}{"3DF}}

```

Set capital Cyrillic characters.

```

2171 \def\M@cyrillicupper@set{%
2172 \edef\M@cyrillicupper@font{M\M@cyrillicupper@shape\@tempa}
2173 \DeclareMathSymbol{\cyrA}{\mathalpha}{\M@cyrillicupper@font}{"410}
2174 \DeclareMathSymbol{\cyrBe}{\mathalpha}{\M@cyrillicupper@font}{"411}
2175 \DeclareMathSymbol{\cyrVe}{\mathalpha}{\M@cyrillicupper@font}{"412}
2176 \DeclareMathSymbol{\cyrGhe}{\mathalpha}{\M@cyrillicupper@font}{"413}
2177 \DeclareMathSymbol{\cyrDe}{\mathalpha}{\M@cyrillicupper@font}{"414}
2178 \DeclareMathSymbol{\cyrIe}{\mathalpha}{\M@cyrillicupper@font}{"415}
2179 \DeclareMathSymbol{\cyrZhe}{\mathalpha}{\M@cyrillicupper@font}{"416}
2180 \DeclareMathSymbol{\cyrZe}{\mathalpha}{\M@cyrillicupper@font}{"417}
2181 \DeclareMathSymbol{\cyrI}{\mathalpha}{\M@cyrillicupper@font}{"418}
2182 \DeclareMathSymbol{\cyrKa}{\mathalpha}{\M@cyrillicupper@font}{"41A}
2183 \DeclareMathSymbol{\cyrEl}{\mathalpha}{\M@cyrillicupper@font}{"41B}
2184 \DeclareMathSymbol{\cyrEm}{\mathalpha}{\M@cyrillicupper@font}{"41C}
2185 \DeclareMathSymbol{\cyrEn}{\mathalpha}{\M@cyrillicupper@font}{"41D}
2186 \DeclareMathSymbol{\cyrO}{\mathalpha}{\M@cyrillicupper@font}{"41E}
2187 \DeclareMathSymbol{\cyrPe}{\mathalpha}{\M@cyrillicupper@font}{"41F}
2188 \DeclareMathSymbol{\cyrEr}{\mathalpha}{\M@cyrillicupper@font}{"420}
2189 \DeclareMathSymbol{\cyrEs}{\mathalpha}{\M@cyrillicupper@font}{"421}
2190 \DeclareMathSymbol{\cyrTe}{\mathalpha}{\M@cyrillicupper@font}{"422}
2191 \DeclareMathSymbol{\cyrU}{\mathalpha}{\M@cyrillicupper@font}{"423}
2192 \DeclareMathSymbol{\cyrEf}{\mathalpha}{\M@cyrillicupper@font}{"424}
2193 \DeclareMathSymbol{\cyrHa}{\mathalpha}{\M@cyrillicupper@font}{"425}
2194 \DeclareMathSymbol{\cyrTse}{\mathalpha}{\M@cyrillicupper@font}{"426}
2195 \DeclareMathSymbol{\cyrChe}{\mathalpha}{\M@cyrillicupper@font}{"427}
2196 \DeclareMathSymbol{\cyrSha}{\mathalpha}{\M@cyrillicupper@font}{"428}
2197 \DeclareMathSymbol{\cyrShcha}{\mathalpha}{\M@cyrillicupper@font}{"429}
2198 \DeclareMathSymbol{\cyrHard}{\mathalpha}{\M@cyrillicupper@font}{"42A}

```

```

2199 \DeclareMathSymbol{\cyrYeru}{\mathalpha}{\M@cyrillicupper@font}{"42B}
2200 \DeclareMathSymbol{\cyrSoft}{\mathalpha}{\M@cyrillicupper@font}{"42C}
2201 \DeclareMathSymbol{\cyrE}{\mathalpha}{\M@cyrillicupper@font}{"42D}
2202 \DeclareMathSymbol{\cyrYu}{\mathalpha}{\M@cyrillicupper@font}{"42E}
2203 \DeclareMathSymbol{\cyrYa}{\mathalpha}{\M@cyrillicupper@font}{"42F}
2204 \DeclareMathSymbol{\cyrvarI}{\mathalpha}{\M@cyrillicupper@font}{"419}}

```

Set minuscule Cyrillic characters.

```

2205 \def\M@cyrilliclower@set{%
2206   \edef\M@cyrilliclower@font{M\M@cyrilliclower@shape\@tempa}
2207   \DeclareMathSymbol{\cyra}{\mathalpha}{\M@cyrilliclower@font}{"430}
2208   \DeclareMathSymbol{\cyrbe}{\mathalpha}{\M@cyrilliclower@font}{"431}
2209   \DeclareMathSymbol{\cyrve}{\mathalpha}{\M@cyrilliclower@font}{"432}
2210   \DeclareMathSymbol{\cyrge}{\mathalpha}{\M@cyrilliclower@font}{"433}
2211   \DeclareMathSymbol{\cyrde}{\mathalpha}{\M@cyrilliclower@font}{"434}
2212   \DeclareMathSymbol{\cyrje}{\mathalpha}{\M@cyrilliclower@font}{"435}
2213   \DeclareMathSymbol{\cyrzhe}{\mathalpha}{\M@cyrilliclower@font}{"436}
2214   \DeclareMathSymbol{\cyrze}{\mathalpha}{\M@cyrilliclower@font}{"437}
2215   \DeclareMathSymbol{\cyri}{\mathalpha}{\M@cyrilliclower@font}{"438}
2216   \DeclareMathSymbol{\cyrka}{\mathalpha}{\M@cyrilliclower@font}{"43A}
2217   \DeclareMathSymbol{\cyrel}{\mathalpha}{\M@cyrilliclower@font}{"43B}
2218   \DeclareMathSymbol{\cyrem}{\mathalpha}{\M@cyrilliclower@font}{"43C}
2219   \DeclareMathSymbol{\cyren}{\mathalpha}{\M@cyrilliclower@font}{"43D}
2220   \DeclareMathSymbol{\cyro}{\mathalpha}{\M@cyrilliclower@font}{"43E}
2221   \DeclareMathSymbol{\cyrpe}{\mathalpha}{\M@cyrilliclower@font}{"43F}
2222   \DeclareMathSymbol{\cyrer}{\mathalpha}{\M@cyrilliclower@font}{"440}
2223   \DeclareMathSymbol{\cyres}{\mathalpha}{\M@cyrilliclower@font}{"441}
2224   \DeclareMathSymbol{\cyrte}{\mathalpha}{\M@cyrilliclower@font}{"442}
2225   \DeclareMathSymbol{\cyru}{\mathalpha}{\M@cyrilliclower@font}{"443}
2226   \DeclareMathSymbol{\cyref}{\mathalpha}{\M@cyrilliclower@font}{"444}
2227   \DeclareMathSymbol{\cyrha}{\mathalpha}{\M@cyrilliclower@font}{"445}
2228   \DeclareMathSymbol{\cyrtshe}{\mathalpha}{\M@cyrilliclower@font}{"446}
2229   \DeclareMathSymbol{\cyrche}{\mathalpha}{\M@cyrilliclower@font}{"447}
2230   \DeclareMathSymbol{\cyrsha}{\mathalpha}{\M@cyrilliclower@font}{"448}
2231   \DeclareMathSymbol{\cyrshcha}{\mathalpha}{\M@cyrilliclower@font}{"449}
2232   \DeclareMathSymbol{\cyrhard}{\mathalpha}{\M@cyrilliclower@font}{"44A}
2233   \DeclareMathSymbol{\cyrYeru}{\mathalpha}{\M@cyrilliclower@font}{"44B}
2234   \DeclareMathSymbol{\cyrsoft}{\mathalpha}{\M@cyrilliclower@font}{"44C}
2235   \DeclareMathSymbol{\cyre}{\mathalpha}{\M@cyrilliclower@font}{"44D}
2236   \DeclareMathSymbol{\cyrYu}{\mathalpha}{\M@cyrilliclower@font}{"44E}
2237   \DeclareMathSymbol{\cyrYa}{\mathalpha}{\M@cyrilliclower@font}{"44F}
2238   \DeclareMathSymbol{\cyrvarI}{\mathalpha}{\M@cyrilliclower@font}{"439}}

```

Set Hebrew characters.

```

2239 \def\M@hebrew@set{%
2240   \edef\M@hebrew@font{M\M@hebrew@shape\@tempa}
2241   \DeclareMathSymbol{\aleph}{\mathalpha}{\M@hebrew@font}{"5D0}
2242   \DeclareMathSymbol{\beth}{\mathalpha}{\M@hebrew@font}{"5D1}
2243   \DeclareMathSymbol{\gimel}{\mathalpha}{\M@hebrew@font}{"5D2}

```

```

2244 \DeclareMathSymbol{\daleth}{\mathalpha}{\M@hebrew@font}{"5D3}
2245 \DeclareMathSymbol{\he}{\mathalpha}{\M@hebrew@font}{"5D4}
2246 \DeclareMathSymbol{\vav}{\mathalpha}{\M@hebrew@font}{"5D5}
2247 \DeclareMathSymbol{\zayin}{\mathalpha}{\M@hebrew@font}{"5D6}
2248 \DeclareMathSymbol{\het}{\mathalpha}{\M@hebrew@font}{"5D7}
2249 \DeclareMathSymbol{\tet}{\mathalpha}{\M@hebrew@font}{"5D8}
2250 \DeclareMathSymbol{\yod}{\mathalpha}{\M@hebrew@font}{"5D9}
2251 \DeclareMathSymbol{\kaf}{\mathalpha}{\M@hebrew@font}{"5DB}
2252 \DeclareMathSymbol{\lamed}{\mathalpha}{\M@hebrew@font}{"5DC}
2253 \DeclareMathSymbol{\mem}{\mathalpha}{\M@hebrew@font}{"5DE}
2254 \DeclareMathSymbol{\nun}{\mathalpha}{\M@hebrew@font}{"5E0}
2255 \DeclareMathSymbol{\samekh}{\mathalpha}{\M@hebrew@font}{"5E1}
2256 \DeclareMathSymbol{\ayin}{\mathalpha}{\M@hebrew@font}{"5E2}
2257 \DeclareMathSymbol{\pe}{\mathalpha}{\M@hebrew@font}{"5E4}
2258 \DeclareMathSymbol{\tsadi}{\mathalpha}{\M@hebrew@font}{"5E6}
2259 \DeclareMathSymbol{\qof}{\mathalpha}{\M@hebrew@font}{"5E7}
2260 \DeclareMathSymbol{\resh}{\mathalpha}{\M@hebrew@font}{"5E8}
2261 \DeclareMathSymbol{\shin}{\mathalpha}{\M@hebrew@font}{"5E9}
2262 \DeclareMathSymbol{\tav}{\mathalpha}{\M@hebrew@font}{"5EA}
2263 \DeclareMathSymbol{\varkaf}{\mathalpha}{\M@hebrew@font}{"5DA}
2264 \DeclareMathSymbol{\varem}{\mathalpha}{\M@hebrew@font}{"5DD}
2265 \DeclareMathSymbol{\varnun}{\mathalpha}{\M@hebrew@font}{"5DF}
2266 \DeclareMathSymbol{\varpe}{\mathalpha}{\M@hebrew@font}{"5E3}
2267 \DeclareMathSymbol{\vartsadi}{\mathalpha}{\M@hebrew@font}{"5E5}}

```

Set digits.

```

2268 \def\M@digits@set{%
2269 \edef\M@digits@font{M\M@digitsshape\@tempa}
2270 \DeclareMathSymbol{0}{\mathalpha}{\M@digits@font}{`0}
2271 \DeclareMathSymbol{1}{\mathalpha}{\M@digits@font}{`1}
2272 \DeclareMathSymbol{2}{\mathalpha}{\M@digits@font}{`2}
2273 \DeclareMathSymbol{3}{\mathalpha}{\M@digits@font}{`3}
2274 \DeclareMathSymbol{4}{\mathalpha}{\M@digits@font}{`4}
2275 \DeclareMathSymbol{5}{\mathalpha}{\M@digits@font}{`5}
2276 \DeclareMathSymbol{6}{\mathalpha}{\M@digits@font}{`6}
2277 \DeclareMathSymbol{7}{\mathalpha}{\M@digits@font}{`7}
2278 \DeclareMathSymbol{8}{\mathalpha}{\M@digits@font}{`8}
2279 \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_EX 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.

```

2280 \def\M@operator@set{%

```



```

2281 \ifM@adjust@font
2282 \edef\M@operator@num{\number\csname symM\M@operatorshape\@tempa\endcsname}
2283 \protected\edef\M@operator@mathcodes{%
2284   \Umathcode`A=7+\M@operator@num+`A\relax
2285   \Umathcode`B=7+\M@operator@num+`B\relax
2286   \Umathcode`C=7+\M@operator@num+`C\relax
2287   \Umathcode`D=7+\M@operator@num+`D\relax
2288   \Umathcode`E=7+\M@operator@num+`E\relax
2289   \Umathcode`F=7+\M@operator@num+`F\relax
2290   \Umathcode`G=7+\M@operator@num+`G\relax
2291   \Umathcode`H=7+\M@operator@num+`H\relax
2292   \Umathcode`I=7+\M@operator@num+`I\relax
2293   \Umathcode`J=7+\M@operator@num+`J\relax
2294   \Umathcode`K=7+\M@operator@num+`K\relax
2295   \Umathcode`L=7+\M@operator@num+`L\relax
2296   \Umathcode`M=7+\M@operator@num+`M\relax
2297   \Umathcode`N=7+\M@operator@num+`N\relax
2298   \Umathcode`O=7+\M@operator@num+`O\relax
2299   \Umathcode`P=7+\M@operator@num+`P\relax
2300   \Umathcode`Q=7+\M@operator@num+`Q\relax
2301   \Umathcode`R=7+\M@operator@num+`R\relax
2302   \Umathcode`S=7+\M@operator@num+`S\relax
2303   \Umathcode`T=7+\M@operator@num+`T\relax
2304   \Umathcode`U=7+\M@operator@num+`U\relax
2305   \Umathcode`V=7+\M@operator@num+`V\relax
2306   \Umathcode`W=7+\M@operator@num+`W\relax
2307   \Umathcode`X=7+\M@operator@num+`X\relax
2308   \Umathcode`Y=7+\M@operator@num+`Y\relax
2309   \Umathcode`Z=7+\M@operator@num+`Z\relax
2310   \Umathcode`a=7+\M@operator@num+`a\relax
2311   \Umathcode`b=7+\M@operator@num+`b\relax
2312   \Umathcode`c=7+\M@operator@num+`c\relax
2313   \Umathcode`d=7+\M@operator@num+`d\relax
2314   \Umathcode`e=7+\M@operator@num+`e\relax
2315   \Umathcode`f=7+\M@operator@num+`f\relax
2316   \Umathcode`g=7+\M@operator@num+`g\relax
2317   \Umathcode`h=7+\M@operator@num+`h\relax
2318   \Umathcode`i=7+\M@operator@num+`i\relax
2319   \Umathcode`j=7+\M@operator@num+`j\relax
2320   \Umathcode`k=7+\M@operator@num+`k\relax
2321   \Umathcode`l=7+\M@operator@num+`l\relax
2322   \Umathcode`m=7+\M@operator@num+`m\relax
2323   \Umathcode`n=7+\M@operator@num+`n\relax
2324   \Umathcode`o=7+\M@operator@num+`o\relax
2325   \Umathcode`p=7+\M@operator@num+`p\relax
2326   \Umathcode`q=7+\M@operator@num+`q\relax
2327   \Umathcode`r=7+\M@operator@num+`r\relax

