Package **mathfont** v. 2.2 Implementation

Conrad Kosowsky

December 2022

`kosowsky.latex@gmail.com`

For easy, off-the-shelf use, type the following in your preamble and compile with X$_{\overline{E}}$LAT_{E}X or LuaLAT_{E}X:

`\usepackage[`⟨*font name*⟩`]{mathfont}`

As of version 2.0, using LuaLAT_{E}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 LuaT$_{E}$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}[2022/12/04 v. 2.2 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?
4 \newif\ifM@Noluaotfload   % cannot find luaotfload.sty?
```

Acknowledgements: Thanks to Lyric Bingham for her work checking my unicode hex values. Thanks to 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 their **mathastext** package.

1

```
 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\newmathbfit{\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_ETEX or LuaTEX. 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 X$_{\text{E}}$T$_{\text{E}}$X or LuaT$_{\text{E}}$X (and cannot load OpenType fonts) and one if T$_{\text{E}}$X cannot find the luaotfload package. In this case, the package will stop loading, so we want a particularly conspicuous error message.

The error message itself is organized as follows. For each message, we check the appropriate conditional to determine if we need to raise the error. If yes, we change + to active and define it to be equal to a space character. We use + to print multiple spaces inside the error message, and we put the catcode change inside a group to keep it local. We define a \GenericError inside a macro and then call the macro for a cleaner error message. The \@gobbletwo eats the extra period and return that L$^{\!A}$T$_{\text{E}}$X adds to the error message. Notice that we \endgroup immediately after issuing the error—this is because we need \M@NoFontspecError to both tokenize its definition and then evaluate while + has catcode 13. Otherwise, T$_{\text{E}}$X will issue an \inaccessible error. However, we want \AtBeginDocument and \endinput outside the group. The \expandafter means that we expand the final \fi before \endinput, which balances the original conditional.

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

```
76   \AtEndOfPackage{\typeout{:: mathfont :: Failed to load\on@line.}}
77   \expandafter\endinput % we should \endinput with a balanced conditional
78 \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.

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

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

```
90 \ifM@Noluaotfload % false by default; true if LuaTeX AND no luaotfload.sty
91   \begingroup
92     \catcode`\+=\active
93     \def+{ }
94     \def\M@NoluaotfloadError{\GenericError{}
95       {\MessageBreak\MessageBreak
96       Package mathfont error:
97       \MessageBreak\MessageBreak
98       +***********************\MessageBreak
99       +*+++++++++++++++++++++*\MessageBreak
100      +*+++++++UNABLE TO+++++++*\MessageBreak
101      +*+++++LOAD MATHFONT+++++*\MessageBreak
102      +*+++++++++++++++++++++*\MessageBreak
103      +*++++Cannot find the++++*\MessageBreak
104      +*+++file luaotfload.sty++*\MessageBreak
105      +*+++++++++++++++++++++*\MessageBreak
106      +***********************\MessageBreak\@gobbletwo}
107      {You are likely seeing this message because you haven't^^J%
108      installed luaotfload. Check your TeX distribution for a^^J%
109      list of the packages on your system. See the mathfont^^J%
110      documentation for further explanation.^^J}
111      {It looks like the current engine is LuaTeX, so I\MessageBreak
112      need the luaotfload package to make mathfont work\MessageBreak
```

```
113        correctly. I can't find luaotfload, so I'm going to\MessageBreak
114        stop reading in the mathfont package file now. (You\MessageBreak
115        won't be able to use commands from mathfont in your\MessageBreak
116        document.) To make mathfont work correctly, make\MessageBreak
117        sure luaotfload.sty is installed on your computer\MessageBreak
118        in a directory searchable by TeX or compile with\MessageBreak
119        XeLaTeX.^^J}}
120      \M@NoluaotfloadError
121    \endgroup
122    \AtEndOfPackage{\typeout{:: mathfont :: Failed to load\on@line.}}
123    \expandafter\endinput % we should \endinput with a balanced conditional
124 \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.

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

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

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

Easter egg!

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

```
153        lure an egg?^^J^^J}
154      \EasterEggUpdate
155    \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
156        Still wrangling. Try back later.^^J^^J}
157    \AtBeginDocument{\bgroup
158      \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J:%
159        If we have zero eggs^^J%
160        and zero bunnies, how^^J%
161        many gnats does it take^^J%
162        to change a lightbulb??^^J^^J}
163      \EasterEggUpdate
164      \egroup}
165    \AtEndDocument{%
166      \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
167        Happy, happy day! Happy,^^J%
168        happy day! Clap your hands,^^J%
169        and be glad your hovercraft^^J%
170        isn't full of eels!^^J^^J}
171      \EasterEggUpdate
172      \let\E@sterEggUpd@te\relax
173      \let\EasterEggUpdate\relax}
174  \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.

```
175 \DeclareOption{adjust}{\M@adjust@fonttrue}
176 \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`.

```
177 \DeclareOption*{\M@font@loadedtrue\edef\M@font@load{\CurrentOption}}
178 \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 LuaTeX, so we print a message depending on `\ifM@adjust@font`.

```
179 \ifdefined\directlua
180   \ifM@adjust@font
181     \AtEndOfPackage{%
182       \typeout{:: mathfont :: Lua-based font adjustments enabled.}}
183   \else
184     \AtEndOfPackage{%
185       \typeout{:: mathfont :: Lua-based font adjustments disabled.}}
186   \fi
187 \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.

```
188   \AtEndOfPackage{%
189     \typeout{:: mathfont :: Lua-based font adjustments disabled.}}
```

```
190  \ifM@adjust@font
191    \AtEndOfPackage{%
192    \PackageError{mathfont}{Option^^J"adjust" ignored with XeTeX}
193    {Your package option "adjust" was ignored.\MessageBreak
194    This option works only with LuaTeX, and it\MessageBreak
195    looks like the current engine is XeTeX. To\MessageBreak
196    enable Lua-based font adjustments, compile\MessageBreak
197    with LuaLaTeX.^^J}}
198    \M@adjust@fontfalse
199  \fi
200 \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.

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

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

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

Error messages associated with \mathfont.

```
221 \def\M@InvalidOptionError#1{\PackageError{mathfont}
222    {Invalid^^Joption "#1" for \string\mathfont\on@line}
223    {Hm. You used a keyword that isn't actually an optional\MessageBreak
224    argument for \string\mathfont. Check
225    that you spelled the keyword\MessageBreak
226    correctly. Otherwise, I'm not sure what's wrong. Is this\MessageBreak
227    option listed in the package documentation? In any event,\MessageBreak
```

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

Table 1: Various Messages and Errors and Their Uses

Command	Use
\M@FontChangeInfo	Use a symbol font for some characters
\M@NewFontCommandInfo	Declare new alphanumeric font-change command
\M@SymbolFontInfo	Declare new symbol font
\M@CharsSetWarning	Warning when calling \mathfont multiple times for same keyword
\M@InternalsRestoredError	User called \mathfont after restoring kernel
\M@InvalidOptionError	Bad option for \mathfont
\M@InvalidSupoptionError	Bad suboption for \mathfont
\M@MissingOptionError	Missing an option for \mathfont
\M@MissingSuboptionError	Missing suboption for \mathfont
\M@BadMathConstantsFontError	Argument not previously fed to \mathfont
\M@BadMathConstantsFontTypeError	Argument not "upright" or "italic"
\M@LuaTeXOnlyWarning	User called \mathcontsantsfont in X$_\text{H}$TEX
\M@DeprecatedWarning	Warning for certain deprecated macros
\M@DoubleArgError	Gave multiple tokens to be the font-change macro
\M@HModeError	Font-change command used outside math mode
\M@MissingControlSequenceError	No macro provided to be font-change command
\M@NoFontspecFamilyError	Improper option fontspec for \mathfont
\M@NoFontspecError	Option fontspec for \mathfont declared without having loaded fontspec
\M@BadIntegerError	Font metric adjustment value was not an integer
\M@ForbiddenCharmFile	Charm file contains a bad character
\M@ForbiddenCharmLine	Charm line contains a bad character
\M@NoFontAdjustError	Command called when Lua-based font adjustment was disabled