```

```

2328   \Umathcode`s=7+\M@operator@num+`s\relax
2329   \Umathcode`t=7+\M@operator@num+`t\relax
2330   \Umathcode`u=7+\M@operator@num+`u\relax
2331   \Umathcode`v=7+\M@operator@num+`v\relax
2332   \Umathcode`w=7+\M@operator@num+`w\relax
2333   \Umathcode`x=7+\M@operator@num+`x\relax
2334   \Umathcode`y=7+\M@operator@num+`y\relax
2335   \Umathcode`z=7+\M@operator@num+`z\relax
2336   \Umathchardef\imath=7+\M@operator@num+1044506\relax
2337   \Umathchardef\jmath=7+\M@operator@num+1044500\relax}
2338 \else
2339   \let\M@operator@mathcodes\@empty
2340 \fi

```

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

```

2341 \xdef\operator@font{\noexpand\mathgroup
2342   \csname symM\M@operator@shape\@tempa\endcsname\M@operator@mathcodes}}

```

Set delimiters.

```

2343 \ifM@adjust@font
2344   \def\M@delimiters@set{%
2345     \edef\M@delimiters@font{M\M@delimiters@shape\@tempa}
2346     \DeclareMathSymbol{(\mathopen}{\M@delimiters@font}{"28}
2347     \DeclareMathSymbol{)}{\mathclose}{\M@delimiters@font}{"29}
2348     \DeclareMathSymbol{[}{\mathopen}{\M@delimiters@font}{"5B}
2349     \DeclareMathSymbol{]}\mathclose}{\M@delimiters@font}{"5D}
2350     \ifM@symbols\else
2351       \DeclareMathSymbol{|}{\mathord}{\M@delimiters@font}{"7C}
2352     \fi
2353     \DeclareMathSymbol{\leftbrace}{\mathopen}{\M@delimiters@font}{"7B}
2354     \DeclareMathSymbol{\rightbrace}{\mathclose}{\M@delimiters@font}{"7D}
2355     \global\Udelcode40+=\csname sym\M@delimiters@font\endcsname+40\relax % (
2356     \global\Udelcode41+=\csname sym\M@delimiters@font\endcsname+41\relax % )
2357     \global\Udelcode47+=\csname sym\M@delimiters@font\endcsname+47\relax % /
2358     \global\Udelcode91+=\csname sym\M@delimiters@font\endcsname+91\relax % [
2359     \global\Udelcode93+=\csname sym\M@delimiters@font\endcsname+93\relax % ]
2360     \global\Udelcode124+=\csname sym\M@delimiters@font\endcsname+124\relax % |
2361     \global\let\vert=|
2362     \protected\gdef\backslash{\ifmmode\mathbackslash\else\textbackslash\fi}
2363     \protected\xdef\mathbackslash{%
2364       \Udelimiter+2+\number\csname sym\M@delimiters@font\endcsname
2365       +92\relax} % backslash
2366     \protected\xdef\lbrace{%
2367       \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname
2368       +123\relax} % {
2369     \protected\xdef\rbrace{%
2370       \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
2371       +125\relax} % }
2372     \protected\xdef\lguil{%

```

```

2373     \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname
2374     +8249\relax} % single left guilement
2375 \protected\xdef\rguil{%
2376     \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
2377     +8250\relax} % single right guilement
2378 \protected\xdef\llguil{%
2379     \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname
2380     +171\relax} % double left guilement
2381 \protected\xdef\rrguil{%
2382     \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
2383     +187\relax} % double right guilement
2384 \protected\xdef\fakelangle{%
2385     \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname
2386     +1044508\relax} % fake left angle
2387 \protected\xdef\fakerangle{%
2388     \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
2389     +1044509\relax} % fake right angle
2390 \protected\xdef\fakellangle{%
2391     \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname
2392     +1044510\relax} % fake double left angle
2393 \protected\xdef\fakerrangle{%
2394     \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
2395     +1044511\relax} % fake double right angle
2396 }
2397 \else
2398 \def\M@delimiters@set{%
2399 \edef\M@delimiters@font{M\M@delimitersshape\@tempa}
2400 \DeclareMathSymbol{\}\{\mathopen}\{\M@delimiters@font}\{"28}
2401 \DeclareMathSymbol{\}\}\{\mathclose}\{\M@delimiters@font}\{"29}
2402 \DeclareMathSymbol{\[}\{\mathopen}\{\M@delimiters@font}\{"5B}
2403 \DeclareMathSymbol{\]}\{\mathclose}\{\M@delimiters@font}\{"5D}
2404 \DeclareMathSymbol{\lguil}\{\mathopen}\{\M@delimiters@font}\{"2039}
2405 \DeclareMathSymbol{\rguil}\{\mathclose}\{\M@delimiters@font}\{"203A}
2406 \DeclareMathSymbol{\llguil}\{\mathopen}\{\M@delimiters@font}\{"AB}
2407 \DeclareMathSymbol{\rrguil}\{\mathclose}\{\M@delimiters@font}\{"BB}
2408 \DeclareMathSymbol{\leftbrace}\{\mathopen}\{\M@delimiters@font}\{"7B}
2409 \DeclareMathSymbol{\rightbrace}\{\mathclose}\{\M@delimiters@font}\{"7D}}
2410 \fi
Radicals.
2411 \ifM@adjust@font
2412 \def\M@radical@set{%
2413 \edef\M@radical@font{M\M@radicalshape\@tempa}
2414 \DeclareMathSymbol{\surd}\{\mathord}\{\M@radical@font}\{"221A}
2415 \xdef\@sqrts@gn##1{%
2416 \Uradical+\number\csname sym\M@radical@font\endcsname+8730\relax{##1}}

```

We redefine `\r@ot`, 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.

```

2417 \gdef\r@t##1##2{%
2418   \setbox\z@\hbox{\m@th##1\sqrtsign{##2}$}%
2419   \setbox\surdbox\hbox{\m@th##1\@sqrts@gn{%
2420     \hbox{\vphantom{\m@th##1##2$}}}$}
2421   \dimen@ht\surdbox
2422   \advance\dimen@\dp\surdbox
2423   \dimen@=0.6\dimen@
2424   \advance\dimen@-\dp\surdbox
2425   \ifdim\wd\rootbox<0.6\wd\surdbox
2426     \kern0.6\wd\surdbox
2427   \else
2428     \kern\wd\rootbox
2429   \fi
2430   \raise\dimen@\hbox{\llap{\copy\rootbox}}
2431   \kern-0.6\wd\surdbox
2432   \box\z@}
2433 \gdef\sqrtsign##1{\@sqrts@gn{\mkern\radicandoffset##1}}
2434 \else
2435 \def\M@radical@set{%
2436   \edef\M@radical@font{M\M@radicalshape\@tempa}
2437   \DeclareMathSymbol{\surd}{\mathord}{\M@radical@font}{"221A}}
2438 \fi

```

Big operators.

```

2439 \def\M@bigops@set{%
2440   \edef\M@bigops@font{M\M@bigopsshape\@tempa}
2441   \let\sum\@undefined
2442   \let\prod\@undefined
2443   \DeclareMathSymbol{\sum}{\mathop}{\M@bigops@font}{"2211}
2444   \DeclareMathSymbol{\prod}{\mathop}{\M@bigops@font}{"220F}
2445   \DeclareMathSymbol{\intop}{\mathop}{\M@bigops@font}{"222B}}

```

Extended big operators.

```

2446 \def\M@extbigops@set{%
2447   \edef\M@extbigops@font{M\M@extbigopsshape\@tempa}
2448   \let\coprod\@undefined
2449   \let\bigvee\@undefined
2450   \let\bigwedge\@undefined
2451   \let\bigcup\@undefined
2452   \let\bigcap\@undefined
2453   \let\bigoplus\@undefined
2454   \let\bigotimes\@undefined
2455   \let\bigodot\@undefined
2456   \let\bigsqcup\@undefined
2457   \DeclareMathSymbol{\coprod}{\mathop}{\M@extbigops@font}{"2210}
2458   \DeclareMathSymbol{\bigvee}{\mathop}{\M@extbigops@font}{"22C1}
2459   \DeclareMathSymbol{\bigwedge}{\mathop}{\M@extbigops@font}{"22C0}

```

```

2460 \DeclareMathSymbol{\bigcup}{\mathop}{\M@extbigops@font}{"22C3}
2461 \DeclareMathSymbol{\bigcap}{\mathop}{\M@extbigops@font}{"22C2}
2462 \DeclareMathSymbol{\iintop}{\mathop}{\M@extbigops@font}{"222C}
2463 \protected\gdef\iint{\iintop\nolimits}
2464 \DeclareMathSymbol{\iiintop}{\mathop}{\M@extbigops@font}{"222D}
2465 \protected\gdef\iiint{\iiintop\nolimits}
2466 \DeclareMathSymbol{\ointop}{\mathop}{\M@extbigops@font}{"222E}
2467 \protected\gdef\oint{\ointop\nolimits}
2468 \DeclareMathSymbol{\oiintop}{\mathop}{\M@extbigops@font}{"222F}
2469 \protected\gdef\oiint{\oiintop\nolimits}
2470 \DeclareMathSymbol{\oiiintop}{\mathop}{\M@extbigops@font}{"2230}
2471 \protected\gdef\oiiint{\oiiintop\nolimits}
2472 \DeclareMathSymbol{\bigoplus}{\mathop}{\M@extbigops@font}{"2A01}
2473 \DeclareMathSymbol{\bigotimes}{\mathop}{\M@extbigops@font}{"2A02}
2474 \DeclareMathSymbol{\bigodot}{\mathop}{\M@extbigops@font}{"2A00}
2475 \DeclareMathSymbol{\bigsqcap}{\mathop}{\M@extbigops@font}{"2A05}
2476 \DeclareMathSymbol{\bigsqcup}{\mathop}{\M@extbigops@font}{"2A06}}

```

Set symbols.

```

2477 \def\M@symbols@set{%
2478 \edef\M@symbols@font{M\M@symbols@shape\@tempa}
2479 \let\colon\@undefined
2480 \let\mathellipsis\@undefined
2481 \DeclareMathSymbol{.}{\mathord}{\M@symbols@font}{"2E}
2482 \DeclareMathSymbol{@}{\mathord}{\M@symbols@font}{"40}
2483 \DeclareMathSymbol{'}{\mathord}{\M@symbols@font}{"2032}
2484 \DeclareMathSymbol{\prime}{\mathord}{\M@symbols@font}{"2032}
2485 \DeclareMathSymbol{"}{\mathord}{\M@symbols@font}{"2033}
2486 \DeclareMathSymbol{\mathhash}{\mathord}{\M@symbols@font}{"23}
2487 \DeclareMathSymbol{\mathdollar}{\mathord}{\M@symbols@font}{"24}
2488 \DeclareMathSymbol{\mathpercent}{\mathord}{\M@symbols@font}{"25}
2489 \DeclareMathSymbol{\mathand}{\mathord}{\M@symbols@font}{"26}
2490 \DeclareMathSymbol{\mathparagraph}{\mathord}{\M@symbols@font}{"B6}
2491 \DeclareMathSymbol{\mathsection}{\mathord}{\M@symbols@font}{"A7}
2492 \DeclareMathSymbol{\mathsterling}{\mathord}{\M@symbols@font}{"A3}
2493 \DeclareMathSymbol{\neg}{\mathord}{\M@symbols@font}{"AC}
2494 \DeclareMathSymbol{|}{\mathord}{\M@symbols@font}{"7C}
2495 \DeclareMathSymbol{\infty}{\mathord}{\M@symbols@font}{"221E}
2496 \DeclareMathSymbol{\partial}{\mathord}{\M@symbols@font}{"2202}
2497 \DeclareMathSymbol{\degree}{\mathord}{\M@symbols@font}{"B0}
2498 \DeclareMathSymbol{\increment}{\mathord}{\M@symbols@font}{"2206}
2499 \DeclareMathSymbol{\comma}{\mathord}{\M@symbols@font}{"2C}
2500 \DeclareMathSymbol{+}{\mathbin}{\M@symbols@font}{"2B}
2501 \DeclareMathSymbol{-}{\mathbin}{\M@symbols@font}{"2212}
2502 \DeclareMathSymbol{*}{\mathbin}{\M@symbols@font}{"2A}
2503 \DeclareMathSymbol{\times}{\mathbin}{\M@symbols@font}{"D7}
2504 \DeclareMathSymbol{/}{\mathbin}{\M@symbols@font}{"2F}
2505 \DeclareMathSymbol{\fractionslash}{\mathbin}{\M@symbols@font}{"2215}}

```

```

2506 \DeclareMathSymbol{\div}\mathbin{\M@symbols@font}{F7}
2507 \DeclareMathSymbol{\pm}\mathbin{\M@symbols@font}{B1}
2508 \DeclareMathSymbol{\bullet}\mathbin{\M@symbols@font}{2022}
2509 \DeclareMathSymbol{\dagger}\mathbin{\M@symbols@font}{2020}
2510 \DeclareMathSymbol{\ddagger}\mathbin{\M@symbols@font}{2021}
2511 \DeclareMathSymbol{\cdot}\mathbin{\M@symbols@font}{2219}
2512 \DeclareMathSymbol{\setminus}\mathbin{\M@symbols@font}{5C}
2513 \DeclareMathSymbol{=}\mathrel{\M@symbols@font}{3D}
2514 \DeclareMathSymbol{<}\mathrel{\M@symbols@font}{3C}
2515 \DeclareMathSymbol{>}\mathrel{\M@symbols@font}{3E}
2516 \DeclareMathSymbol{\leq}\mathrel{\M@symbols@font}{2264}
2517 \DeclareMathSymbol{\geq}\mathrel{\M@symbols@font}{2265}
2518 \DeclareMathSymbol{\sim}\mathrel{\M@symbols@font}{7E}
2519 \DeclareMathSymbol{\approx}\mathrel{\M@symbols@font}{2248}
2520 \DeclareMathSymbol{\equiv}\mathrel{\M@symbols@font}{2261}
2521 \DeclareMathSymbol{\mid}\mathrel{\M@symbols@font}{7C}
2522 \DeclareMathSymbol{\parallel}\mathrel{\M@symbols@font}{2016}
2523 \DeclareMathSymbol{:}\mathrel{\M@symbols@font}{3A}
2524 \DeclareMathSymbol{?}\mathclose{\M@symbols@font}{3F}
2525 \DeclareMathSymbol{!}\mathclose{\M@symbols@font}{21}
2526 \DeclareMathSymbol{,}\mathpunct{\M@symbols@font}{2C}
2527 \DeclareMathSymbol{;}\mathpunct{\M@symbols@font}{3B}
2528 \DeclareMathSymbol{\colon}\mathord{\M@symbols@font}{3A}
2529 \DeclareMathSymbol{\mathellipsis}\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.