```
242 \def\M@MissingSuboptionError{\PackageError{mathfont}
243   {Missing^^Jsuboption for \string\mathfont\on@line}
244   {It looks like you included an = somewhere\MessageBreak
245   but didn't put the suboption after it. Either\MessageBreak
246   that or you typed == instead of = in the\MessageBreak
247   optional argument of \string\mathfont.^^J}}
248 \def\M@InternalsRestoredError{\PackageError{mathfont}
249   {Internal^^Jcommands restored}
250   {This package slightly changes two LaTeX\MessageBreak
251   internal commands, and you really shouldn't\MessageBreak
252   be loading new math fonts without those\MessageBreak
253   adjustments. What happened here is that you\MessageBreak
254   used \string\mathfont\space in a situation where those\MessageBreak
255   two commands retain their original defini-\MessageBreak
256   tions. Presumably you used \string\mathfont\space after\MessageBreak
257   calling the \string\restoremathinternals\space command.\MessageBreak
258   I'm going to ignore this call to \string\mathfont.\MessageBreak
259   Try typesetting this document with all\MessageBreak
260   \string\mathfont\space commands placed before you call\MessageBreak
261   \string\restoremathinternals.^^J}}
262 \def\M@NoFontspecFamilyError{\PackageError{mathfont}
263   {No previous^^Jfont loaded by fontspec}
264   {You called \string\mathfont\space
265   with the argument "fontspec" \MessageBreak
266   on line \the\inputlineno,
267   and that tells me to use the previous \MessageBreak
268   font loaded by the fontspec package. However, it \MessageBreak
269   looks like you haven't loaded any fonts yet with \MessageBreak
270   fontspec. To resolve this error, try using for \MessageBreak
271   example \string\setmainfont\space
272   before calling \string\mathfont.^^J}}
273 \def\M@NoFontspecError{\PackageError{mathfont}
274   {Missing^^Jpackage fontspec}
275   {You called \string\mathfont\space
276   with the argument "fontspec" \MessageBreak
277   on line \the\inputlineno,
278   and that tells me to use the previous \MessageBreak
279   font loaded by the fontspec package. However, you\MessageBreak
280   haven't loaded fontspec, so some things are about\MessageBreak
281   to get messed up. To resolve this error, load\MessageBreak
282   fontspec before calling \string\mathfont.^^J}}
```

Error messages for \mathconstantsfont.

```
283 \def\M@BadMathConstantsFontError#1{\PackageError{mathfont}
284   {Invalid\MessageBreak font specifier for
285   \string\mathconstantsfont:\MessageBreak"#1"}
286   {Your command was ignored--I can't parse your argument.\MessageBreak
287   Please make sure to use text that you have previously\MessageBreak
```

```
288   fed to \string\mathfont\space for the argument of
289   \string\mathconstantsfont.^^J}}
290 \def\M@BadMathConstantsFontTypeError#1{\PackageError{mathfont}
291   {Invalid\MessageBreak font specifier for
292   \string\mathconstantsfont:\MessageBreak"#1"}
293   {The optional argument of \string\mathconstantsfont\MessageBreak
294   should be "upright" or "italic." Right now,\MessageBreak
295   it's "#1."^^J}}
296 \def\M@LuaTeXOnlyWarning{\PackageWarning{mathfont}
297      {Your \string\mathconstantsfont\space
298      on line \the\inputlineno\space is\MessageBreak
299      for LuaTeX only, and I'm ignoring it\@gobble}}
```

Error messages for the \newmathrm, etc. commands.

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

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

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

```
333  {One of the lines in your \string\CharmFile\space
334  from line \the\inputlineno\MessageBreak
335  contains the character #1, which will mess me up\MessageBreak
336  if I try to read it, so I'm ignoring this line\MessageBreak
337  from your file. To resolve this error, make sure\MessageBreak
338  your charm information contains only integers,\MessageBreak
339  floats, asterisks, commas, and spaces.^^J}}
340 \def\M@NoFontAdjustError#1{\PackageError{mathfont}
341  {Your command \MessageBreak\string#1 is invalid\MessageBreak
342  without Lua-based font adjustments}
343  {You haven't enabled Lua-based font adjustments,\MessageBreak
344  but the macro you called won't do anything without\MessageBreak
345  them. I'm going to ignore your command for now. To\MessageBreak
346  resolve this error, load mathfont with the package\MessageBreak
347  option "adjust" or compile with LuaLaTeX.^^J}}
348 \def\M@BadIntegerError#1#2{\PackageError{mathfont}
349  {Bad argument for\MessageBreak\string#1}
350  {Your command was ignored. Please make sure\MessageBreak
351  that your argument of \string#1\space\MessageBreak
352  is a nonnegative integer. Right now it's\MessageBreak
353  "#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.

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

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

```
364 \let\@@set@mathchar\set@mathchar
365 \let\@@set@mathsymbol\set@mathsymbol
366 \let\@@set@mathaccent\set@mathaccent
367 \let\@@DeclareSymbolFont\DeclareSymbolFont
368 \@onlypreamble\@@set@mathchar
```

```
369 \@onlypreamble\@@set@mathsymbol
370 \@onlypreamble\@@set@mathaccent
371 \@onlypreamble\@@DeclareSymbolFont
372 \wlog{Package mathfont Info: Adapting \noexpand\set@mathchar for unicode.}
373 \wlog{Package mathfont Info: Adapting \noexpand\set@mathsymbol for unicode.}
374 \wlog{Package mathfont Info: Adapting \noexpand\set@mathaccent for unicode.}
375 \wlog{Package mathfont Info: Increasing upper bound on
376    \noexpand\DeclareSymbolFont to 256.}
```

Kernel command to set math characters from keystrokes.

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

Kernel command to set math characters from control sequences.

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

Kernel command to set accents.

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

We increase the upper bound on the number of symbol fonts to be 256. LuaTeX and XƎTeX allow up to 256 math families, but the LATEX 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 XƎTeX and LuaTeX. 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 LATEX 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`.

```
390 \ifx\DeclareSymbolFont@m@dropped\@undefined
391    \edef\@tempa{\expandafter\strip@prefix\meaning\DeclareSymbolFont}
392    \def\@tempb{\def\DeclareSymbolFont##1##2##3##4##5}
393 \else
394    \edef\@tempa{\expandafter\strip@prefix\meaning\DeclareSymbolFont@m@dropped}
395    \def\@tempb{\def\DeclareSymbolFont@m@dropped##1##2##3##4##5}
```

396 `\fi`
397 `\def\M@p@tch@decl@re#1<15#2\@nil{#1<\e@mathgroup@top#2}`
398 `\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 LuaTeX and a safe version of `\scantokens` in XƎTeX. 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`.

399 `\ifdefined\directlua`
400   `\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`.

401 `\else`
402   `\begingroup`
403   `\everyeof{\noexpand}`
404   `\endlinechar\m@ne`

The first `\edef` expands `\M@DecSymDef` and defines `\M@retokenize` to be `\scantokens{`
`\unexpanded{⟨new definition⟩}}`, 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`.

405   `\edef\M@retokenize{\noexpand\scantokens{\noexpand\unexpanded{\M@DecSymDef}}}`
406   `\edef\M@retokenize{\M@retokenize}`
407   `\expandafter\endgroup`
408   `\expandafter\@tempb\expandafter{\M@retokenize}`
409 `\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.