```

2530 \protected\gdef\#{\ifmmode\mathhash\else\char"23\relax\fi}
2531 \protected\gdef\%{\ifmmode\mathpercent\else\char"25\relax\fi}
2532 \protected\gdef\&{\ifmmode\mathand\else\char"26\relax\fi}
2533 \DeclareMathSymbol{\@relbar}\mathbin{symbols}{00}
2534 \DeclareMathSymbol{\@Relbar}\mathrel{operators}{3D}
2535 \DeclareMathSymbol{\@verticalbar}\mathord{symbols}{6A}
2536 \ifM@extsymbols\else
2537   \protected\gdef\simeq{\mathrel{\mathpalette\stack@flatrel{-}\sim}}
2538   \protected\gdef\cong{\mathrel{\mathpalette\stack@flatrel{=}\sim}}
2539 \fi
2540 \protected\gdef\relbar{\mathrel{\smash{@relbar}}}
2541 \protected\gdef\Relbar{\mathrel{\@Relbar}}
2542 \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`.

```

2543 \ifM@adjust@font

```

```

2544 \DeclareMathSymbol{\bigat}{\mathop}{\M@symbols@font}{"40}
2545 \DeclareMathSymbol{\bighash}{\mathop}{\M@symbols@font}{"23}
2546 \DeclareMathSymbol{\bigdollar}{\mathop}{\M@symbols@font}{"24}
2547 \DeclareMathSymbol{\bigpercent}{\mathop}{\M@symbols@font}{"25}
2548 \DeclareMathSymbol{\bigand}{\mathop}{\M@symbols@font}{"26}
2549 \DeclareMathSymbol{\bigplus}{\mathop}{\M@symbols@font}{"2B}
2550 \DeclareMathSymbol{\bigp}{\mathop}{\M@symbols@font}{"21}
2551 \DeclareMathSymbol{\bigq}{\mathop}{\M@symbols@font}{"3F}
2552 \DeclareMathSymbol{\bigS}{\mathop}{\M@symbols@font}{"A7}
2553 \DeclareMathSymbol{\bigtimes}{\mathop}{\M@symbols@font}{"D7}
2554 \DeclareMathSymbol{\bigdiv}{\mathop}{\M@symbols@font}{"F7}

```

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

```

2555 \DeclareMathSymbol{\nabla}{\mathord}{\M@symbols@font}{"2207}
2556 \fi}

```

Set extended symbols.

```

2557 \def\M@extsymbols@set{%
2558 \edef\M@extsymbols@font{M\M@extsymbolsshape\@tempa}
2559 \let\angle\@undefined
2560 \let\simeq\@undefined
2561 \let\sqsubset\@undefined
2562 \let\sqsupset\@undefined
2563 \let\bowtie\@undefined
2564 \let\doteq\@undefined
2565 \let\neq\@undefined
2566 \DeclareMathSymbol{\wp}{\mathord}{\M@extsymbols@font}{"2118}
2567 \DeclareMathSymbol{\Re}{\mathord}{\M@extsymbols@font}{"211C}
2568 \DeclareMathSymbol{\Im}{\mathord}{\M@extsymbols@font}{"2111}
2569 \DeclareMathSymbol{\ell}{\mathord}{\M@extsymbols@font}{"2113}
2570 \DeclareMathSymbol{\forall}{\mathord}{\M@extsymbols@font}{"2200}
2571 \DeclareMathSymbol{\exists}{\mathord}{\M@extsymbols@font}{"2203}
2572 \DeclareMathSymbol{\emptyset}{\mathord}{\M@extsymbols@font}{"2205}
2573 \DeclareMathSymbol{\in}{\mathord}{\M@extsymbols@font}{"2208}
2574 \DeclareMathSymbol{\ni}{\mathord}{\M@extsymbols@font}{"220B}
2575 \DeclareMathSymbol{\mp}{\mathord}{\M@extsymbols@font}{"2213}
2576 \DeclareMathSymbol{\angle}{\mathord}{\M@extsymbols@font}{"2220}
2577 \DeclareMathSymbol{\top}{\mathord}{\M@extsymbols@font}{"22A4}
2578 \DeclareMathSymbol{\bot}{\mathord}{\M@extsymbols@font}{"22A5}
2579 \DeclareMathSymbol{\vdash}{\mathord}{\M@extsymbols@font}{"22A2}
2580 \DeclareMathSymbol{\dashv}{\mathord}{\M@extsymbols@font}{"22A3}
2581 \DeclareMathSymbol{\flat}{\mathord}{\M@extsymbols@font}{"266D}
2582 \DeclareMathSymbol{\natural}{\mathord}{\M@extsymbols@font}{"266E}
2583 \DeclareMathSymbol{\sharp}{\mathord}{\M@extsymbols@font}{"266F}
2584 \DeclareMathSymbol{\fflat}{\mathord}{\M@extsymbols@font}{"1D12B}
2585 \DeclareMathSymbol{\ssharp}{\mathord}{\M@extsymbols@font}{"1D12A}
2586 \DeclareMathSymbol{\bclubsuit}{\mathord}{\M@extsymbols@font}{"2663}
2587 \DeclareMathSymbol{\bdiamondsuit}{\mathord}{\M@extsymbols@font}{"2666}
2588 \DeclareMathSymbol{\bheartsuit}{\mathord}{\M@extsymbols@font}{"2665}

```

```

2589 \DeclareMathSymbol{\bspadesuit}{\mathord}{\M@extsymbols@font}{"2660}
2590 \DeclareMathSymbol{\wclubsuit}{\mathord}{\M@extsymbols@font}{"2667}
2591 \DeclareMathSymbol{\wdiamondsuit}{\mathord}{\M@extsymbols@font}{"2662}
2592 \DeclareMathSymbol{\wheartsuit}{\mathord}{\M@extsymbols@font}{"2661}
2593 \DeclareMathSymbol{\wspadesuit}{\mathord}{\M@extsymbols@font}{"2664}
2594 \global\let\spadesuit\bspadesuit
2595 \global\let\heartsuit\wheartsuit
2596 \global\let\diamondsuit\wdiamondsuit
2597 \global\let\clubsuit\bclubsuit
2598 \DeclareMathSymbol{\wedge}{\mathbin}{\M@extsymbols@font}{"2227}
2599 \DeclareMathSymbol{\vee}{\mathbin}{\M@extsymbols@font}{"2228}
2600 \DeclareMathSymbol{\cap}{\mathord}{\M@extsymbols@font}{"2229}
2601 \DeclareMathSymbol{\cup}{\mathbin}{\M@extsymbols@font}{"222A}
2602 \DeclareMathSymbol{\sqcap}{\mathbin}{\M@extsymbols@font}{"2293}
2603 \DeclareMathSymbol{\sqcup}{\mathbin}{\M@extsymbols@font}{"2294}
2604 \DeclareMathSymbol{\amalg}{\mathbin}{\M@extsymbols@font}{"2A3F}
2605 \DeclareMathSymbol{\wr}{\mathbin}{\M@extsymbols@font}{"2240}
2606 \DeclareMathSymbol{\ast}{\mathbin}{\M@extsymbols@font}{"2217}
2607 \DeclareMathSymbol{\star}{\mathbin}{\M@extsymbols@font}{"22C6}
2608 \DeclareMathSymbol{\diamond}{\mathbin}{\M@extsymbols@font}{"22C4}
2609 \DeclareMathSymbol{\varcdot}{\mathbin}{\M@extsymbols@font}{"22C5}
2610 \DeclareMathSymbol{\varsetminus}{\mathbin}{\M@extsymbols@font}{"2216}
2611 \DeclareMathSymbol{\oplus}{\mathbin}{\M@extsymbols@font}{"2295}
2612 \DeclareMathSymbol{\otimes}{\mathbin}{\M@extsymbols@font}{"2297}
2613 \DeclareMathSymbol{\ominus}{\mathbin}{\M@extsymbols@font}{"2296}
2614 \DeclareMathSymbol{\odiv}{\mathbin}{\M@extsymbols@font}{"2A38}
2615 \DeclareMathSymbol{\oslash}{\mathbin}{\M@extsymbols@font}{"2298}
2616 \DeclareMathSymbol{\odot}{\mathbin}{\M@extsymbols@font}{"2299}
2617 \DeclareMathSymbol{\sqplus}{\mathbin}{\M@extsymbols@font}{"229E}
2618 \DeclareMathSymbol{\sqtimes}{\mathbin}{\M@extsymbols@font}{"22A0}
2619 \DeclareMathSymbol{\sqminus}{\mathbin}{\M@extsymbols@font}{"229F}
2620 \DeclareMathSymbol{\sqdot}{\mathbin}{\M@extsymbols@font}{"22A1}
2621 \DeclareMathSymbol{\in}{\mathrel}{\M@extsymbols@font}{"2208}
2622 \DeclareMathSymbol{\ni}{\mathrel}{\M@extsymbols@font}{"220B}
2623 \DeclareMathSymbol{\subset}{\mathrel}{\M@extsymbols@font}{"2282}
2624 \DeclareMathSymbol{\supset}{\mathrel}{\M@extsymbols@font}{"2283}
2625 \DeclareMathSymbol{\subseteq}{\mathrel}{\M@extsymbols@font}{"2286}
2626 \DeclareMathSymbol{\supseteq}{\mathrel}{\M@extsymbols@font}{"2287}
2627 \DeclareMathSymbol{\sqsubset}{\mathrel}{\M@extsymbols@font}{"228F}
2628 \DeclareMathSymbol{\sqsupset}{\mathrel}{\M@extsymbols@font}{"2290}
2629 \DeclareMathSymbol{\sqsubseteq}{\mathrel}{\M@extsymbols@font}{"2291}
2630 \DeclareMathSymbol{\sqsupseteq}{\mathrel}{\M@extsymbols@font}{"2292}
2631 \DeclareMathSymbol{\triangleleft}{\mathrel}{\M@extsymbols@font}{"22B2}
2632 \DeclareMathSymbol{\triangleright}{\mathrel}{\M@extsymbols@font}{"22B3}
2633 \DeclareMathSymbol{\trianglelefteq}{\mathrel}{\M@extsymbols@font}{"22B4}
2634 \DeclareMathSymbol{\trianglerighteq}{\mathrel}{\M@extsymbols@font}{"22B5}
2635 \DeclareMathSymbol{\propto}{\mathrel}{\M@extsymbols@font}{"221D}

```



```

2636 \DeclareMathSymbol{\bowtie}{\mathrel}{\M@extsymbols@font}{"22C8}
2637 \DeclareMathSymbol{\hourglass}{\mathrel}{\M@extsymbols@font}{"29D6}
2638 \DeclareMathSymbol{\therefore}{\mathrel}{\M@extsymbols@font}{"2234}
2639 \DeclareMathSymbol{\because}{\mathrel}{\M@extsymbols@font}{"2235}
2640 \DeclareMathSymbol{\ratio}{\mathrel}{\M@extsymbols@font}{"2236}
2641 \DeclareMathSymbol{\proportion}{\mathrel}{\M@extsymbols@font}{"2237}
2642 \DeclareMathSymbol{\ll}{\mathrel}{\M@extsymbols@font}{"226A}
2643 \DeclareMathSymbol{\gg}{\mathrel}{\M@extsymbols@font}{"226B}
2644 \DeclareMathSymbol{\lll}{\mathrel}{\M@extsymbols@font}{"22D8}
2645 \DeclareMathSymbol{\ggg}{\mathrel}{\M@extsymbols@font}{"22D9}
2646 \DeclareMathSymbol{\leqq}{\mathrel}{\M@extsymbols@font}{"2266}
2647 \DeclareMathSymbol{\geqq}{\mathrel}{\M@extsymbols@font}{"2267}
2648 \DeclareMathSymbol{\lapprox}{\mathrel}{\M@extsymbols@font}{"2A85}
2649 \DeclareMathSymbol{\gapprox}{\mathrel}{\M@extsymbols@font}{"2A86}
2650 \DeclareMathSymbol{\simeq}{\mathrel}{\M@extsymbols@font}{"2243}
2651 \DeclareMathSymbol{\eqsim}{\mathrel}{\M@extsymbols@font}{"2242}
2652 \DeclareMathSymbol{\simeqq}{\mathrel}{\M@extsymbols@font}{"2245}
2653 \global\let\cong\simeqq
2654 \DeclareMathSymbol{\approxeq}{\mathrel}{\M@extsymbols@font}{"224A}
2655 \DeclareMathSymbol{\sssim}{\mathrel}{\M@extsymbols@font}{"224B}
2656 \DeclareMathSymbol{\seq}{\mathrel}{\M@extsymbols@font}{"224C}
2657 \DeclareMathSymbol{\doteq}{\mathrel}{\M@extsymbols@font}{"2250}
2658 \DeclareMathSymbol{\coloneq}{\mathrel}{\M@extsymbols@font}{"2254}
2659 \DeclareMathSymbol{\eqcolon}{\mathrel}{\M@extsymbols@font}{"2255}
2660 \DeclareMathSymbol{\ringeq}{\mathrel}{\M@extsymbols@font}{"2257}
2661 \DeclareMathSymbol{\arceq}{\mathrel}{\M@extsymbols@font}{"2258}
2662 \DeclareMathSymbol{\wedgreek}{\mathrel}{\M@extsymbols@font}{"2259}
2663 \DeclareMathSymbol{\veeeq}{\mathrel}{\M@extsymbols@font}{"225A}
2664 \DeclareMathSymbol{\stareq}{\mathrel}{\M@extsymbols@font}{"225B}
2665 \DeclareMathSymbol{\triangleeq}{\mathrel}{\M@extsymbols@font}{"225C}
2666 \DeclareMathSymbol{\defeq}{\mathrel}{\M@extsymbols@font}{"225D}
2667 \DeclareMathSymbol{\qeq}{\mathrel}{\M@extsymbols@font}{"225F}
2668 \DeclareMathSymbol{\lsim}{\mathrel}{\M@extsymbols@font}{"2272}
2669 \DeclareMathSymbol{\gsim}{\mathrel}{\M@extsymbols@font}{"2273}
2670 \DeclareMathSymbol{\prec}{\mathrel}{\M@extsymbols@font}{"227A}
2671 \DeclareMathSymbol{\succ}{\mathrel}{\M@extsymbols@font}{"227B}
2672 \DeclareMathSymbol{\preceq}{\mathrel}{\M@extsymbols@font}{"227C}
2673 \DeclareMathSymbol{\succeq}{\mathrel}{\M@extsymbols@font}{"227D}
2674 \DeclareMathSymbol{\preceqq}{\mathrel}{\M@extsymbols@font}{"2AB3}
2675 \DeclareMathSymbol{\succeqq}{\mathrel}{\M@extsymbols@font}{"2AB4}
2676 \DeclareMathSymbol{\precsim}{\mathrel}{\M@extsymbols@font}{"227E}
2677 \DeclareMathSymbol{\succsim}{\mathrel}{\M@extsymbols@font}{"227F}
2678 \DeclareMathSymbol{\precapprox}{\mathrel}{\M@extsymbols@font}{"2AB7}
2679 \DeclareMathSymbol{\succapprox}{\mathrel}{\M@extsymbols@font}{"2AB8}
2680 \DeclareMathSymbol{\precprec}{\mathrel}{\M@extsymbols@font}{"2ABB}
2681 \DeclareMathSymbol{\succsucc}{\mathrel}{\M@extsymbols@font}{"2ABC}
2682 \DeclareMathSymbol{\asymp}{\mathrel}{\M@extsymbols@font}{"224D}

```

```

2683 \DeclareMathSymbol{\nin}{\mathrel}{\M@extsymbols@font}{"2209}
2684 \DeclareMathSymbol{\nni}{\mathrel}{\M@extsymbols@font}{"220C}
2685 \DeclareMathSymbol{\nsubset}{\mathrel}{\M@extsymbols@font}{"2284}
2686 \DeclareMathSymbol{\nsupset}{\mathrel}{\M@extsymbols@font}{"2285}
2687 \DeclareMathSymbol{\nsubseteq}{\mathrel}{\M@extsymbols@font}{"2288}
2688 \DeclareMathSymbol{\nsupseteq}{\mathrel}{\M@extsymbols@font}{"2289}
2689 \DeclareMathSymbol{\subsetneq}{\mathrel}{\M@extsymbols@font}{"228A}
2690 \DeclareMathSymbol{\supsetneq}{\mathrel}{\M@extsymbols@font}{"228B}
2691 \DeclareMathSymbol{\nsqsubseteq}{\mathrel}{\M@extsymbols@font}{"22E2}
2692 \DeclareMathSymbol{\nsqsupseteq}{\mathrel}{\M@extsymbols@font}{"22E3}
2693 \DeclareMathSymbol{\sqsubseteq}{\mathrel}{\M@extsymbols@font}{"22E4}
2694 \DeclareMathSymbol{\sqsupseteq}{\mathrel}{\M@extsymbols@font}{"22E5}
2695 \DeclareMathSymbol{\neq}{\mathrel}{\M@extsymbols@font}{"2260}
2696 \DeclareMathSymbol{\nl}{\mathrel}{\M@extsymbols@font}{"226E}
2697 \DeclareMathSymbol{\nleq}{\mathrel}{\M@extsymbols@font}{"2270}
2698 \DeclareMathSymbol{\ngeq}{\mathrel}{\M@extsymbols@font}{"2271}
2699 \DeclareMathSymbol{\lneq}{\mathrel}{\M@extsymbols@font}{"2A87}
2700 \DeclareMathSymbol{\gneq}{\mathrel}{\M@extsymbols@font}{"2A88}
2701 \DeclareMathSymbol{\lneqq}{\mathrel}{\M@extsymbols@font}{"2268}
2702 \DeclareMathSymbol{\gneqq}{\mathrel}{\M@extsymbols@font}{"2269}
2703 \DeclareMathSymbol{\ntriangleleft}{\mathrel}{\M@extsymbols@font}{"22EA}
2704 \DeclareMathSymbol{\ntriangleright}{\mathrel}{\M@extsymbols@font}{"22EB}
2705 \DeclareMathSymbol{\ntrianglelefteq}{\mathrel}{\M@extsymbols@font}{"22EC}
2706 \DeclareMathSymbol{\ntrianglerighteq}{\mathrel}{\M@extsymbols@font}{"22ED}
2707 \DeclareMathSymbol{\nsim}{\mathrel}{\M@extsymbols@font}{"2241}
2708 \DeclareMathSymbol{\napprox}{\mathrel}{\M@extsymbols@font}{"2249}
2709 \DeclareMathSymbol{\nsimeq}{\mathrel}{\M@extsymbols@font}{"2244}
2710 \DeclareMathSymbol{\nsimeqq}{\mathrel}{\M@extsymbols@font}{"2247}
2711 \DeclareMathSymbol{\simneqq}{\mathrel}{\M@extsymbols@font}{"2246}
2712 \DeclareMathSymbol{\nlsim}{\mathrel}{\M@extsymbols@font}{"2274}
2713 \DeclareMathSymbol{\ngsim}{\mathrel}{\M@extsymbols@font}{"2275}
2714 \DeclareMathSymbol{\lnsim}{\mathrel}{\M@extsymbols@font}{"22E6}
2715 \DeclareMathSymbol{\gnsim}{\mathrel}{\M@extsymbols@font}{"22E7}
2716 \DeclareMathSymbol{\lnapprox}{\mathrel}{\M@extsymbols@font}{"2A89}
2717 \DeclareMathSymbol{\gnapprox}{\mathrel}{\M@extsymbols@font}{"2A8A}
2718 \DeclareMathSymbol{\nprec}{\mathrel}{\M@extsymbols@font}{"2280}
2719 \DeclareMathSymbol{\nsucc}{\mathrel}{\M@extsymbols@font}{"2281}
2720 \DeclareMathSymbol{\npreceq}{\mathrel}{\M@extsymbols@font}{"22E0}
2721 \DeclareMathSymbol{\nsucceq}{\mathrel}{\M@extsymbols@font}{"22E1}
2722 \DeclareMathSymbol{\precneq}{\mathrel}{\M@extsymbols@font}{"2AB1}
2723 \DeclareMathSymbol{\succneq}{\mathrel}{\M@extsymbols@font}{"2AB2}
2724 \DeclareMathSymbol{\precneqq}{\mathrel}{\M@extsymbols@font}{"2AB5}
2725 \DeclareMathSymbol{\succneqq}{\mathrel}{\M@extsymbols@font}{"2AB6}
2726 \DeclareMathSymbol{\precnsim}{\mathrel}{\M@extsymbols@font}{"22E8}
2727 \DeclareMathSymbol{\succnsim}{\mathrel}{\M@extsymbols@font}{"22E9}
2728 \DeclareMathSymbol{\precnapprox}{\mathrel}{\M@extsymbols@font}{"2AB9}
2729 \DeclareMathSymbol{\succnapprox}{\mathrel}{\M@extsymbols@font}{"2ABA}

```

```
2730 \DeclareMathSymbol{\nequiv}{\mathrel}{\M@extsymbols@font}{2262}
```

We handle `\ng` specially. The \LaTeX kernel defines `\ng` as a text symbol, so we define `\mathng` like for `\$,` etc.

```
2731 \global\let\textng\ng
```

```
2732 \let\ng\@undefined
```

```
2733 \DeclareMathSymbol{\mathng}{\mathrel}{\M@extsymbols@font}{226F}
```

```
2734 \protected\gdef\ng{\ifmmode\mathng\else\textng\fi}
```

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

```
2735 \ifM@adjust@font\else
```

```
2736   \DeclareMathSymbol{\nabla}{\mathord}{\M@extsymbols@font}{2207}
```

```
2737 \fi}
```

Set arrows.

```
2738 \def\M@arrows@set{%
```

```
2739   \edef\M@arrows@font{M\M@arrowsshape\@tempa}
```

```
2740   \let\uparrow\@undefined
```

```
2741   \let\Uparrow\@undefined
```

```
2742   \let\downarrow\@undefined
```

```
2743   \let\Downarrow\@undefined
```

```
2744   \let\updownarrow\@undefined
```

```
2745   \let\Updownarrow\@undefined
```

```
2746   \let\longrightarrow\@undefined
```

```
2747   \let\longleftarrow\@undefined
```

```
2748   \let\longlefttrightarrow\@undefined
```

```
2749   \let\hookrightarrow\@undefined
```

```
2750   \let\hookleftarrow\@undefined
```

```
2751   \let\Longrightarrow\@undefined
```

```
2752   \let\Longleftarrow\@undefined
```

```
2753   \let\Longlefttrightarrow\@undefined
```

```
2754   \let\rightleftharpoons\@undefined
```

```
2755   \DeclareMathSymbol{\rightarrow}{\mathrel}{\M@arrows@font}{2192}
```

```
2756   \global\let\to\rightarrow
```

```
2757   \DeclareMathSymbol{\nrightarrow}{\mathrel}{\M@arrows@font}{219B}
```

```
2758   \DeclareMathSymbol{\Rightarrow}{\mathrel}{\M@arrows@font}{21D2}
```

```
2759   \DeclareMathSymbol{\nRightarrow}{\mathrel}{\M@arrows@font}{21CF}
```

```
2760   \DeclareMathSymbol{\Rrightarrow}{\mathrel}{\M@arrows@font}{21DB}
```

```
2761   \DeclareMathSymbol{\longrightarrow}{\mathrel}{\M@arrows@font}{27F6}
```

```
2762   \DeclareMathSymbol{\Longrightarrow}{\mathrel}{\M@arrows@font}{27F9}
```

```
2763   \DeclareMathSymbol{\rightbararrow}{\mathrel}{\M@arrows@font}{21A6}
```

```
2764   \global\let\mapsto\rightbararrow
```

```
2765   \DeclareMathSymbol{\Rightbararrow}{\mathrel}{\M@arrows@font}{2907}
```

```
2766   \DeclareMathSymbol{\longrightbararrow}{\mathrel}{\M@arrows@font}{27FC}
```

```
2767   \global\let\longmapsto\longrightbararrow
```

```
2768   \DeclareMathSymbol{\Longrightbararrow}{\mathrel}{\M@arrows@font}{27FE}
```

```
2769   \DeclareMathSymbol{\hookrightarrow}{\mathrel}{\M@arrows@font}{21AA}
```

```
2770   \DeclareMathSymbol{\righdasharrow}{\mathrel}{\M@arrows@font}{21E2}
```

```
2771   \DeclareMathSymbol{\rightharpoonup}{\mathrel}{\M@arrows@font}{21C0}
```

```

2772 \DeclareMathSymbol{\rightharpoondown}{\mathrel}{\M@arrows@font}{"21C1}
2773 \DeclareMathSymbol{\rightarrowtail}{\mathrel}{\M@arrows@font}{"21A3}
2774 \DeclareMathSymbol{\rightplusarrow}{\mathrel}{\M@arrows@font}{"27F4}
2775 \DeclareMathSymbol{\rightwvearrow}{\mathrel}{\M@arrows@font}{"219D}
2776 \DeclareMathSymbol{\rightsquigarrow}{\mathrel}{\M@arrows@font}{"21DD}
2777 \DeclareMathSymbol{\longrightsquigarrow}{\mathrel}{\M@arrows@font}{"27FF}
2778 \DeclareMathSymbol{\looparrowright}{\mathrel}{\M@arrows@font}{"21AC}
2779 \DeclareMathSymbol{\curvearrowright}{\mathrel}{\M@arrows@font}{"293B}
2780 \DeclareMathSymbol{\circlearrowright}{\mathrel}{\M@arrows@font}{"21BB}
2781 \DeclareMathSymbol{\twoheadrightarrow}{\mathrel}{\M@arrows@font}{"21A0}
2782 \DeclareMathSymbol{\rightarrowto bar}{\mathrel}{\M@arrows@font}{"21E5}
2783 \DeclareMathSymbol{\rightarrowto bar}{\mathrel}{\M@arrows@font}{"21E5}
2784 \DeclareMathSymbol{\rightarrowto bar}{\mathrel}{\M@arrows@font}{"21E5}
2784 \DeclareMathSymbol{\rightarrowto bar}{\mathrel}{\M@arrows@font}{"21E5}
2785 \DeclareMathSymbol{\rightarrowto bar}{\mathrel}{\M@arrows@font}{"21E5}
2786 \DeclareMathSymbol{\leftarrow}{\mathrel}{\M@arrows@font}{"2190}
2787 \global\let\from\leftarrow
2788 \DeclareMathSymbol{\nleftarrow}{\mathrel}{\M@arrows@font}{"219A}
2789 \DeclareMathSymbol{\Leftarrow}{\mathrel}{\M@arrows@font}{"21D0}
2790 \DeclareMathSymbol{\nLeftarrow}{\mathrel}{\M@arrows@font}{"21CD}
2791 \DeclareMathSymbol{\Lleftarrow}{\mathrel}{\M@arrows@font}{"21DA}
2792 \DeclareMathSymbol{\longleftarrow}{\mathrel}{\M@arrows@font}{"27F5}
2793 \DeclareMathSymbol{\Longleftarrow}{\mathrel}{\M@arrows@font}{"27F8}
2794 \DeclareMathSymbol{\leftbararrow}{\mathrel}{\M@arrows@font}{"21A4}
2795 \global\let\mapsfrom\leftbararrow
2796 \DeclareMathSymbol{\Leftbararrow}{\mathrel}{\M@arrows@font}{"2906}
2797 \DeclareMathSymbol{\longleftbararrow}{\mathrel}{\M@arrows@font}{"27FB}
2798 \global\let\longmapsfrom\longleftbararrow
2799 \DeclareMathSymbol{\Longleftbararrow}{\mathrel}{\M@arrows@font}{"27FD}
2800 \DeclareMathSymbol{\hookleftarrow}{\mathrel}{\M@arrows@font}{"21A9}
2801 \DeclareMathSymbol{\leftdasharrow}{\mathrel}{\M@arrows@font}{"21E0}
2802 \DeclareMathSymbol{\leftharpoonup}{\mathrel}{\M@arrows@font}{"21BC}
2803 \DeclareMathSymbol{\leftharpoondown}{\mathrel}{\M@arrows@font}{"21BD}
2804 \DeclareMathSymbol{\leftarrowtail}{\mathrel}{\M@arrows@font}{"21A2}
2805 \DeclareMathSymbol{\leftplusarrow}{\mathrel}{\M@arrows@font}{"2B32}
2806 \DeclareMathSymbol{\leftwvearrow}{\mathrel}{\M@arrows@font}{"219C}
2807 \DeclareMathSymbol{\leftsquigarrow}{\mathrel}{\M@arrows@font}{"21DC}
2808 \DeclareMathSymbol{\longleftsquigarrow}{\mathrel}{\M@arrows@font}{"2B33}
2809 \DeclareMathSymbol{\looparrowleft}{\mathrel}{\M@arrows@font}{"21AB}
2810 \DeclareMathSymbol{\curvearrowleft}{\mathrel}{\M@arrows@font}{"293A}
2811 \DeclareMathSymbol{\circlearrowleft}{\mathrel}{\M@arrows@font}{"21BA}
2812 \DeclareMathSymbol{\twoheadleftarrow}{\mathrel}{\M@arrows@font}{"219E}
2813 \DeclareMathSymbol{\leftarrowto bar}{\mathrel}{\M@arrows@font}{"21E4}
2814 \DeclareMathSymbol{\leftwhitearrow}{\mathrel}{\M@arrows@font}{"21E6}
2815 \DeclareMathSymbol{\leftleftarrows}{\mathrel}{\M@arrows@font}{"21C7}
2816 \DeclareMathSymbol{\leftleftleftarrows}{\mathrel}{\M@arrows@font}{"2B31}
2817 \DeclareMathSymbol{\leftrightarrow}{\mathrel}{\M@arrows@font}{"2194}
2818 \DeclareMathSymbol{\Leftrightarrow}{\mathrel}{\M@arrows@font}{"21D4}