410 `\newbox\surdbox`
411 `\newcount\M@count`
412 `\newcount\M@rule@thickness@factor`
413 `\newcount\M@integral@italic@factor`
414 `\newcount\M@surd@vertical@factor`
415 `\newcount\M@surd@horizontal@factor`
416 `\newmuskip\radicandoffset`
417 `\newread\M@Charm`
418 `\newtoks\M@toks`
419 `\M@count\z@`

```
420 \M@rule@thickness@factor\@m
421 \M@integral@italic@factor=400\relax
422 \M@surd@horizontal@factor\@m
423 \M@surd@vertical@factor\@m
424 \radicandoffset=3mu\relax
```

Necessary booleans and default math font shapes.

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

Default shapes.

```
455 \def\M@uppershape{italic} % latin upper
456 \def\M@lowershape{italic} % latin lower
457 \def\M@diacriticsshape{upright} % diacritics
458 \def\M@greekuppershape{upright} % greek upper
459 \def\M@greeklowershape{italic} % greek lower
460 \def\M@agreekuppershape{upright} % ancient greek upper
461 \def\M@agreeklowershape{italic} % ancient greek lower
462 \def\M@cyrillicuppershape{upright} % cyrillic upper
463 \def\M@cyrilliclowershape{italic} % cyrillic lower
464 \def\M@hebrewshape{upright} % hebrew
```

```
465 \def\M@digitsshape{upright} % numerals
466 \def\M@operatorshape{upright} % operator font
467 \def\M@delimitersshape{upright} % delimiters
468 \def\M@radicalshape{upright} % surd
469 \def\M@bigopsshape{upright} % big operators
470 \def\M@extbigopsshape{upright} % extended big operators
471 \def\M@symbolsshape{upright} % basic symbols
472 \def\M@extsymbolsshape{upright} % extended symbols
473 \def\M@arrowsshape{upright} % arrows
474 \def\M@bbshape{upright} % blackboard bold
475 \def\M@calshape{upright} % caligraphic
476 \def\M@frakshape{upright} % fraktur
477 \def\M@bcalshape{upright} % bold caligraphic
478 \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.

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

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

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

Default OpenType features for loading fonts.

```
489 \def\M@otf@features{script=latin;language=DFLT;%
490   tlig=true;liga=true;smcp=false;lnum=true}
491 \def\M@otf@features@sc{script=latin;language=DFLT;%
492   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 a 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@f@ntn@me` and
`\M@f@nt@n@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.

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

The macro `\M@fill@nfss@shapes` accepts two arguments and does the actual work of en-
suring that the NFSS contains the appropriate series and shapes. The first argument should
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`.

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

Upright shape.

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

Italic shape.

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

Bold series with upright shape.

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

Bold series with italic shape.

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

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

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

The main font-loading macro. This macro takes a single argument, which should have the form ⟨*font name*⟩:⟨*optional features*⟩, 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`.

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

If the argument is "`fontspec`," we want to use the last font loaded by fontspec, which is stored in `\l_fontspec_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_fontspec_family_tl` is empty or if the user has not loaded fontspec. If we use fontspec to laod the font, we don't get a separate font in base mode.

```
542   \ifx\@tempa\@tempb
543     \@ifpackageloaded{fontspec}{%
544       \expandafter\ifx\csname l_fontspec_family_tl\endcsname\@empty
545         \M@NoFontspecFamilyError
546       \else
547         \expandafter
```

```
548        \let\expandafter\M@f@ntn@me\csname l_fontspec_family_tl\encsname
549        \def\@tempbase{\M@f@ntn@me\space(from fontspec)}
550        \let\M@f@ntn@meb@se\M@f@ntn@me % no separate font in base mode
551      \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 LuaTeX, 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.

```
552    \else
553      \@ifpackageloaded{fontspec}
554      {\csname fontspec_set_family:Nnn\endcsname\M@f@ntn@me{}{\@tempa}
555        % no separate font in base mode
556        \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 LuaTeX, `\@basefeatures` will be the same as `\@tempfeatures` except with
`mode=base` at the end, and if we are using XeTeX, it will be exactly the same.

```
557      {\M@Decl@reF@milytrue
558        \M@Decl@reF@milyB@setrue
559        \ifx\@tempfeatures\@empty
560          \ifdefined\directlua
561            \edef\@basefeatures{mode=base}
562          \else
563            \let\@basefeatures\@tempfeatures
564          \fi
565        \else
566          \edef\@tempfeatures{\expandafter\M@strip@colon\@tempfeatures}
567          \ifdefined\directlua
568            \edef\@basefeatures{\@tempfeatures;mode=base}
569          \else
570            \let\@basefeatures\@tempfeatures
571          \fi
572        \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.

```
573        \edef@nospace\@tempa{\@tempa}
574        \edef@nospace\@tempb{\@tempbase}
575        \M@Decl@reF@milytrue
576        \@tfor\@i:=\@tempa\@tempb\@tempbase\do{%
577          \ifcsname TU+\@i\endcsname
578            \expandafter\let\expandafter\M@f@ntn@me\@i
```

```
579            \M@Decl@reF@milyfalse
580            \M@fill@nfss@shapes\M@f@ntn@me\@tempfeatures
581            \@break@tfor
582        \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`.

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

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

```
587          \M@fill@nfss@shapes\M@f@ntn@me\@tempfeatures
588        \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`.

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

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

```
601 \@onlypreamble\M@fill@nfss@shapes
602 \@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 LaTeX'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, math-font 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.

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

Do the same thing for the suboption.

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

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

```
630   \ifx\@temp@opt\@empty
631     \M@MissingOptionError
632   \else
```

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

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

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

```
640      \ifx\@temp@sub\@empty
641      \else
642        \edef\@temp@sub{\expandafter\M@strip@equals\@temp@sub}
643        \M@check@suboption@valid\@temp@sub % check that suboption is valid
644      \fi
645    \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.

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

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

```
654 \long\def\edef@nospace#1#2{%
655   \edef#1{#2}%
656   \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`.

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

The internal font-changing command.

```
658 \def\@mathfont[#1]#2{%
659   \ifx\set@mathchar\@@set@mathchar
660     \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.

```
661   \else
662     \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.

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

Expand, zap spaces from, and store the optional argument in `\@tempa`, and then perform the loop. We store the current keyword-suboption pair in `\@i` and then feed it to `\M@parse@option`. We need two `\edef`s 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.

```
673     \edef@nospace\@tempb{#1}
674     \@for\@i:=\@tempb\do{\expandafter\M@parse@option\@i=\@nil
675       \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!)

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

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

```
680          \edef\@tempc{\the\M@toks^^J\@temp@opt}
681          \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`.

```
682          \if@suboptionpresent
683            \expandafter\edef\csname M@\@temp@opt shape\endcsname{\@temp@sub}
684          \fi
685          \edef\@tempc{\the\M@toks\space
686            (\csname M@\@temp@opt shape\endcsname)}
687          \M@toks\expandafter{\@tempc}
688          \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.

```
689          \def\@tempb{upright}
690          \ifx\@tempb\@tempc
691            \let\@tempb\shapedefault
692          \else
693            \let\@tempb\itdefault
694          \fi
```

At this point we have the information we need to declare the symbol font: the NFSS family (`\M@f@ntn@me`), 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.

```
695          \ifcsname symM\@tempc\@tempa\endcsname\else
696            \M@SymbolFontInfo{\@tempbase}{\M@f@ntn@meb@se}
697              {\mddefault/\@tempb}{M\@tempc\@tempa}
698            \DeclareSymbolFont
699              {M\@tempc\@tempa}{TU}{\M@f@ntn@meb@se}{\mddefault}{\@tempb}
700          \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.

```
701          \expandafter
702            \edef\csname M@\@temp@opt @fontinfo\endcsname{\@tempbase}
703          \M@FontChangeInfo{\@temp@opt}{\@tempbase}
```

And now the magic happens!

```
704          \csname M@\@temp@opt @set\endcsname % set default font
705          \csname M@\@temp@opt true\endcsname % set switch to true
706        \fi
707      \fi}
```

Display concluding messages for the user.