```

```

2819 \DeclareMathSymbol{\nLeftrightarrow}{\mathrel}{\M@arrows@font}{"21CE}
2820 \DeclareMathSymbol{\longleftarrow}{\mathrel}{\M@arrows@font}{"27F7}
2821 \DeclareMathSymbol{\Longleftarrow}{\mathrel}{\M@arrows@font}{"27FA}
2822 \DeclareMathSymbol{\leftrightharpoonup}{\mathrel}{\M@arrows@font}{"21AD}
2823 \DeclareMathSymbol{\leftrightharpoons}{\mathrel}{\M@arrows@font}{"21C6}
2824 \DeclareMathSymbol{\leftrightharpoons}{\mathrel}{\M@arrows@font}{"21CB}
2825 \DeclareMathSymbol{\leftrightharpoons}{\mathrel}{\M@arrows@font}{"21B9}
2826 \DeclareMathSymbol{\rightleftarrows}{\mathrel}{\M@arrows@font}{"21C4}
2827 \DeclareMathSymbol{\rightleftharpoons}{\mathrel}{\M@arrows@font}{"21CC}
2828 \DeclareMathSymbol{\uparrow}{\mathrel}{\M@arrows@font}{"2191}
2829 \DeclareMathSymbol{\Uparrow}{\mathrel}{\M@arrows@font}{"21D1}
2830 \DeclareMathSymbol{\Upuparrows}{\mathrel}{\M@arrows@font}{"290A}
2831 \DeclareMathSymbol{\upbararrow}{\mathrel}{\M@arrows@font}{"21A5}
2832 \DeclareMathSymbol{\updasharrow}{\mathrel}{\M@arrows@font}{"21E1}
2833 \DeclareMathSymbol{\upharpoonleft}{\mathrel}{\M@arrows@font}{"21BF}
2834 \DeclareMathSymbol{\upharpoonright}{\mathrel}{\M@arrows@font}{"21BE}
2835 \DeclareMathSymbol{\twoheaduparrow}{\mathrel}{\M@arrows@font}{"219F}
2836 \DeclareMathSymbol{\uparrowto bar}{\mathrel}{\M@arrows@font}{"2912}
2837 \DeclareMathSymbol{\upwhitearrow}{\mathrel}{\M@arrows@font}{"21E7}
2838 \DeclareMathSymbol{\upwhitebararrow}{\mathrel}{\M@arrows@font}{"21EA}
2839 \DeclareMathSymbol{\upuparrows}{\mathrel}{\M@arrows@font}{"21C8}
2840 \DeclareMathSymbol{\downarrow}{\mathrel}{\M@arrows@font}{"2193}
2841 \DeclareMathSymbol{\Downarrow}{\mathrel}{\M@arrows@font}{"21D3}
2842 \DeclareMathSymbol{\Ddownarrow}{\mathrel}{\M@arrows@font}{"290B}
2843 \DeclareMathSymbol{\downbararrow}{\mathrel}{\M@arrows@font}{"21A7}
2844 \DeclareMathSymbol{\downdasharrow}{\mathrel}{\M@arrows@font}{"21E3}
2845 \DeclareMathSymbol{\zigzagarrow}{\mathrel}{\M@arrows@font}{"21AF}
2846 \global\let\lightningboltarrow\zigzagarrow
2847 \DeclareMathSymbol{\downharpoonleft}{\mathrel}{\M@arrows@font}{"21C3}
2848 \DeclareMathSymbol{\downharpoonright}{\mathrel}{\M@arrows@font}{"21C2}
2849 \DeclareMathSymbol{\twoheaddownarrow}{\mathrel}{\M@arrows@font}{"21A1}
2850 \DeclareMathSymbol{\downarrowto bar}{\mathrel}{\M@arrows@font}{"2913}
2851 \DeclareMathSymbol{\downwhitearrow}{\mathrel}{\M@arrows@font}{"21E9}
2852 \DeclareMathSymbol{\downdownarrows}{\mathrel}{\M@arrows@font}{"21CA}
2853 \DeclareMathSymbol{\updownarrow}{\mathrel}{\M@arrows@font}{"2195}
2854 \DeclareMathSymbol{\Updownarrow}{\mathrel}{\M@arrows@font}{"21D5}
2855 \DeclareMathSymbol{\updownarrows}{\mathrel}{\M@arrows@font}{"21C5}
2856 \DeclareMathSymbol{\downuparrows}{\mathrel}{\M@arrows@font}{"21F5}
2857 \DeclareMathSymbol{\updownharpoons}{\mathrel}{\M@arrows@font}{"296E}
2858 \DeclareMathSymbol{\downupharpoons}{\mathrel}{\M@arrows@font}{"296F}
2859 \DeclareMathSymbol{\nearrow}{\mathrel}{\M@arrows@font}{"2197}
2860 \DeclareMathSymbol{\Nearrow}{\mathrel}{\M@arrows@font}{"21D7}
2861 \DeclareMathSymbol{\nwarrows}{\mathrel}{\M@arrows@font}{"2196}
2862 \DeclareMathSymbol{\Nwarrows}{\mathrel}{\M@arrows@font}{"21D6}
2863 \DeclareMathSymbol{\searrow}{\mathrel}{\M@arrows@font}{"2198}
2864 \DeclareMathSymbol{\Searrow}{\mathrel}{\M@arrows@font}{"21D8}
2865 \DeclareMathSymbol{\swarrow}{\mathrel}{\M@arrows@font}{"2199}

```

```

2866 \DeclareMathSymbol{\Swarrow}{\mathrel}{\M@arrows@font}{"21D9}
2867 \DeclareMathSymbol{\nwsearrow}{\mathrel}{\M@arrows@font}{"2921}
2868 \DeclareMathSymbol{\neswarrow}{\mathrel}{\M@arrows@font}{"2922}
2869 \DeclareMathSymbol{\lrcirclearrow}{\mathrel}{\M@arrows@font}{"27F2}
2870 \DeclareMathSymbol{\rcirclearrow}{\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.

```

2871 \def\M@bb@set{%
2872   \protected\def\mathbb##1{\relax
2873     \ifmmode\else
2874       \M@HModeError\mathbb
2875       $%
2876     \fi
2877     \begingroup
2878       \M@bb@mathcodes
2879     ##1%
2880   \endgroup}
2881 \edef\M@bb@num{\number\csname symM\M@bbshape\@tempa\endcsname}
2882 \protected\edef\M@bb@mathcodes{%
2883 \Umathcode`A=0+\M@bb@num"1D538\relax
2884 \Umathcode`B=0+\M@bb@num"1D539\relax
2885 \Umathcode`C=0+\M@bb@num"2102\relax
2886 \Umathcode`D=0+\M@bb@num"1D53B\relax
2887 \Umathcode`E=0+\M@bb@num"1D53C\relax
2888 \Umathcode`F=0+\M@bb@num"1D53D\relax
2889 \Umathcode`G=0+\M@bb@num"1D53E\relax
2890 \Umathcode`H=0+\M@bb@num"210D\relax
2891 \Umathcode`I=0+\M@bb@num"1D540\relax
2892 \Umathcode`J=0+\M@bb@num"1D541\relax
2893 \Umathcode`K=0+\M@bb@num"1D542\relax
2894 \Umathcode`L=0+\M@bb@num"1D543\relax
2895 \Umathcode`M=0+\M@bb@num"1D544\relax
2896 \Umathcode`N=0+\M@bb@num"2115\relax
2897 \Umathcode`O=0+\M@bb@num"1D546\relax
2898 \Umathcode`P=0+\M@bb@num"2119\relax
2899 \Umathcode`Q=0+\M@bb@num"211A\relax
2900 \Umathcode`R=0+\M@bb@num"211D\relax
2901 \Umathcode`S=0+\M@bb@num"1D54A\relax
2902 \Umathcode`T=0+\M@bb@num"1D54B\relax
2903 \Umathcode`U=0+\M@bb@num"1D54C\relax
2904 \Umathcode`V=0+\M@bb@num"1D54D\relax
2905 \Umathcode`W=0+\M@bb@num"1D54E\relax

```

```

2906 \Umathcode`X=0+\M@bb@num"1D54F\relax
2907 \Umathcode`Y=0+\M@bb@num"1D550\relax
2908 \Umathcode`Z=0+\M@bb@num"2124\relax
2909 \Umathcode`a=0+\M@bb@num"1D552\relax
2910 \Umathcode`b=0+\M@bb@num"1D553\relax
2911 \Umathcode`c=0+\M@bb@num"1D554\relax
2912 \Umathcode`d=0+\M@bb@num"1D555\relax
2913 \Umathcode`e=0+\M@bb@num"1D556\relax
2914 \Umathcode`f=0+\M@bb@num"1D557\relax
2915 \Umathcode`g=0+\M@bb@num"1D558\relax
2916 \Umathcode`h=0+\M@bb@num"1D559\relax
2917 \Umathcode`i=0+\M@bb@num"1D55A\relax
2918 \Umathcode`j=0+\M@bb@num"1D55B\relax
2919 \Umathcode`k=0+\M@bb@num"1D55C\relax
2920 \Umathcode`l=0+\M@bb@num"1D55D\relax
2921 \Umathcode`m=0+\M@bb@num"1D55E\relax
2922 \Umathcode`n=0+\M@bb@num"1D55F\relax
2923 \Umathcode`o=0+\M@bb@num"1D560\relax
2924 \Umathcode`p=0+\M@bb@num"1D561\relax
2925 \Umathcode`q=0+\M@bb@num"1D562\relax
2926 \Umathcode`r=0+\M@bb@num"1D563\relax
2927 \Umathcode`s=0+\M@bb@num"1D564\relax
2928 \Umathcode`t=0+\M@bb@num"1D565\relax
2929 \Umathcode`u=0+\M@bb@num"1D566\relax
2930 \Umathcode`v=0+\M@bb@num"1D567\relax
2931 \Umathcode`w=0+\M@bb@num"1D568\relax
2932 \Umathcode`x=0+\M@bb@num"1D569\relax
2933 \Umathcode`y=0+\M@bb@num"1D56A\relax
2934 \Umathcode`z=0+\M@bb@num"1D56B\relax
2935 \Umathcode`0=0+\M@bb@num"1D7D8\relax
2936 \Umathcode`1=0+\M@bb@num"1D7D9\relax
2937 \Umathcode`2=0+\M@bb@num"1D7DA\relax
2938 \Umathcode`3=0+\M@bb@num"1D7DB\relax
2939 \Umathcode`4=0+\M@bb@num"1D7DC\relax
2940 \Umathcode`5=0+\M@bb@num"1D7DD\relax
2941 \Umathcode`6=0+\M@bb@num"1D7DE\relax
2942 \Umathcode`7=0+\M@bb@num"1D7DF\relax
2943 \Umathcode`8=0+\M@bb@num"1D7E0\relax
2944 \Umathcode`9=0+\M@bb@num"1D7E1\relax}}

```

Set caligraphic letters.

```

2945 \def\M@cal@set{%
2946 \protected\def\mathcal##1{\relax
2947 \ifmmode\else
2948 \M@HModeError\mathcal
2949 $%
2950 \fi
2951 \begingroup

```

```

2952     \M@cal@mathcodes
2953     ##1%
2954     \endgroup}
2955     \edef\M@cal@num{\number\csname symM\M@cal@shape\@tempa\endcsname}
2956     \protected\edef\M@cal@mathcodes{%
2957     \Umathcode`A=0+\M@cal@num"1D49C\relax
2958     \Umathcode`B=0+\M@cal@num"212C\relax
2959     \Umathcode`C=0+\M@cal@num"1D49E\relax
2960     \Umathcode`D=0+\M@cal@num"1D49F\relax
2961     \Umathcode`E=0+\M@cal@num"2130\relax
2962     \Umathcode`F=0+\M@cal@num"2131\relax
2963     \Umathcode`G=0+\M@cal@num"1D4A2\relax
2964     \Umathcode`H=0+\M@cal@num"210B\relax
2965     \Umathcode`I=0+\M@cal@num"2110\relax
2966     \Umathcode`J=0+\M@cal@num"1D4A5\relax
2967     \Umathcode`K=0+\M@cal@num"1D4A6\relax
2968     \Umathcode`L=0+\M@cal@num"2112\relax
2969     \Umathcode`M=0+\M@cal@num"2133\relax
2970     \Umathcode`N=0+\M@cal@num"1D4A9\relax
2971     \Umathcode`O=0+\M@cal@num"1D4AA\relax
2972     \Umathcode`P=0+\M@cal@num"1D4AB\relax
2973     \Umathcode`Q=0+\M@cal@num"1D4AC\relax
2974     \Umathcode`R=0+\M@cal@num"211B\relax
2975     \Umathcode`S=0+\M@cal@num"1D4AE\relax
2976     \Umathcode`T=0+\M@cal@num"1D4AF\relax
2977     \Umathcode`U=0+\M@cal@num"1D4B0\relax
2978     \Umathcode`V=0+\M@cal@num"1D4B1\relax
2979     \Umathcode`W=0+\M@cal@num"1D4B2\relax
2980     \Umathcode`X=0+\M@cal@num"1D4B3\relax
2981     \Umathcode`Y=0+\M@cal@num"1D4B4\relax
2982     \Umathcode`Z=0+\M@cal@num"1D4B5\relax
2983     \Umathcode`a=0+\M@cal@num"1D4B6\relax
2984     \Umathcode`b=0+\M@cal@num"1D4B7\relax
2985     \Umathcode`c=0+\M@cal@num"1D4B8\relax
2986     \Umathcode`d=0+\M@cal@num"1D4B9\relax
2987     \Umathcode`e=0+\M@cal@num"212F\relax
2988     \Umathcode`f=0+\M@cal@num"1D4BB\relax
2989     \Umathcode`g=0+\M@cal@num"210A\relax
2990     \Umathcode`h=0+\M@cal@num"1D4BD\relax
2991     \Umathcode`i=0+\M@cal@num"1D4BE\relax
2992     \Umathcode`j=0+\M@cal@num"1D4BF\relax
2993     \Umathcode`k=0+\M@cal@num"1D4C0\relax
2994     \Umathcode`l=0+\M@cal@num"1D4C1\relax
2995     \Umathcode`m=0+\M@cal@num"1D4C2\relax
2996     \Umathcode`n=0+\M@cal@num"1D4C3\relax
2997     \Umathcode`o=0+\M@cal@num"2134\relax
2998     \Umathcode`p=0+\M@cal@num"1D4C5\relax

```



```

2999 \Umathcode`q=0+\M@cal@num"1D4C6\relax
3000 \Umathcode`r=0+\M@cal@num"1D4C7\relax
3001 \Umathcode`s=0+\M@cal@num"1D4C8\relax
3002 \Umathcode`t=0+\M@cal@num"1D4C9\relax
3003 \Umathcode`u=0+\M@cal@num"1D4CA\relax
3004 \Umathcode`v=0+\M@cal@num"1D4CB\relax
3005 \Umathcode`w=0+\M@cal@num"1D4CC\relax
3006 \Umathcode`x=0+\M@cal@num"1D4CD\relax
3007 \Umathcode`y=0+\M@cal@num"1D4CE\relax
3008 \Umathcode`z=0+\M@cal@num"1D4CF\relax}]

```

Set fraktur letters.

```

3009 \def\M@frak@set{%
3010 \protected\def\mathfrak##1{\relax
3011 \ifmmode\else
3012 \M@HModeError\mathfrak
3013 $%
3014 \fi
3015 \begingroup
3016 \M@frak@mathcodes
3017 ##1%
3018 \endgroup}
3019 \edef\M@frak@num{\number\csname symM\M@frakshape\@tempa\endcsname}
3020 \protected\edef\M@frak@mathcodes{%
3021 \Umathcode`A=0+\M@frak@num"1D504\relax
3022 \Umathcode`B=0+\M@frak@num"1D505\relax
3023 \Umathcode`C=0+\M@frak@num"212D\relax
3024 \Umathcode`D=0+\M@frak@num"1D507\relax
3025 \Umathcode`E=0+\M@frak@num"1D508\relax
3026 \Umathcode`F=0+\M@frak@num"1D509\relax
3027 \Umathcode`G=0+\M@frak@num"1D50A\relax
3028 \Umathcode`H=0+\M@frak@num"210C\relax
3029 \Umathcode`I=0+\M@frak@num"2111\relax
3030 \Umathcode`J=0+\M@frak@num"1D50D\relax
3031 \Umathcode`K=0+\M@frak@num"1D50E\relax
3032 \Umathcode`L=0+\M@frak@num"1D50F\relax
3033 \Umathcode`M=0+\M@frak@num"1D510\relax
3034 \Umathcode`N=0+\M@frak@num"1D511\relax
3035 \Umathcode`O=0+\M@frak@num"1D512\relax
3036 \Umathcode`P=0+\M@frak@num"1D513\relax
3037 \Umathcode`Q=0+\M@frak@num"1D514\relax
3038 \Umathcode`R=0+\M@frak@num"211C\relax
3039 \Umathcode`S=0+\M@frak@num"1D516\relax
3040 \Umathcode`T=0+\M@frak@num"1D517\relax
3041 \Umathcode`U=0+\M@frak@num"1D518\relax
3042 \Umathcode`V=0+\M@frak@num"1D519\relax
3043 \Umathcode`W=0+\M@frak@num"1D51A\relax
3044 \Umathcode`X=0+\M@frak@num"1D51B\relax

```

```

3045 \Umathcode`Y=0+\M@frac@num"1D51C\relax
3046 \Umathcode`Z=0+\M@frac@num"2128\relax
3047 \Umathcode`a=0+\M@frac@num"1D51E\relax
3048 \Umathcode`b=0+\M@frac@num"1D51F\relax
3049 \Umathcode`c=0+\M@frac@num"1D520\relax
3050 \Umathcode`d=0+\M@frac@num"1D521\relax
3051 \Umathcode`e=0+\M@frac@num"1D522\relax
3052 \Umathcode`f=0+\M@frac@num"1D523\relax
3053 \Umathcode`g=0+\M@frac@num"1D524\relax
3054 \Umathcode`h=0+\M@frac@num"1D525\relax
3055 \Umathcode`i=0+\M@frac@num"1D526\relax
3056 \Umathcode`j=0+\M@frac@num"1D527\relax
3057 \Umathcode`k=0+\M@frac@num"1D528\relax
3058 \Umathcode`l=0+\M@frac@num"1D529\relax
3059 \Umathcode`m=0+\M@frac@num"1D52A\relax
3060 \Umathcode`n=0+\M@frac@num"1D52B\relax
3061 \Umathcode`o=0+\M@frac@num"1D52C\relax
3062 \Umathcode`p=0+\M@frac@num"1D52D\relax
3063 \Umathcode`q=0+\M@frac@num"1D52E\relax
3064 \Umathcode`r=0+\M@frac@num"1D52F\relax
3065 \Umathcode`s=0+\M@frac@num"1D530\relax
3066 \Umathcode`t=0+\M@frac@num"1D531\relax
3067 \Umathcode`u=0+\M@frac@num"1D532\relax
3068 \Umathcode`v=0+\M@frac@num"1D533\relax
3069 \Umathcode`w=0+\M@frac@num"1D534\relax
3070 \Umathcode`x=0+\M@frac@num"1D535\relax
3071 \Umathcode`y=0+\M@frac@num"1D536\relax
3072 \Umathcode`z=0+\M@frac@num"1D537\relax}}

```

Set bold caligraphic letters.

```

3073 \def\M@bcal@set{%
3074 \protected\def\mathbcal##1{\relax
3075 \ifmmode\else
3076 \M@HModeError\mathbcal
3077 $%
3078 \fi
3079 \begingroup
3080 \M@bcal@mathcodes
3081 ##1%
3082 \endgroup}
3083 \edef\M@bcal@num{\number\csname sym\M@bcalshape\@tempa\endcsname}
3084 \protected\edef\M@bcal@mathcodes{%
3085 \Umathcode`A=0+\M@bcal@num"1D4D0\relax
3086 \Umathcode`B=0+\M@bcal@num"1D4D1\relax
3087 \Umathcode`C=0+\M@bcal@num"1D4D2\relax
3088 \Umathcode`D=0+\M@bcal@num"1D4D3\relax
3089 \Umathcode`E=0+\M@bcal@num"1D4D4\relax
3090 \Umathcode`F=0+\M@bcal@num"1D4D5\relax

```

```

3091 \Umathcode`G=0+\M@bcal@num"1D4D6\relax
3092 \Umathcode`H=0+\M@bcal@num"1D4D7\relax
3093 \Umathcode`I=0+\M@bcal@num"1D4D8\relax
3094 \Umathcode`J=0+\M@bcal@num"1D4D9\relax
3095 \Umathcode`K=0+\M@bcal@num"1D4DA\relax
3096 \Umathcode`L=0+\M@bcal@num"1D4DB\relax
3097 \Umathcode`M=0+\M@bcal@num"1D4DC\relax
3098 \Umathcode`N=0+\M@bcal@num"1D4DD\relax
3099 \Umathcode`O=0+\M@bcal@num"1D4DE\relax
3100 \Umathcode`P=0+\M@bcal@num"1D4DF\relax
3101 \Umathcode`Q=0+\M@bcal@num"1D4E0\relax
3102 \Umathcode`R=0+\M@bcal@num"1D4E1\relax
3103 \Umathcode`S=0+\M@bcal@num"1D4E2\relax
3104 \Umathcode`T=0+\M@bcal@num"1D4E3\relax
3105 \Umathcode`U=0+\M@bcal@num"1D4E4\relax
3106 \Umathcode`V=0+\M@bcal@num"1D4E5\relax
3107 \Umathcode`W=0+\M@bcal@num"1D4E6\relax
3108 \Umathcode`X=0+\M@bcal@num"1D4E7\relax
3109 \Umathcode`Y=0+\M@bcal@num"1D4E8\relax
3110 \Umathcode`Z=0+\M@bcal@num"1D4E9\relax
3111 \Umathcode`a=0+\M@bcal@num"1D4EA\relax
3112 \Umathcode`b=0+\M@bcal@num"1D4EB\relax
3113 \Umathcode`c=0+\M@bcal@num"1D4EC\relax
3114 \Umathcode`d=0+\M@bcal@num"1D4ED\relax
3115 \Umathcode`e=0+\M@bcal@num"1D4EE\relax
3116 \Umathcode`f=0+\M@bcal@num"1D4EF\relax
3117 \Umathcode`g=0+\M@bcal@num"1D4F0\relax
3118 \Umathcode`h=0+\M@bcal@num"1D4F1\relax
3119 \Umathcode`i=0+\M@bcal@num"1D4F2\relax
3120 \Umathcode`j=0+\M@bcal@num"1D4F3\relax
3121 \Umathcode`k=0+\M@bcal@num"1D4F4\relax
3122 \Umathcode`l=0+\M@bcal@num"1D4F5\relax
3123 \Umathcode`m=0+\M@bcal@num"1D4F6\relax
3124 \Umathcode`n=0+\M@bcal@num"1D4F7\relax
3125 \Umathcode`o=0+\M@bcal@num"1D4F8\relax
3126 \Umathcode`p=0+\M@bcal@num"1D4F9\relax
3127 \Umathcode`q=0+\M@bcal@num"1D4FA\relax
3128 \Umathcode`r=0+\M@bcal@num"1D4FB\relax
3129 \Umathcode`s=0+\M@bcal@num"1D4FC\relax
3130 \Umathcode`t=0+\M@bcal@num"1D4FD\relax
3131 \Umathcode`u=0+\M@bcal@num"1D4FE\relax
3132 \Umathcode`v=0+\M@bcal@num"1D4FF\relax
3133 \Umathcode`w=0+\M@bcal@num"1D500\relax
3134 \Umathcode`x=0+\M@bcal@num"1D501\relax
3135 \Umathcode`y=0+\M@bcal@num"1D502\relax
3136 \Umathcode`z=0+\M@bcal@num"1D503\relax}}

```

Set bold fraktur letters.

```

3137 \def\M@bfrac@set{%
3138   \protected\def\mathbfrac##1{\relax
3139     \ifmmode\else
3140       \M@HModeError\mathbfrac
3141       $%
3142     \fi
3143     \begingroup
3144       \M@bfrac@mathcodes
3145       ##1%
3146     \endgroup}
3147 \edef\M@bfrac@num{\number\csname symM\M@bfracshape\@tempa\endcsname}
3148 \protected\edef\M@bfrac@mathcodes{%
3149   \Umathcode`A=0+\M@bfrac@num"1D56C\relax
3150   \Umathcode`B=0+\M@bfrac@num"1D56D\relax
3151   \Umathcode`C=0+\M@bfrac@num"1D56E\relax
3152   \Umathcode`D=0+\M@bfrac@num"1D56F\relax
3153   \Umathcode`E=0+\M@bfrac@num"1D570\relax
3154   \Umathcode`F=0+\M@bfrac@num"1D571\relax
3155   \Umathcode`G=0+\M@bfrac@num"1D572\relax
3156   \Umathcode`H=0+\M@bfrac@num"1D573\relax
3157   \Umathcode`I=0+\M@bfrac@num"1D574\relax
3158   \Umathcode`J=0+\M@bfrac@num"1D575\relax
3159   \Umathcode`K=0+\M@bfrac@num"1D576\relax
3160   \Umathcode`L=0+\M@bfrac@num"1D577\relax
3161   \Umathcode`M=0+\M@bfrac@num"1D578\relax
3162   \Umathcode`N=0+\M@bfrac@num"1D579\relax
3163   \Umathcode`O=0+\M@bfrac@num"1D57A\relax
3164   \Umathcode`P=0+\M@bfrac@num"1D57B\relax
3165   \Umathcode`Q=0+\M@bfrac@num"1D57C\relax
3166   \Umathcode`R=0+\M@bfrac@num"1D57D\relax
3167   \Umathcode`S=0+\M@bfrac@num"1D57E\relax
3168   \Umathcode`T=0+\M@bfrac@num"1D57F\relax
3169   \Umathcode`U=0+\M@bfrac@num"1D580\relax
3170   \Umathcode`V=0+\M@bfrac@num"1D581\relax
3171   \Umathcode`W=0+\M@bfrac@num"1D582\relax
3172   \Umathcode`X=0+\M@bfrac@num"1D583\relax
3173   \Umathcode`Y=0+\M@bfrac@num"1D584\relax
3174   \Umathcode`Z=0+\M@bfrac@num"1D585\relax
3175   \Umathcode`a=0+\M@bfrac@num"1D586\relax
3176   \Umathcode`b=0+\M@bfrac@num"1D587\relax
3177   \Umathcode`c=0+\M@bfrac@num"1D588\relax
3178   \Umathcode`d=0+\M@bfrac@num"1D589\relax
3179   \Umathcode`e=0+\M@bfrac@num"1D58A\relax
3180   \Umathcode`f=0+\M@bfrac@num"1D58B\relax
3181   \Umathcode`g=0+\M@bfrac@num"1D58C\relax
3182   \Umathcode`h=0+\M@bfrac@num"1D58D\relax
3183   \Umathcode`i=0+\M@bfrac@num"1D58E\relax

```

```
3184 \Umathcode`j=0+\M@bfrac@num"1D58F\relax
3185 \Umathcode`k=0+\M@bfrac@num"1D590\relax
3186 \Umathcode`l=0+\M@bfrac@num"1D591\relax
3187 \Umathcode`m=0+\M@bfrac@num"1D592\relax
3188 \Umathcode`n=0+\M@bfrac@num"1D593\relax
3189 \Umathcode`o=0+\M@bfrac@num"1D594\relax
3190 \Umathcode`p=0+\M@bfrac@num"1D595\relax
3191 \Umathcode`q=0+\M@bfrac@num"1D596\relax
3192 \Umathcode`r=0+\M@bfrac@num"1D597\relax
3193 \Umathcode`s=0+\M@bfrac@num"1D598\relax
3194 \Umathcode`t=0+\M@bfrac@num"1D599\relax
3195 \Umathcode`u=0+\M@bfrac@num"1D59A\relax
3196 \Umathcode`v=0+\M@bfrac@num"1D59B\relax
3197 \Umathcode`w=0+\M@bfrac@num"1D59C\relax
3198 \Umathcode`x=0+\M@bfrac@num"1D59D\relax
3199 \Umathcode`y=0+\M@bfrac@num"1D59E\relax
3200 \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>1.1b July 2018
—initial release</p> <p>1.2 August 2018
—minor bug fix for <code>\mathfrak</code>
—eliminated redundant batchfile</p> <p>1.3 January 2019
—added <code>symbols</code> keyword
—created <code>mathfont_example.pdf</code>
—corrected the description of the <code>mathastext</code> package
—font-change <code>\message</code> added to <code>\mathfont</code></p> <p>1.4 April 2019
—<code>\setfont</code> command added
—<code>\mathfont</code> optional argument can parse spaces
—<code>no-operators</code> now default package optional argument
—added <code>\comma</code> command
—new fancy fatal error message
—improved messaging for <code>\mathfont</code>
—internal command <code>\mathpound</code> changed to <code>\mathhash</code>
—added a missing <code>#1</code> after <code>\char`\"</code> in the example code redefining <code>"</code> in the user guide</p> <p>1.5 April 2019
—separated <code>\increment</code> and <code>\Delta</code>
—version history added
—initial off-the-shelf use insert added</p> <p>1.6 December 2019
—separated implementation and user documentation
—created <code>mathfont_heading.tex</code>
—created <code>mathfont_doc_patch.tex</code> for use with the index
—changed <code>mathfont_greek.pdf</code> to <code>mathfont_symbol_list.pdf</code></p> | <p>—eliminated <code>mathfont_example.pdf</code>
—eliminated <code>operators</code> package option
—eliminated <code>packages</code> package option
—font name can be package option
—added Hebrew and Cyrillic characters
—separated ancient Greek from modern Greek characters
—created new keywords: <code>extsymbols</code>, <code>delimiters</code>, <code>arrows</code>, <code>diacritics</code>, <code>bigops</code>, <code>extbigops</code>
—improved messaging
—improved internal code for local font-change commands
—improved space parsing for the optional argument of <code>\mathfont</code>
—bug fix for <code>\#</code>, etc. commands
—bad input for <code>\mathbb</code>, etc. now gives a warning
—improved error checking for <code>\newmathrm</code>, etc. commands
—<code>\mathfont</code> now ignores bad options (on top of issuing an error)
—internal commands now begin with <code>\M@...</code>
—added Easter Egg!
—improved indexing
—<code>mathfont.dtx</code> renamed as <code>mathfont_code.dtx</code>
—<code>\newmathbold</code> renamed as <code>\newmathbf</code>
—default local font changes now use <code>\updefault</code>, etc.
—added fatal error for missing <code>fontspec</code>
—fatal errors result in <code>\endinput</code> rather than <code>\@@end</code></p> <p>2.0 December 2021</p> <div style="border: 1px solid black; padding: 5px; margin: 5px 0;"> <p>Big Change: Font adjustments for LuaTeX: new glyph boundaries for Latin letters in math mode, resizable delimiters, actual 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) and text (as pronunciation symbol)