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

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

```
722 \protected\def\setfont#1{%
723   \mathfont{#1}
724   \mathconstantsfont{#1}
725   \setmathfontcommands{#1}
726   \let\rmdefault\M@f@ntn@me}
727 \@onlypreamble\setfont
```

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

```
728 \ifdefined\directlua
729   \let\M@SetMathConstants\relax
730   \protected\def\mathconstantsfont{\@ifnextchar[{\@mathconstantsfont}
731     {\@mathconstantsfont[upright]}}
732   \def\@mathconstantsfont[#1]#2{%
733     \edef\@tempa{\csname M@fontfamily@base@#2\endcsname}
734     \expandafter\ifx\@tempa\relax
735       \M@BadMathConstantsFontError{#2}
736     \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`.

```
737       \def\@tempb{#1}
738       \def\@tempc{upright}
739       \ifx\@tempb\@tempc
740         \let\@tempc\shapedefault
741       \else
742         \def\@tempc{italic}
743         \ifx\@tempb\@tempc
744           \let\@tempc\itdefault
```

```
745          \else
746            \M@BadMathConstantsFontTypeError{#1}
747          \fi
748        \fi
749        \ifcsname symM#1\csname M@fontid@\@tempa\endcsname\endcsname\else
750          \DeclareSymbolFont{M#1\csname M@fontid@\@tempa\endcsname}
751            {TU}{\@tempa}{\mddefault}{\@tempc}
752        \fi
753        \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 LaTeX 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 LaTeX 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.

```
754        \protected\def\M@SetMathConstants{%
755          \begingroup
756          \escapechar\m@ne
757          \expandafter\getanddefine@fonts
758            \csname symM#1\csname M@fontid@\m@th@const@nts@font\endcsname
759              \expandafter
760            \endcsname % expands to \symMupright<id>
761            \csname TU/\m@th@const@nts@font/\seriesdefault/\@tempc
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 using `\M@check@arglength`; 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@good@arg` 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@meb@se}{#3}{#4}
790     \DeclareMathAlphabet{#1}{TU}{\M@f@ntn@meb@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 control sequence that the user will specify; and `##2`—the user's font information. 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       if num then % if number?
848         if num == num - (num \@percentchar 1) then % if integer?
849           if num >= 0 then % if nonnegative?
850             tex.print("\@backslashchar\@backslashchar endgroup%
851               \@backslashchar\@backslashchar M@arg@goodtrue")
852           end
853         end
854       end}
855     \fi
856   \fi}
```

Define `\RuleThicknessFactor`.

```
857   \protected\def\RuleThicknessFactor#1{%
858     \M@check@int{#1}
859     \ifM@arg@good
860       \global\M@rule@thickness@factor=#1\relax
861     \else
862       \M@BadIntegerError\RuleThicknessFactor{#1}
863     \fi}
```

Define `\IntegralItalicFactor`.

```
864   \protected\def\IntegralItalicFactor#1{%
865     \M@check@int{#1}
866     \ifM@arg@good
```

```
867      \global\M@integral@italic@factor=#1\relax
868    \else
869      \M@BadIntegerError\IntegralItalicFactor{#1}
870    \fi}
```

Define \SurdHorizontalFactor.

```
871   \protected\def\SurdHorizontalFactor#1{%
872    \M@check@int{#1}
873    \ifM@arg@good
874      \global\M@surd@horizontal@factor=#1\relax
875    \else
876      \M@BadIntegerError\SurdHorizontalFactor{#1}
877    \fi}
```

Define \SurdVerticalFactor.

```
878   \protected\def\SurdVerticalFactor#1{%
879    \M@check@int{#1}
880    \ifM@arg@good
881      \global\M@surd@vertical@factor=#1\relax
882    \else
883      \M@BadIntegerError\SurdVerticalFactor{#1}
884    \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.

```
885 \else
886   \@tfor\@i:=\RuleThicknessFactor\IntegralItalicFactor\SurdHorizontalFactor
887    \SurdVerticalFactor\CharmLine\CharmFile
888      \do{%
889        \protected\expandafter\edef\@i{\noexpand\M@NoFontAdjustError
890          \expandafter\noexpand\@i
891          \noexpand\@gobble}}
892 \fi
```

These commands should appear in the preamble only.

```
893 \@onlypreamble\RuleThicknessFactor
894 \@onlypreamble\IntegralItalicFactor
895 \@onlypreamble\SurdHorizontalFactor
896 \@onlypreamble\SurdVerticalFactor
897 \@onlypreamble\CharmLine
898 \@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.

```
899 \def\restoremathinternals{%
900   \ifx\set@mathchar\@@set@mathchar
901   \else
902     \wlog{Package mathfont Info: Restoring \string\set@mathchar.}
903     \wlog{Package mathfont Info: Restoring \string\set@mathsymbol.}
```

```
904    \wlog{Package mathfont Info: Restoring \string\set@mathaccent.}
905    \wlog{Package mathfont Info: Restoring \string\DeclareSymbolFont.}
906    \let\set@mathchar\@@set@mathchar
907    \let\set@mathsymbol\@@set@mathsymbol
908    \let\set@mathaccent\@@set@mathaccent
909    \let\DeclareSymbolFont\@@DeclareSymbolFont
910  \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 LuaTeX—perhaps in a future version.

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

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

```
928 \ifnum\tracinglostchars<\tw@
929   \tracinglostchars\tw@
930 \fi
```

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

```
931 \ifdefined\XeTeXrevision
932   \ifM@font@loaded
933     \AtEndOfPackage{%
934     \PackageWarningNoLine{mathfont}
935     {XeTeX detected. It looks like you\MessageBreak
936     specified a font when you loaded\MessageBreak
937     mathfont. If you run into problems\MessageBreak
938     with a font whose name is multiple\MessageBreak
939     words, try compiling with LuaLaTeX\MessageBreak
940     or call \string\setfont\space or \string\mathfont\MessageBreak
```

```
941    manually}}
942  \fi
943 \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 acutally 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.

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

Now print the messages.

```
953 \AtBeginDocument{%
954   \def\@tempa{%    <-- everything should be 14 characters long
955     upper:\@spaces\@spaces,%
956     lower:\@spaces\@spaces,%
957     diacritics:\space\space\space,%
958     greekupper:\space\space\space,%
959     greeklower:\space\space\space,%
960     agreekupper:\space\space,%
961     agreeklower:\space\space,%
962     cyrillicupper:,%
963     cyrilliclower:,%
964     hebrew:\@spaces\space\space\space,%
965     digits:\@spaces\space\space\space,%
966     operator:\@spaces\space,%
967     delimiters:\space\space\space,%
968     radical:\@spaces\space\space,%
969     bigops:\@spaces\space\space\space,%
970     extbigops:\@spaces,%
971     symbols:\@spaces\space\space,%
972     extsymbols:\space\space\space,%
973     arrows:\@spaces\space\space\space,%
974     bb:\@spaces\@spaces\space\space\space,%
975     cal:\@spaces\@spaces\space\space,%
976     frak:\@spaces\@spaces\space,%
977     bcal:\@spaces\@spaces\space,%
978     bfrak:\@spaces\@spaces}
979   \wlog{^^JPackage mathfont Info: List of changes made in the preamble.}
980   \@for\@i:=\@tempa\do{%
981     \expandafter\keyword@info@begindocument\@i\@nil}
```

982   `\wlog{}}`

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

983 `\ifM@font@loaded`

984   `\AtEndOfPackage{\setfont\M@font@load}`

985 `\fi`

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

986 `\@onlypreamble\M@upper@set`

987 `\@onlypreamble\M@lower@set`

988 `\@onlypreamble\M@diacritics@set`

989 `\@onlypreamble\M@greekupper@set`

990 `\@onlypreamble\M@greeklower@set`

991 `\@onlypreamble\M@agreekupper@set`

992 `\@onlypreamble\M@agreeklower@set`

993 `\@onlypreamble\M@cyrillicupper@set`

994 `\@onlypreamble\M@cyrilliclower@set`

995 `\@onlypreamble\M@hebrew@set`

996 `\@onlypreamble\M@digits@set`

997 `\@onlypreamble\M@operator@set`

998 `\@onlypreamble\M@symbols@set`

999 `\@onlypreamble\M@extsymbols@set`

1000 `\@onlypreamble\M@delimiters@set`

1001 `\@onlypreamble\M@arrows@set`

1002 `\@onlypreamble\M@bigops@set`

1003 `\@onlypreamble\M@extbigops@set`

1004 `\@onlypreamble\M@bb@set`

1005 `\@onlypreamble\M@cal@set`

1006 `\@onlypreamble\M@frak@set`

1007 `\@onlypreamble\M@bcal@set`

1008 `\@onlypreamble\M@bfrak@set`

9   Adjust Fonts: Setup

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

1. Initialize a Lua table that contains new metrics for certain characters specific to math mode, such as letters with wider bounding boxes and large operator symbols.

2. Provide an interface for the user to change this metric information.

3. Write functions that accept a fontdata object and (a) change top-level math specs to indicate that we have a math function; (b) alter characters according to our Lua table of new metric information; and (c) populate a MathConstants table for the font.

4. Create callbacks that call these functions. Insert them into `luaotfload.patch_font`.

Table 2: Fields of Character Subtables in `mathfont`

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

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

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

Here's how to think about the dynamics of our approach. We use character metric information at three different times: pre-processing, interim processing, and post-processing. In pre-processing, which we implement in this section, we assemble initial character metric information into entries in `mathfont`. In other words, pre-processing means creating the initial `mathfont` subtables and happens during package loading. Interim processing means the user altering entries in `mathfont` and happens through `\CharmLine` and `\CharmFile`. This can occur at any point in the pramble. In post-processing, which we implement in the next section,

Table 3: Functions in `mathfont`

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

`mathfont` extracts information from the current state of the `mathfont` table and uses it to alter a fontdata object. Post-processing happens through the `luaotfload.patch_font` callback and occurs once at the point when TeX loads the font file. As a rule, LaTeX does not like to load fonts before it uses them, so post-processing typically happens `\AtBeginDocument` in the case of the main text font or whenever the user calls a `\text⟨`*font keyword*`⟩` command or enters math mode, whichever happens first. This is also why you cannot adjust fonts that TeX loaded before `mathfont`.

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

```
1009 \ifM@adjust@font
1010 \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.

```
1011 \bgroup
1012   \catcode`\~=0
1013   ~catcode`~\=12
1014   ~@firstofone{
```

```
1015 ~egroup
1016 ~def~M@number@ssert{"\n%
1017   Package mathfont error: Nonnumeric charm value.\n\n%
1018   I'm having trouble with a character metric.\n%
1019   Your \\CharmLine or \\CharmFile contains \""..temp_string.."\"\n%
1020   which is not a number. Make sure that your\n%
1021   charm information is all integers, floats,\n%
1022   or asterisks separated by commas or spaces.\n"}
1023 ~def~M@index@ssert{"\n%
1024   Package mathfont error: Invalid unicode index.\n\n%
1025   The unicode index \""..split_string[1].."\" is invalid. Make sure\n%
1026   that the first number in your \\CharmLine and in each\n%
1027   line of your \\CharmFile is an integer between 0 and\n%
1028   1,114,111.\n"}
1029 ~def~M@entries@ssert{"\n%
1030   Package mathfont error: Charm values too short.\n\n%
1031   Your charm information for U+"..index.." needs more\n%
1032   entries. Right now you have "..number_of_entries.." entries, and\n%
1033   you need at least "..entries_needed..". If you aren't sure what\n%
1034   to do, try adding asterisks to your \\CharmLine\n%
1035   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.

```
1036 \protected\def\CharmLine#1{%
1037   \begingroup
1038   \edef\@tempa{#1}
1039   \edef\@tempa{\detokenize\expandafter{\@tempa}}
1040   \@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.

```
1041   \ifin@ % is " in #1?
1042     \ifM@fromCharmFile
1043       \M@ForbiddenCharmFile{"}
1044     \else
1045       \M@ForbiddenCharmLine{"}
1046     \fi
1047   \else
1048     \@expandtwoargs\in@{\@backslashchar}{\@tempa}
1049     \ifin@ % is backslash in #1?
1050       \ifM@fromCharmFile
1051         \M@ForbiddenCharmFile{\@backslashchar}
1052       \else
1053         \M@ForbiddenCharmLine{\@backslashchar}
```

```
1054        \fi
1055      \else
```

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

```
1056        \directlua{mathfont:add_to_charm("\@tempa")}
1057      \fi
1058    \fi
1059    \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.

```
1060 \protected\def\CharmFile#1{%
1061    \begingroup
1062    \M@fromCharmFiletrue
1063    \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.

```
1064    \def\@next{%
1065      \read\M@Charm to \@tempa
1066      \CharmLine\@tempa
1067      \ifeof\M@Charm\else % if file has more lines?
1068        \expandafter\@next
1069      \fi}
```

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

```
1070    \@next
1071    \immediate\closein\M@Charm
1072    \M@fromCharmFilefalse
1073    \endgroup}
```

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

```
1074 \directlua{
1075 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.

```
1076 function mathfont:new_type_a(index, next, data)
1077    self[index] = {}
```

```
1078    self[index].type = "a"
1079    self[index].next = next
1080    self[index].left_stretch = data[1] / 1000
1081    self[index].right_stretch = data[2] / 1000
1082    self[index].top_accent_stretch = data[3] / 1000
1083    self[index].bot_accent_stretch = data[4] / 1000
1084 end
```

Initializing type `e` characters is more complicated. The `index` argument is the base-10 unicode value of the caracter 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.

```
1085 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`.

```
1086    local v = \string# next
1087    self[index] = {}
1088    self[index].type = "e"
1089    self[index].smash = smash
1090    self[index].next = next
1091    self[index].total_variants = v
1092    self[index].data = {}
```

We expect `data` to have $2v + 2$ entries, which we consider in pairs. The ith pair (i.e. entries i and $i + 1$ of `data`) encodes the horizontal and vertical scale factors for the ith 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.

```
1093    for i = 1, v, 1 do
1094       self[index].data[i] = {}
1095       self[index].data[i].x = data[2*i-1] / 1000
1096       self[index].data[i].y = data[2*i] / 1000
1097    end
1098    self[index].top_accent_stretch = data[2*v+1] / 1000
1099    self[index].bot_accent_stretch = data[2*v+2] / 1000
1100 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.

```
1101 function mathfont:new_type_u(index, data)
1102    self[index] = {}
1103    self[index].type = "u"
1104    self[index].top_accent_stretch = data[1] / 1000
1105    self[index].bot_accent_stretch = data[2] / 1000
1106 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`.

```
1107 function mathfont.parse_charm(charm_input)
1108   local split_string = {}
1109   local charm_string = charm_input
1110   local temp_string = ""
1111   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` preceeding the first comma or space and append it to `split_string` as a separate entry.

```
1112   while string.find(charm_string, " ") or string.find(charm_string, ",") do
1113     local length = string.len(charm_string)
1114     local first_space = string.find(charm_string, " ") or length
1115     local first_comma = string.find(charm_string, ",") or length
```

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

```
1116     local sep = first_space
1117     if first_comma < first_space then
1118       sep = first_comma
1119     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`.

```
1120     temp_string = string.sub(charm_string, 1, sep-1)
1121     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.

```
1122     if temp_string \noexpand~= "" then
1123       if tonumber(temp_string) then % if a number, append number
1124         split_string[i] = tonumber(temp_string)
1125         i = i+1
1126       elseif temp_string == "*" then % if asterisk, append asterisk
1127         split_string[i] = temp_string
1128         i = i+1
1129       else % if neither, raise error
1130         error(\M@number@ssert)
1131       end
1132     end
1133   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.