Index

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

Symbols	
<code>\#</code>	2530
<code>\%</code>	2531
<code>\&</code>	2532
<code>\=</code>	1018
<code>\@Relbar</code>	2534, 2541
<code>\@backslashchar</code>	
... 841, 855, 857, 858, 1053, 1056, 1058	
<code>\@basefeatures</code> ..	559, 561, 566, 568 , 597
<code>\@expandtwoargs</code>	837, 841, 1045, 1053
<code>\@mathconstantsfont</code>	729, 730
<code>\@mathfont</code>	655, 656 , 719
<code>\@optionpresentfalse</code>	611, 624
<code>\@optionpresenttrue</code>	606
<code>\@percentchar</code> .	849, 1153, 1225, 1305, 1385
<code>\@relbar</code>	2533, 2540
<code>\@sqrts@gn</code>	2415, 2419, 2433
<code>\@tempbase</code>	16
<code>\@tempfeatures</code>	16
<code>\@verticalbar</code>	2535, 2542
<code>\~</code>	1017
<code>_</code>	50, 91
A	
<code>\aacute</code>	2064
<code>\acute</code>	2063
<code>\aftergroup</code>	767
<code>\aleph</code>	2241
<code>\Alpha</code>	1823, 2076
<code>\amalg</code>	2604
<code>\approx</code>	2519
<code>\approxeq</code>	2654
<code>\arceq</code>	2661
<code>\asympt</code>	2682
<code>\AtEndDocument</code>	164
<code>\ayin</code>	2256
B	
<code>\backslash</code>	2362
Bad argument for	10
<code>\bar</code>	2071
<code>\bclubsuit</code>	2586, 2597
<code>\bdiamondsuit</code>	2587
<code>\because</code>	2639
<code>\Beta</code>	1824, 2077
<code>\beta</code>	1794, 2117
<code>\beth</code>	2242
<code>\bheartsuit</code>	2588
<code>\bigand</code>	2548
<code>\bigat</code>	2544
<code>\bigdiv</code>	2554
<code>\bigdollar</code>	2546
<code>\bighash</code>	2545
<code>\bigp</code>	2550
<code>\bigpercent</code>	2547
<code>\bigplus</code>	2549
<code>\bigq</code>	2551
<code>\bigS</code>	2552
<code>\bigsqcap</code>	1885, 2475
<code>\bigtimes</code>	2553
<code>\bot</code>	2578
<code>\breve</code>	2068
<code>\bspadesuit</code>	2589, 2594
<code>\bullet</code>	2508
C	
cannot find the file luaotfload	4
catcode changes	3
<code>\cdot</code>	2511
<code>\CharmLine</code>	
. 40, 40 , 323, 327, 892, 902, 1041 , 1071	
<code>\check</code>	2070
<code>\Chi</code>	1844, 2097
<code>\chi</code>	1814, 2137
<code>\circlearrowleft</code>	2811
<code>\circlearrowright</code>	2780
<code>\coloneq</code>	2658
<code>\comma</code>	2499
<code>\cramped</code>	1491
<code>\curvearrowleft</code>	2810
<code>\curvearrowright</code>	2779
D	
<code>\dagger</code>	2509

- `\daleth` 2244
`\dashv` 2580
`\ddagger` 2510
`\ddot` 2066
`\Ddownarrow` 2842
`\DeclareFontFamily` 584, 595
`\DeclareFontShape`
 .. 498, 503, 508, 513, 518, 523, 528, 533
`\DeclareMathAlphabet` 790
`\DeclareSymbolFont@m@dropped`
 389, 393, **394**
`\defeq` 2666
`\degree` 2497
`\Delta` 1826, 2079
`\delta` 1796, 2119
deprecated 5, 8
`\Digamma` 2151
`\digamma` 2163
`\directlua` 42, 77, 178,
 398, 558, 565, 587, 726, 845, 1061, 1079
`\div` 1874, 2506
`\downarrowto` 2850
`\downbararrow` 2843
`\downdasharrow` 2844
`\downdownarrows` 2852
`\downharpoonleft` 2847
`\downharpoonright` 2848
`\downuparrows` 2856
`\downupharpoons` 2858
`\downwhitearrow` 2851
- E**
- `\edef@nospace` 571, 572, **652**, 671
`\emptyset` 2572
`\encsname` 546
`\endinput` 75, 122
`\Epsilon` 1827, 2080
`\epsilon` 1797, 2120
`\eqcolon` 2659
`\eqsim` 2651
`\equiv` 2520
`\Eta` 1829, 2082
`\eta` 1799, 2122
`\exists` 2571
- F**
- `\fakelangle` 1860, 2384
`\fakellangle` 1862, 2390
`\fakerrangle` 1861, 2387
`\fakerrangle` 1863, 2393
`\fflat` 2584
`\flat` 2581
`\forall` 2570
Forbidden charm info 10
`\fractionslash` 2505
- G**
- `\Gamma` 1825, 2078
`\gamma` 1795, 2118
`\gapprox` 2649
`\geq` 2517
`\geqq` 2647
`\getanddefine@fonts` 755
`\ggg` 2645
`\gimel` 2243
`\gnapprox` 2717
`\gneq` 2700
`\gneqq` 2702
`\gnsim` 2715
`\grave` 2067
`\gsim` 2669
- H**
- `\hat` 2069
`\hb@xt@` 916, 923, 928
`\hbar` 2027, 2059
`\hbox` 920, 921, 2418–2420, 2430
`\het` 2248
`\Heta` 2149
`\heta` 2161
`\hfil` 924, 926, 929, 931
`\hourglass` 2637
- I**
- I already set the font 8, 22
if-parser 22
`\ifdim` 2425
`\ifE@sterEggDecl@red` 7, 140
`\ifin@` 838, 842, 1046, 1054
`\ifM@adjust@font` 5,
 179, 189, 485, 831, 1014, 1936, 1937,
 1996, 2101, 2281, 2343, 2411, 2543, 2735
`\ifM@arg@good` . 450, 787, 864, 871, 878, 885
`\ifM@Decl@reF@mily` 451, 581
`\ifM@Decl@reF@milyB@se` 452
`\ifM@font@loaded` 6, 937, 988

- | | | | |
|--|-----------------------------------|--|------------------------|
| <code>\ifM@fromCharmFile</code> | 453, 1047, 1055 | <code>keyword frak</code> | 89 |
| <code>\ifM@Noluaotfload</code> | 4, 89 | <code>keyword greeklower</code> | 69 |
| <code>\ifM@radical</code> | 439 | <code>keyword greekupper</code> | 68 |
| <code>\ifM@XeTeXLuaTeX</code> | 3, 48 | <code>keyword hebrew</code> | 71 |
| <code>\iiintop</code> | 2464, 2465 | <code>keyword lower</code> | 66 |
| <code>\iintop</code> | 2462, 2463 | <code>keyword operator</code> | 72 |
| <code>\Im</code> | 2568 | <code>keyword radical</code> | 75 |
| <code>\imath</code> | 1765, 2025, 2057, 2336 | <code>keyword symbols</code> | 77 |
| <code>\in@</code> | 837, 841, 1045, 1053 | <code>keyword upper</code> | 65 |
| <code>\increment</code> | 2103, 2108, 2498 | <code>\Koppa</code> | 2152 |
| <code>\infty</code> | 2495 | <code>\koppa</code> | 2164 |
| <code>\IntegralItalicFactor</code> | | | |
| | 34, 35, 869, 874, 891, 899 | | |
| Internal commands restored | 8, 29 | | |
| <code>\intop</code> | 1888, 2445 | | |
| invalid command | 2 | | |
| Invalid font specifier | 9 | | |
| Invalid Option for <code>\mathfont</code> | 8 | | |
| Invalid Suboption for <code>\mathfont</code> | 8 | | |
| <code>\Iota</code> | 1831, 2084 | | |
| <code>\iota</code> | 1801, 2124 | | |
| | | | |
| | J | | |
| <code>\jmath</code> | 1766, 2026, 2058, 2337 | | |
| | | | |
| | K | | |
| <code>\kaf</code> | 2251 | | |
| <code>\Kappa</code> | 1832, 2085 | | |
| <code>\kappa</code> | 1802, 2125 | | |
| <code>\kern</code> | 2426, 2428, 2431 | | |
| keyword options for <code>\mathfont</code> | | | |
| | <i>14, 15, 19, 21–23</i> | | |
| <code>\keyword@info@begindocument</code> | 949 , 986 | | |
| keyword <code>agreeklower</code> | 70 | | |
| keyword <code>agreekupper</code> | 69 | | |
| keyword <code>arrows</code> | 83 | | |
| keyword <code>bb</code> | 86 | | |
| keyword <code>bcal</code> | 90 | | |
| keyword <code>bfrac</code> | 91 | | |
| keyword <code>bigops</code> | 76 | | |
| keyword <code>cal</code> | 87 | | |
| keyword <code>cyrilliclower</code> | 71 | | |
| keyword <code>cyrillicupper</code> | 70 | | |
| keyword <code>delimiters</code> | 74 | | |
| keyword <code>diacritics</code> | 67 | | |
| keyword <code>digits</code> | 72 | | |
| keyword <code>extbigops</code> | 76 | | |
| keyword <code>extsymbols</code> | 79 | | |
| | | | |
| | | L | |
| | | <code>\Lambda</code> | 1833, 2086 |
| | | <code>\lambda</code> | 2126 |
| | | <code>\lamed</code> | 2252 |
| | | <code>\lapprox</code> | 2648 |
| | | L ^A T _E X kernel | <i>11, 25, 76</i> |
| | | <code>\lbrace</code> | 2366 |
| | | <code>\lcirclearrow</code> | 2869 |
| | | <code>\Leftarrow</code> | 2789 |
| | | <code>\leftarrowtail</code> | 2804 |
| | | <code>\leftarrowto</code> | 2813 |
| | | <code>\Leftbararrow</code> | 2796 |
| | | <code>\leftbararrow</code> | 2794, 2795 |
| | | <code>\leftbrace</code> | 2353, 2408 |
| | | <code>\leftdasharrow</code> | 2801 |
| | | <code>\leftharpoondown</code> | 2803 |
| | | <code>\leftharpoonup</code> | 2802 |
| | | <code>\leftleftarrows</code> | 2815 |
| | | <code>\leftleftleftarrows</code> | 2816 |
| | | <code>\leftoplusarrow</code> | 2805 |
| | | <code>\Leftrightarrow</code> | 2818 |
| | | <code>\leftrightarrows</code> | 2823 |
| | | <code>\leftrightharpoonrightarrow</code> | 2825 |
| | | <code>\leftrightharpoons</code> | 2824 |
| | | <code>\leftrightharpoonrightarrow</code> | 2822 |
| | | <code>\leftsquigarrow</code> | 2807 |
| | | <code>\leftwavearrow</code> | 2806 |
| | | <code>\leftwhitearrow</code> | 2814 |
| | | <code>\leq</code> | 2516 |
| | | <code>\leqq</code> | 2646 |
| | | <code>\lguil</code> | 1397, 1856, 2372, 2404 |
| | | <code>\llap</code> | 2430 |
| | | <code>\Lleftarrow</code> | 2791 |
| | | <code>\llguil</code> | 1399, 1858, 2378, 2406 |
| | | <code>\lll</code> | 2644 |
| | | <code>\lnapprox</code> | 2716 |

- `\lneq` 2699
 - `\lneqq` 2701
 - `\lnsim` 2714
 - local font changes 26
 - log file 19, 23, 30, 31
 - `\Longleftbararrow` 2799
 - `\longleftbararrow` 2797, 2798
 - `\longleftsquigarrow` 2808
 - `\Longrightbararrow` 2768
 - `\longrightbararrow` 2766, 2767
 - `\longrightsquigarrow` 2777
 - `\looparrowleft` 2809
 - `\looparrowright` 2778
 - `\lsim` 2668
- M**
- `\M@greeklower@set` 23, 997, **2159**
 - `\M@greeklowershape` 460, 2160
 - `\M@greekupper@set` 23, 996, **2147**
 - `\M@greekuppershape` 459, 2148
 - `\M@arrows@set` 23, 1006, **2738**
 - `\M@arrowsshape` 472, 2739
 - `\M@BadIntegerError` **347**, 867, 874, 881, 888
 - `\M@BadMathConstantsFontError` .. **282**, 733
 - `\M@BadMathConstantsFontTypeError` ..
..... **289**, 744
 - `\M@bb@mathcodes` 2878, **2882**
 - `\M@bb@num` **2881**, 2883–2944
 - `\M@bb@set` 23, 1009, **2871**
 - `\M@bbshape` 473, 2881
 - `\M@bcal@mathcodes` 3080, **3084**
 - `\M@bcal@num` **3083**, 3085–3136
 - `\M@bcal@set` 23, 1012, **3073**
 - `\M@bcalshape` 476, 3083
 - `\M@bfrak@mathcodes` 3144, **3148**
 - `\M@bfrak@num` **3147**, 3149–3200
 - `\M@bfrak@set` 23, 1013, **3137**
 - `\M@bfrakshape` 477, 3147
 - `\M@bigops@set` 23, 1007, **2439**
 - `\M@bigopsshape` 468, 2440
 - `\M@cal@mathcodes` 2952, **2956**
 - `\M@cal@num` **2955**, 2957–3008
 - `\M@cal@set` 23, 1010, **2945**
 - `\M@calshape` 474, 2955
 - `\M@Charm` 416, 1068, 1070, 1072, 1076
 - `\M@CharsSetWarning` **211**, 676
 - `\M@check@csarg` **773**, 786, **795**
 - `\M@check@int` **832**, 863, 870, 877, 884
 - `\M@check@option@valid` **601**, 631
 - `\M@check@suboption@valid` **614**, 641
 - `\M@cyrilliclower@set` 23, 999, **2205**
 - `\M@cyrilliclowershape` **462**, 2206
 - `\M@cyrillicupper@set` 23, 998, **2171**
 - `\M@cyrillicuppershape` **461**, 2172
 - `\M@Decl@reF@milyB@settrue` 556
 - `\M@DecSymDef` **397**, 399, 404
 - `\M@default@newmath@cmds` .. **797**, 806, **816**
 - `\M@defaultkeys` **483**, 486, **486**, 655
 - `\M@define@newmath@cmd` **793**, 806, 815
 - `\M@delimiters@set` . 23, 1005, **2344**, **2398**
 - `\M@delimitersshape` **466**, 2345, 2399
 - `\M@DeprecatedWarning` **216**, 828, 830
 - `\M@diacritics@set` 23, 993, **2061**
 - `\M@diacriticsshape` **456**, 2062
 - `\M@digits@font` **2269**, 2270–2279
 - `\M@digits@set` 23, 1001, **2268**
 - `\M@digitsshape` **464**, 2269
 - `\M@DoubleArgError` **305**, 782
 - `\M@entries@assert` 1167, 1183, 1200
 - `\M@extbigops@set` 23, 1008, **2446**
 - `\M@extbigopsshape` **469**, 2447
 - `\M@extsymbols@set` 23, 1004, **2557**
 - `\M@extsymbolsshape` **471**, 2558
 - `\M@fill@nfss@shapes` **495**, 578, 585, 597, 599
 - `\M@FontChangeInfo` **205**, 701
 - `\M@ForbiddenCharmFile` ... **330**, 1048, 1056
 - `\M@ForbiddenCharmLine` ... **321**, 1050, 1058
 - `\M@frak@mathcodes` 3016, **3020**
 - `\M@frak@num` **3019**, 3021–3072
 - `\M@frak@set` 23, 1011, **3009**
 - `\M@frakshape` **475**, 3019
 - `\M@greeklower@set` 23, 995, **2114**
 - `\M@greeklowershape` **458**, 2115
 - `\M@greekupper@set` 23, 994, **2074**
 - `\M@greekuppershape` **457**, 2075
 - `\M@hebrew@set` 23, 1000, **2239**
 - `\M@hebrewshape` **463**, 2240
 - `\M@HModeError`
..... **313**, 2874, 2948, 3012, 3076, 3140
 - `\M@index@assert` 1151, 1155
 - `\M@integral@italic@factor` . 412, 420, 872
 - `\M@InternalsRestoredError` **247**, 658
 - `\M@InvalidOptionError` **220**, 602
 - `\M@InvalidSuboptionError` **228**, 615
 - `\M@keys` **478**, 603

- `\M@lower@set` 23, 992, **1997**, **2029**
`\M@lowershape` **455**, 1998, 2030
`\M@LuaTeXOnlyWarning` **295**, 769
`\M@MissingControlSequenceError` **299**, 778
`\M@MissingOptionError` **236**, 629
`\M@MissingSuboptionError` **241**, 635
`\M@newfont` **536**, 600, 661, 788
`\M@NewFontCommandInfo` **207**, 789
`\M@NoFontAdjustError` **339**, 894
`\M@NoFontspecError` **272**, 549
`\M@NoFontspecFamilyError` **261**, 543
`\M@NoLuaotfloadError` **92**, 120
`\M@NoMathfontError` **9**,
15, 17, 18, 20–29, 31, 33, 35, 37, 39–41
`\M@number@assert` 1135, 1146
`\M@operator@mathcodes` . **2283**, **2339**, 2342
`\M@operator@num` **2282**, 2284–2337
`\M@operator@set` 23, 1002, **2280**
`\M@operator@shape` **465**, 2282, 2342
`\M@Optiondeprecated` . . **124**, 131, 134, 137
`\M@otf@features` . . **488**, 499, 504, 509, 514
`\M@otf@features@sc` **490**, 519, 524, 529, 534
`\M@p@tch@decl@re` **396**, 397
`\M@parse@option` **623**, 672
`\M@radical@set` 23, **2412**, **2435**
`\M@radical@shape` **467**, 2413, 2436
`\M@retokenize` **404**, 405, **405**, 407
`\M@rule@thickness@factor` . . 411, 419, 865
`\M@SetMathConstants` **727**, **752**, 767
`\M@split@colon` **492**, 538
`\M@strip@colon` **494**, 564
`\M@strip@equals` **622**, 640
`\M@surd@horizontal@factor` . 414, 421, 879
`\M@surd@vertical@factor` . . . 413, 422, 886
`\M@SymbolFontInfo` **200**, 694
`\M@symbols@set` 23, 1003, **2477**
`\M@symbolsshape` **470**, 2478
`\m@th@const@nts@font` **751**, 756, 759
`\m@th@const@nts@font@sh@pe`
. **738**, **742**, 749, 761
`\m@thf@nt` 718
`\M@upper@set` 23, 991, **1938**, **1967**
`\M@upper@shape` **454**, 1939, 1968
`\M@XeTeXLuaTeXError` **51**, 73
`\math@fonts` 764, **767**
`\mathand` 2489, 2532
`\mathbackslash` 2362, 2363
`\mathbb` **2872**, 2874
`\mathbcal` **3074**, 3076
`\mathbf` 820
`\mathbfit` 821
`\mathbffrac` **3138**, 3140
`\mathbfsc` 824
`\mathbfscit` 825
`\mathcal` **2946**, 2948
`\mathconstantsfont` 18, **18**, 284,
288, 291, 292, 296, 722, **728**, **769**, 772
`\mathdollar` 2487
`\mathfont` 15, **15**, 135,
138, 214, 221, 223, 229, 231, 237, 239,
242, 246, 253, 255, 257, 259, 263, 271,
274, 281, 287, **655**, 708, 717, 721, 945
`\mathfrac` **3010**, 3012
`\mathhash` 2486, 2530
`\mathit` 819
`\mathng` 2733, 2734
`\mathnolimitsmode` 1015
`\mathpalette` 2537, 2538
`\mathparagraph` 2490
`\mathpercent` 2488, 2531
`\mathring` 2072
`\mathrm` 818
`\mathsc` 822
`\mathscit` 823
`\mathsection` 2491
`\mathsterling` 2492
`\meaning` 390, 393
`\mem` 2253
`\mid` 2521
Missing \$ inserted 10
Missing control sequence 10
Missing Option for `\mathfont` 8
Missing package `fontspec` 8
Missing Suboption for `\mathfont` 8
missing X_YTeX or LuaTeX 3
`\mkern` 2433
`\models` 2542
`\Mu` 1834, 2087
Multiple characters in argument 10
- N**
- `\nabla` 2104, 2111, 2555, 2736
`\napprox` 2708
`\natural` 2582
`\Nearrow` 2860

<code>\nearrow</code>	2859	<code>\nwsearrow</code>	2867
<code>\neg</code>	2493	O	
<code>\nequiv</code>	2730	<code>\odiv</code>	2614
<code>\neswarrow</code>	2868	<code>\oiintop</code>	2470, 2471
<code>\newmathbf</code>	22, 22 , 23, 800, 809, 820, 828	<code>\ointop</code>	2468, 2469
<code>\newmathbffit</code>	23 , 801, 810, 821, 830	<code>\ointop</code>	2466, 2467
<code>\newmathbfsc</code>	28, 28 , 804, 813, 824	<code>\Omega</code>	1846, 2099
<code>\newmathbfscit</code>	29, 29 , 805, 814, 825	<code>\omega</code>	1816, 2139
<code>\newmathbold</code>	24, 24 , 25, 827 , 828	<code>\Omicron</code>	1837, 2090
<code>\newmathboldit</code>	25 , 829 , 830	<code>\omicron</code>	1807, 2130
<code>\newmathfontcommand</code>	<code>\ominus</code>	2613
.....	30 , 31, 785 , 786, 792, 796	<code>\operator@font</code>	2341
<code>\newmathit</code>	21, 21 , 799, 808, 819	<code>\oslash</code>	2615
<code>\newmathrm</code>	20, 20 , 798, 807, 818	P	
<code>\newmathsc</code>	26, 26 , 802, 811, 822	Package <code>mathfont</code> Info	7
<code>\newmathscit</code>	27, 27 , 803, 812, 823	<code>\parallel</code>	2522
<code>\ngeq</code>	2698	parse <code>\mathfont</code> arguments	20
<code>\ngsim</code>	2713	<code>\partial</code>	2496
<code>\nLeftarrow</code>	2790	<code>\PassOptionsToPackage</code>	362
<code>\nleftarrow</code>	2788	<code>\Phi</code>	1843, 2096
<code>\nLeftrightarrow</code>	2819	<code>\phi</code>	1813, 2136
<code>\nleq</code>	2697	<code>\Pi</code>	1838, 2091
<code>\nlsim</code>	2712	<code>\pi</code>	1808, 2131
no previous font	8	<code>\pm</code>	2507
<code>\nprec</code>	2718	<code>\prec</code>	2670
<code>\npreceq</code>	2720	<code>\precapprox</code>	2678
<code>\nrightarrow</code>	2759	<code>\preceq</code>	2672
<code>\nrightrightarrow</code>	2757	<code>\preceqq</code>	2674
<code>\nsim</code>	2707	<code>\precnapprox</code>	2728
<code>\nsimeq</code>	2709	<code>\precneq</code>	2722
<code>\nsimeqq</code>	2710	<code>\precneqq</code>	2724
<code>\nsubseteq</code>	2691	<code>\precnsim</code>	2726
<code>\nsubseteqq</code>	2692	<code>\preccurlyeq</code>	2680
<code>\subset</code>	2685	<code>\precsim</code>	2676
<code>\subseteq</code>	2687	<code>\prime</code>	2484
<code>\succ</code>	2719	<code>\proportion</code>	2641
<code>\succeq</code>	2721	<code>\propto</code>	2635
<code>\supset</code>	2686	<code>\Psi</code>	1845, 2098
<code>\supseteq</code>	2688	Q	
<code>\ntriangleleft</code>	2703	<code>\qeq</code>	2667
<code>\ntrianglelefteq</code>	2705	<code>\qof</code>	2259
<code>\ntriangleright</code>	2704	R	
<code>\ntrianglerighteq</code>	2706	<code>\r@t</code>	2417
<code>\Nu</code>	1835, 2088	<code>\radicandoffset</code>	415, 423, 2433
<code>\nun</code>	2254	<code>\ratio</code>	2640
<code>\Nwarrow</code>	2862		
<code>\nwarrow</code>	2861		

<code>\rbrace</code>	2369	<code>\setmathfontcommands</code>	723, 817 , 826
<code>\rcirclearrow</code>	2870	<code>\setminus</code>	2512
<code>\Relbar</code>	2541, 2542	<code>\sharp</code>	2583
<code>\relbar</code>	2540	<code>\shin</code>	2261
<code>\resh</code>	2260	<code>\Sho</code>	2154
<code>\restoremathinternals</code> .	132, 256, 260, 904	<code>\Sigma</code>	1840, 2093
<code>\rguil</code>	1398, 1857, 2375, 2405	<code>\sigma</code>	1810, 2133
<code>\Rho</code>	1839, 2092	<code>\sim</code>	2518, 2537, 2538
<code>\rho</code>	1809, 2132	<code>\simeq</code>	2537, 2560 , 2650
<code>\Rightarrow</code>	2758	<code>\simeqq</code>	2652, 2653
<code>\rightarrowtail</code>	2773	<code>\simneqq</code>	2711
<code>\rightarrowto</code>	2782	<code>\sqcap</code>	2602
<code>\Rightbararrow</code>	2765	<code>\sqdot</code>	2620
<code>\rightbararrow</code>	2763, 2764	<code>\sqminus</code>	2619
<code>\rightbrace</code>	2354, 2409	<code>\sqplus</code>	2617
<code>\righdasharrow</code>	2770	<code>\sqrtsign</code>	2418, 2433
<code>\rightharpoondown</code>	2772	<code>\sqsubseteq</code>	2629
<code>\rightharpoonup</code>	2771	<code>\sqsubsetneq</code>	2693
<code>\rightleftarrows</code>	2826	<code>\sqsupseteq</code>	2630
<code>\rightoplusarrow</code>	2774	<code>\sqsupsetneq</code>	2694
<code>\rightrightarrows</code>	2784	<code>\sqtimes</code>	2618
<code>\rightrightarrows</code>	2785	<code>\ssharp</code>	2585
<code>\rightsquigarrow</code>	2776	<code>\sssimeq</code>	2655
<code>\rightwvearrow</code>	2775	<code>\st@ck@fl@trel</code>	918, 919
<code>\rightwhitearrow</code>	2783	<code>\stack@flatrel</code>	917 , 2537, 2538
<code>\ringeq</code>	2660	<code>\star</code>	2607
robust commands	78	<code>\stareq</code>	2664
<code>\rootbox</code>	2425, 2428, 2430	<code>\Stigma</code>	2153
<code>\rrguil</code>	1400, 1859, 2381, 2407	<code>\stigma</code>	2165
<code>\Rrightarrow</code>	2760	<code>\strip@prefix</code>	390, 393
<code>\RuleThicknessFactor</code>		suboption <i>italic</i>	20, 23
.....	32 , 33, 862 , 867, 891, 898	suboption <i>roman</i>	20, 23
		suboption <i>upright</i>	20
		<code>\subseteq</code>	2625
		<code>\subsetneq</code>	2689
		<code>\succ</code>	2671
		<code>\succapprox</code>	2679
		<code>\succeq</code>	2673
		<code>\succeqq</code>	2675
		<code>\succnapprox</code>	2729
		<code>\succneq</code>	2723
		<code>\succneqq</code>	2725
		<code>\succnsim</code>	2727
		<code>\succsim</code>	2677
		<code>\succsucc</code>	2681
		<code>\supseteq</code>	2626
		<code>\supsetneq</code>	2690
S			
<code>\samekh</code>	2255		
<code>\Sampi</code>	2150		
<code>\sampi</code>	2162		
<code>\San</code>	2155		
<code>\san</code>	2167		
<code>\scantextokens</code>	399		
<code>\scantokens</code>	404		
<code>\scdefault</code>	516,		
.....	518, 521, 523, 526, 528, 531, 533, 802–805		
<code>\Searrow</code>	2864		
<code>\searrow</code>	2863		
<code>\seq</code>	2656		
<code>\seriesdefault</code>	760		
<code>\setfont</code>	17, 17 , 720 , 725, 945, 989		

`\surd` 1887, 2414, 2437
`\surdbox`
 409, 2419, 2421, 2422, 2424–2426, 2431
`\SurdHorizontalFactor`
 **38**, 39, **876**, 881, 891, 900
`\SurdVerticalFactor`
 **36**, 37, **883**, 888, 892, 901
`\Swarrow` 2866
`\swarrow` 2865
`\symMupright` 758

T

`\Tau` 1841, 2094
`\tau` 1811, 2134
`\tav` 2262
`terminal` 23, 30
`\tet` 2249
`\textbackslash` 2362
`\textng` **2731**, 2734
`\therefore` 2638
`\Theta` 1830, 2083
`\theta` 1800, 2123
`\tilde` 2073
`\tracinglostchars` 933, 934
`\triangleeq` 2665
`\triangleleft` 2631
`\trianglelefteq` 2633
`\triangleright` 2632
`\trianglerighteq` 2634
`\tsadi` 2258
`\twoheaddownarrow` 2849
`\twoheadleftarrow` 2812
`\twoheadrightarrow` 2781
`\twoheaduparrow` 2835

U

`\Udelcode` 2355–2360
`\Udelimiter` ... 2364, 2367, 2370, 2373,
 2376, 2379, 2382, 2385, 2388, 2391, 2394
 unable to load 3, 4
`\unexpanded` 404
`\uparrowto bar` 2836
`\upbararrow` 2831
`\updasharrow` 2832
`\updownarrows` 2855
`\updownharpoons` 2857
`\upharpoonleft` 2833
`\upharpoonright` 2834

`\Upsilon` 1842, 2095
`\upsilon` 1812, 2135
`\upuparrows` 2839
`\upwhitearrow` 2837
`\upwhitebararrow` 2838
`\Uradical` 2416
`\Uparrow` 2830

V

`\varbeta` 1817, 2140
`\vardot` 2609
`\varDigamma` 2157
`\vardigamma` 2169
`\varepsilon` 1818, 2141
`\varkaf` 2263
`\varkappa` 2142
`\varKoppa` 2158
`\varkoppa` 2170
`\varmem` 2264
`\varnun` 2265
`\varpe` 2266
`\varphi` 1822, 2146
`\varrho` 1820, 2144
`\varSampi` 2156
`\varsampi` 2168
`\varsetminus` 2610
`\varsigma` 1821, 2145
`\varTheta` 1847, 2100
`\vartheta` 1819, 2143
`\vartsadi` 2267
`\vav` 2246
`\vdash` 2579
`\veeeq` 2663
`\vert` **2361**
`\vphantom` 2420

W

`\wclubsuit` 2590
`\wdiamondsuit` 2591, 2596
`\wedgreek` 2662
`\wheartsuit` 2592, 2595
`\wp` 2566
`\wspadesuit` 2593

X

`\XeTeXrevision` 45, 936
`\Xi` 1836, 2089

Y

`\yod` 2250
Your command is invalid without Lua-based 10
Your `\mathconstants` on line 9

Z

`\zayin` 2247
`\Zeta` 1828, 2081
`\zeta` 1798, 2121
`\zigzagarrow` 2845, 2846