```
1134   temp_string = charm_string
1135   if temp_string \noexpand~= "" then
1136     if tonumber(temp_string) then % if a number, append number
1137       split_string[i] = tonumber(temp_string)
1138     elseif temp_string == "*" then % if asterisk, append asterisk
1139       split_string[i] = temp_string
1140     else % if neither, raise error
1141       error(\M@number@ssert)
1142     end
1143   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.

```
1144   local index = split_string[1]
1145   if index == "*" then
1146     error(\M@index@ssert)
1147   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 begn 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`.

```
1148   local rounded = index - (index \@percentchar 1) % subtract decimal portion
1149   local max = 1114111
1150   assert(index == rounded and index >= 0 and index <= max, \M@index@ssert)
1151   return split_string
1152 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`.

```
1153 function mathfont:add_to_charm(charm_string)
1154   local charm_metrics = self.parse_charm(charm_string)
1155   local index = charm_metrics[1]
1156   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`.

```
1157   if not self[index] then
1158     self:new_type_u(index, {0, 0})
1159   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 wll overwrite the left and right offset, and the last two overwrite accent placement.

```
1160   if self[index].type == "a" then
1161     local entries_needed = 5
1162     assert(number_of_entries >= entries_needed, \M@entries@ssert)
1163     if charm_metrics[2] \noexpand~= "*" then
1164       self[index].left_stretch = charm_metrics[2] / 1000
1165     end
1166     if charm_metrics[3] \noexpand~= "*" then
1167       self[index].right_stretch = charm_metrics[3] / 1000
1168     end
1169     if charm_metrics[4] \noexpand~= "*" then
1170       self[index].top_accent_stretch = charm_metrics[4] / 1000
1171     end
1172     if charm_metrics[5] \noexpand~= "*" then
1173       self[index].bot_accent_stretch = charm_metrics[5] / 1000
1174     end
```

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

```
1175   elseif self[index].type == "e" then
1176     local tot_variants = self[index].total_variants
1177     local entries_needed = 2 * tot_variants + 3
1178     assert(number_of_entries >= entries_needed, \M@entries@ssert)
1179     for i = 1, tot_variants, 1 do
1180       if charm_metrics[2*i] \noexpand~= "*" then
1181         self[index].data[i].x = charm_metrics[2*i] / 1000
1182       end
1183       if charm_metrics[2*i+1] \noexpand~= "*" then
1184         self[index].data[i].y = charm_metrics[2*i+1] / 1000
1185       end
1186     end
```

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

```
1187     if charm_metrics[2*tot_variants+2] \noexpand~= "*" then
1188       self[index].top_accent_stretch = charm_metrics[2*tot_variants+2] / 1000
1189     end
1190     if charm_metrics[2*tot_variants+3] \noexpand~= "*" then
1191       self[index].bot_accent_stretch = charm_metrics[2*tot_variants+3] / 1000
1192     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.

```
1193   elseif self[index].type == "u" then
1194     local entries_needed = 3
1195     assert(number_of_entries >= entries_needed, \M@entries@ssert)
1196     if charm_metrics[2] \noexpand~= "*" then
```

```
1197        self[index].top_accent_stretch = charm_metrics[2] / 1000
1198      end
1199      if charm_metrics[3] \noexpand~= "*" then
1200        self[index].bot_accent_stretch = charm_metrics[3] / 1000
1201      end
1202   end
1203 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.

```
1204 function mathfont.make_hex_value(integer)
1205   if integer == 0 then
1206     return "0000"
1207   end
1208   if integer == 1 then
1209     return "0001"
1210   end
1211   local hex_digits = "0123456789ABCDEF" % for reference
1212   local hex_string = ""
1213   local curr_val = integer
1214   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`.

```
1215   local i = 0
1216   while 16^i <= curr_val do
1217     i = i+1
1218   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`.

```
1219   for j = i-1, 0, -1 do
1220     remainder = curr_val \@percentchar (16^j)
1221     curr_val = (curr_val - remainder) / (16^j)
1222     hex_string = hex_string .. string.sub(hex_digits, curr_val+1, curr_val+1)
1223     curr_val = remainder
1224   end
```

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

```
1225   if \string# hex_string < 4 then
1226     for i = \string# hex_string, 4, 1 do
1227       hex_string = "0" .. hex_string
1228     end
1229   end
```

Table 4: Callbacks Created by mathfont

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

```
1230    return hex_string
1231 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.

```
1232 function mathfont.glyph_info(char)
1233    local glyph_width = char.width or 0
1234    local glyph_height = char.height or 0
1235    local glyph_depth = char.depth or 0
1236    local glyph_italic = char.italic or 0
1237    return glyph_width, glyph_height, glyph_depth, glyph_italic
1238 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.

```
1239 function mathfont:smash_glyph(index, fontdata)
1240    local smash_table = {}
1241    smash_table.width = 0
1242    smash_table.height = 0
1243    smash_table.depth = 0
1244    smash_table.commands = {{"char", index}}
1245    return smash_table
1246 end
```

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

```
1247 function mathfont.empty(arg)
1248 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.

```
1249 function mathfont.make_a_commands(index, offset)
1250   local c_1 = {"right", offset}
1251   local c_2 = {"char", index}
1252   return {c_1, c_2}
1253 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.

```
1254 function mathfont:make_a_table(index, charm_data, fontdata)
1255   local a_table = {}
1256   local char = fontdata.characters[index] or {}
1257   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.

```
1258   local left_stretch = charm_data.left_stretch
1259   local right_stretch = charm_data.right_stretch
1260   local width, height, depth, italic = self.glyph_info(char)
```

Incorporate the italic correction into the character width.

```
1261   width = width + italic
```

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

```
1262    local offset = width * left_stretch
1263    a_table.width = width * (1 + left_stretch + right_stretch)
1264    a_table.height = height
1265    a_table.depth = depth
1266    a_table.italic = italic
1267    a_table.unicode = index
```

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

```
1268    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 parallogram where the rectangle overhangs both slanted edges of the parallogram 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 parallogram from step (3) previously, and we end up with

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

where σ_1 is the `slant` parameter and `width` and `height` refer to the character in question. This equation simplifies to

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

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.

```
1269    local top_base = (0.5 + left_stretch) * width + 0.5 * slant * height
1270    local bot_base = (0.5 + left_stretch) * width - 0.5 * slant * height
1271    local top_accent_shift = charm_data.top_accent_stretch * width
1272    local bot_accent_shift = charm_data.bot_accent_stretch * width
1273    a_table.top_accent = top_base + top_accent_shift
1274    a_table.bot_accent = bot_base + bot_accent_shift
```

Add the commands to the table.

```
1275    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.

```
1276   a_table.mathkern = {}
1277   a_table.mathkern.top_right = {{height = 0, kern = -italic}}
1278   a_table.mathkern.bottom_right = {{height = 0, kern = -italic}}
1279   a_table.mathkern.top_left = {{height = 0, kern = 0}}
1280   a_table.mathkern.bottom_left = {{height = 0, kern = 0}}
1281   return a_table
1282 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.

```
1283 function mathfont:modify_e_base(index, fontdata)
1284   local char = fontdata.characters[index] or {}
1285   local width, height, depth, italic = self.glyph_info(char)
1286   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`.

```
1287   if index == 8730 then
```

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

```
1288     local horizontal_scale = tex.getcount("M@surd@horizontal@factor") / 1000
1289     local vertical_scale = tex.getcount("M@surd@vertical@factor") / 1000
1290     char.width = horizontal_scale * char.width
1291     char.height = vertical_scale * height
1292   end
```

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

```
1293   if index == 8747 then
1294     local scale_factor = tex.getcount("M@integral@italic@factor") / 1000
1295     char.italic = scale_factor * width
1296   end
1297 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.

```
1298 function mathfont.make_e_commands(index, h_stretch, v_stretch)
1299   local c_1 = {"pdf", "origin", string.format(
1300     "q \@percentchar s 0 0 \@percentchar s 0 0 cm", h_stretch, v_stretch)}
1301   local c_2 = {"char", index}
1302   local c_3 = {"pdf", "origin", "Q"}
1303   return {c_1, c_2, c_3}
1304 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.

```
1305 function mathfont:make_e_table(index, charm_data, fontdata)
1306   local e_table = {}
1307   local char = fontdata.characters[index] or {}
1308   local slant = fontdata.parameters.slant / 65536
1309   local tounicode = self.make_hex_value(index)
1310   local smash_index = charm_data.smash
1311   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.

```
1312   for i = 1, charm_data.total_variants, 1 do
1313     local h_stretch = charm_data.data[i].x
1314     local v_stretch = charm_data.data[i].y
1315     local new_width = width * h_stretch
1316     local new_height = height * v_stretch
1317     local new_depth = depth * v_stretch
1318     local new_italic = italic * h_stretch
```

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

```
1319     e_table[i] = {}
1320     e_table[i].width = new_width
1321     e_table[i].height = new_height
1322     e_table[i].depth = new_depth
1323     e_table[i].italic = new_italic
```

Add the unicode information.

```
1324     e_table[i].unicode = index
1325     e_table[i].tounicode = tounicode
```

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

```
1326     local base_top_accent = 0.5 * new_width + 0.5 * slant * new_height
1327     local base_bot_accent = 0.5 * new_width - 0.5 * slant * new_height
1328     local top_accent_shift = charm_data.top_accent_stretch * new_width
1329     local bot_accent_shift = charm_data.bot_accent_stretch * new_width
1330     e_table[i].top_accent = base_top_accent + top_accent_shift
1331     e_table[i].bot_accent = base_bot_accent + bot_accent_shift
```

Add the commands.

```
1332     e_table[i].commands =
1333       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 the the $i + 1$st character, and we can extract the index from the `charm_information`.

```
1334     if i < charm_data.total_variants then
1335       e_table[i].next = charm_data.next[i+1]
1336     end
1337   end
1338   return e_table
1339 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.

```
1340 function mathfont:make_u_table(index, charm_data, fontdata)
1341   local u_table = fontdata.characters[index] or {}
1342   local slant = fontdata.parameters.slant / 65536 or 0
1343   local width, height, depth, italic = self.glyph_info(u_table)
1344   local new_width = width + italic
1345   u_table.width = new_width
```

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

```
1346   local base_top_accent = 0.5 * new_width + 0.5 * slant * height
1347   local base_bot_accent = 0.5 * new_width - 0.5 * slant * height
1348   local top_accent_shift = charm_data.top_accent_stretch * new_width
1349   local bot_accent_shift = charm_data.bot_accent_stretch * new_width
1350   u_table.top_accent = base_top_accent + top_accent_shift
1351   u_table.bot_accent = base_bot_accent + bot_accent_shift
```

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

```
1352   u_table.mathkern = {}
1353   u_table.mathkern.top_right = {{height = 0, kern = -italic}}
1354   u_table.mathkern.bottom_right = {{height = 0, kern = -italic}}
1355   u_table.mathkern.top_left = {{height = 0, kern = 0}}
1356   u_table.mathkern.bottom_left = {{height = 0, kern = 0}}
1357   return u_table
1358 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.

```
1359 function mathfont.make_fake_angle(index, smash, fontdata)
1360   local temp = {}
1361   local lparen = fontdata.characters[40] or {}
1362   local lparen_height = lparen.height or 0
1363   local lparen_depth = lparen.depth or 0
1364   local glyph = fontdata.characters[index] or {}
1365   local glyph_height = glyph.height or 0
1366   local base_height = 0.9 * glyph_height
1367   local factor = 0
1368   if glyph_height \noexpand~= 0 then
1369     factor = (lparen_height + lparen_depth) / base_height
1370   end
1371   local shift = 0.1 * glyph_height * factor + lparen_depth
1372   temp.height = lparen_height
1373   temp.depth = lparen_depth
1374   temp.width = glyph.width or 0
1375   temp.italic = glyph.italic or 0
1376   temp.top_accent = glyph.top_accent or 0.5 * temp.width
1377   temp.bot_accent = glyph.bot_accent or 0.5 * temp.width
1378   temp.commands = {
1379     {"down", shift},
1380     {"pdf", "origin", string.format("q 1 0 0 \@percentchar s 0 0 cm", factor)},
1381     {"char", smash},
1382     {"pdf", "origin", "Q"},
1383     {"down", -shift}}
1384   return temp
1385 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.

```
1386 function mathfont.set_nomath_true(fontdata)
1387   fontdata.nomath = false
1388   fontdata.oldmath = false
1389 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.

```
1390 function mathfont.apply_charm_info(fontdata)
```

```
1391   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.

```
1392   chars[1044538] = mathfont:smash_glyph(8249, fontdata) % \lguil
1393   chars[1044539] = mathfont:smash_glyph(8250, fontdata) % \rguil
1394   chars[1044540] = mathfont:smash_glyph(171, fontdata) % \llguil
1395   chars[1044541] = mathfont:smash_glyph(187, fontdata) % \rrguil
```

Now add the characters to the font.

```
1396   chars[1044508] = mathfont.make_fake_angle(8249, 1044538, fontdata)
1397   chars[1044509] = mathfont.make_fake_angle(8250, 1044539, fontdata)
1398   chars[1044510] = mathfont.make_fake_angle(171, 1044540, fontdata)
1399   chars[1044511] = mathfont.make_fake_angle(187, 1044541, fontdata)
```

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

```
1400   if not chars[8711] then
1401     chars[8710] = chars[8710] or {}
1402     chars[1044508] = mathfont:smash_glyph(8710, fontdata)
1403     chars[8711] = {}
1404     chars[8711].width = chars[8710].width or 0
1405     chars[8711].height = chars[8710].height or 0
1406     chars[8711].depth = chars[8710].depth or 0
1407     chars[8711].italic = chars[8710].italic or 0
1408     chars[8711].top_accent = chars[8710].top_accent or 0.5 * chars[8711].width
1409     chars[8711].bot_accent = chars[8710].bot_accent or 0.5 * chars[8711].width
1410     chars[8711].unicode = 8711
1411     chars[8711].tounicode = mathfont.make_hex_value(8711)
1412     chars[8711].commands = {
1413       {"down", -chars[8711].height},
1414       {"pdf", "origin", "q 1 0 0 -1 0 0 cm"},
1415       {"char", 1044508},
1416       {"pdf", "origin", "Q"},
1417       {"down", chars[8711].height}}
1418   end
```

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

```
1419   for index, info in pairs(mathfont) do
1420     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.

```
1421       if info.type == "a" then
1422         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 caracters in the font linking all the variants together.

```
1423       elseif info.type == "e" then
1424         local smash = info.smash
```

```
1425        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.

```
1426        chars[index].next = info.next[1]
1427        mathfont:modify_e_base(index, fontdata)
1428        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 time. Each subtable goes in unicode slots that we take from the charm information, specifically the `next` table from `info`.

```
1429        local variants_table = mathfont:make_e_table(index, info, fontdata)
1430        for i = 1, info.total_variants, 1 do
1431          chars[info.next[i]] = variants_table[i]
1432        end
```

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

```
1433      elseif info.type == "u" then
1434        chars[index] = mathfont:make_u_table(index, info, fontdata)
1435      end
1436    end
1437  end
1438 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.

```
1439 function mathfont.math_constants(fontdata)
1440   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.

```
1441   local size = fontdata.size or 0
1442   local ex = fontdata.parameters.x_height or 0
1443   local em = fontdata.parameters.quad or 0
1444   local A_height = 0
1445   local y_depth = 0
1446   if fontdata.characters[65] then
1447     A_height = fontdata.characters[65].height or 0 % A
1448   end
1449   if fontdata.characters[121] then
1450     y_depth = fontdata.characters[121].depth or 0 % y
1451   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.

```
1452   local axis = 0
1453   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.

```
1454   local dim = "FractionRuleThickness"
1455   if not fontdata.MathConstants[dim] then
1456     local scale_factor = tex.getcount("M@rule@thickness@factor") / 1000
1457     rule_thickness = (size / 18) * scale_factor
1458     fontdata.MathConstants[dim] = rule_thickness
1459   else
1460     rule_thickness = fontdata.MathConstants[dim]
1461   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`.

```
1462   local dim = "AxisHeight"
1463   if fontdata.MathConstants[dim] then
1464     axis = fontdata.MathConstants[dim]
1465   else
1466     if fontdata.characters[45] then
1467       axis = fontdata.characters[45].height - 0.5 * rule_thickness
1468     else
1469       axis = 0.8 * ex
1470     end
1471     fontdata.MathConstants[dim] = axis
1472   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 decender, TEX will raise its baseline by ξ_{10}, and if it has a descener, 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.

```
1473   local xi_9 = 2 * rule_thickness              % upper limit minimum clearance
1474   local xi_10 = xi_9 + 0.35 * y_depth          % upper limit attempt placement
1475   local xi_11 = xi_10                          % lower limit minimum clearance
1476   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.

```
1477   local sigma_8 = axis + 1.5 * rule_thickness + y_depth + 0.1 * A_height
1478   local sigma_9 = (axis + 1.35 * rule_thickness + 0.7 * y_depth +
1479      0.07 * A_height)
1480   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.

```
1481   local sigma_11 = (-axis + 1.5  * rule_thickness + 0.5 * y_depth +
1482      1.1 * A_height)
1483   local sigma_12 = (-axis + 1.35 * rule_thickness + 0.35 * y_depth +
1484      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.

```
1485   local sigma_13 = 0.6 * A_height     % attempted superscript height
1486   local sigma_15 = 0.5 * A_height     % attempted superscript for \cramped
1487   local sigma_16 = 1.1 * y_depth      % attempted subscript lower
1488   if sigma_16 < 0.2 * A_height then
1489     sigma_16 = 0.2 * A_height
```

```
1490   end
1491   local sigma_17 = sigma_16            % sigma_16 when superscript present
1492   local sigma_18 = 0.5 * A_height      % superscript lower for boxed subformula
1493   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.

```
1494   local dim = "DisplayOperatorMinHeight"
1495   if not fontdata.MathConstants[dim] then
1496     fontdata.MathConstants[dim] = 1.8 * A_height
1497   end
1498   local dim = "FractionDelimiterDisplayStyleSize"
1499   if not fontdata.MathConstants[dim] then
1500     fontdata.MathConstants[dim] = 2 * size
1501   end
1502   local dim = "FractionDelimiterSize"
1503   if not fontdata.MathConstants[dim] then
1504     fontdata.MathConstants[dim] = 1.3 * size
1505   end
1506   local dim = "FractionDenominatorDisplayStyleShiftDown"
1507   if not fontdata.MathConstants[dim] then
1508     fontdata.MathConstants[dim] = sigma_11
1509   end
1510   local dim = "FractionDenominatorShiftDown"
1511   if not fontdata.MathConstants[dim] then
1512     fontdata.MathConstants[dim] = sigma_12
1513   end
```

We set the minium 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.

```
1514   local dim = "FractionDenominatorDisplayStyleGapMin"
1515   if not fontdata.MathConstants[dim] then
1516     fontdata.MathConstants[dim] = rule_thickness + 0.5 * y_depth
1517   end
1518   local dim = "FractionDenominatorGapMin"
1519   if not fontdata.MathConstants[dim] then
1520     fontdata.MathConstants[dim] = rule_thickness + 0.35 * y_depth
1521   end
1522   local dim = "FractionNumeratorDisplayStyleShiftUp"
1523   if not fontdata.MathConstants[dim] then
1524     fontdata.MathConstants[dim] = sigma_8
1525   end
```

```
1526  local dim = "FractionNumeratorShiftUp"
1527  if not fontdata.MathConstants[dim] then
1528    fontdata.MathConstants[dim] = sigma_9
1529  end
1530  local dim = "FractionNumeratorDisplayStyleGapMin"
1531  if not fontdata.MathConstants[dim] then
1532    fontdata.MathConstants[dim] = rule_thickness
1533  end
1534  local dim = "FractionNumeratorGapMin"
1535  if not fontdata.MathConstants[dim] then
1536    fontdata.MathConstants[dim] = rule_thickness
1537  end
```

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

```
1538  local dim = "SkewedFractionHorizontalGap"
1539  if not fontdata.MathConstants[dim] then
1540    fontdata.MathConstants[dim] = 0.5 * em
1541  end
1542  local dim = "SkewedFractionVerticalGap"
1543  if not fontdata.MathConstants[dim] then
1544    fontdata.MathConstants[dim] = ex
1545  end
```

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

```
1546  local dim = "UpperLimitBaselineRiseMin"
1547  if not fontdata.MathConstants[dim] then
1548    fontdata.MathConstants[dim] = xi_11
1549  end
1550  local dim = "UpperLimitGapMin"
1551  if not fontdata.MathConstants[dim] then
1552    fontdata.MathConstants[dim] = xi_9
1553  end
1554  local dim = "LowerLimitBaselineDropMin"
1555  if not fontdata.MathConstants[dim] then
1556    fontdata.MathConstants[dim] = xi_12
1557  end
1558  local dim = "LowerLimitGapMin"
1559  if not fontdata.MathConstants[dim] then
1560    fontdata.MathConstants[dim] = xi_10
1561  end
```

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

```
1562  local dim = "StretchStackGapBelowMin"
1563  if not fontdata.MathConstants[dim] then
1564    fontdata.MathConstants[dim] = xi_10
1565  end
```

```
1566   local dim = "StretchStackTopShiftUp"
1567   if not fontdata.MathConstants[dim] then
1568     fontdata.MathConstants[dim] = xi_11
1569   end
1570   local dim = "StretchStackGapAboveMin"
1571   if not fontdata.MathConstants[dim] then
1572     fontdata.MathConstants[dim] = xi_9
1573   end
1574   local dim = "StretchStackBottomShiftDown"
1575   if not fontdata.MathConstants[dim] then
1576     fontdata.MathConstants[dim] = xi_12
1577   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.

```
1578   local dim = "OverbarExtraAscender"
1579   if not fontdata.MathConstants[dim] then
1580     fontdata.MathConstants[dim] = rule_thickness
1581   end
1582   local dim = "OverbarRuleThickness"
1583   if not fontdata.MathConstants[dim] then
1584     fontdata.MathConstants[dim] = rule_thickness
1585   end
1586   local dim = "OverbarVerticalGap"
1587   if not fontdata.MathConstants[dim] then
1588     fontdata.MathConstants[dim] = 2 * rule_thickness
1589   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.

```
1590   local dim = "RadicalExtraAscender"
1591   if not fontdata.MathConstants[dim] then
1592     fontdata.MathConstants[dim] = rule_thickness
1593   end
1594   local dim = "RadicalRuleThickness"
1595   if not fontdata.MathConstants[dim] then
1596     fontdata.MathConstants[dim] = rule_thickness
1597   end
1598   local dim = "RadicalDisplayStyleVerticalGap"
1599   if not fontdata.MathConstants[dim] then
1600     fontdata.MathConstants[dim] = 2 * rule_thickness
1601   end
1602   local dim = "RadicalVerticalGap"
1603   if not fontdata.MathConstants[dim] then
1604     fontdata.MathConstants[dim] = rule_thickness
1605   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.

```
1606   local dim = "RadicalKernBeforeDegree"
1607   if not fontdata.MathConstants[dim] then
1608     fontdata.MathConstants[dim] = (5/18) * em
1609   end
1610   local dim = "RadicalKernAfterDegree"
1611   if not fontdata.MathConstants[dim] then
1612     fontdata.MathConstants[dim] = (10/18) * em
1613   end
1614   local dim = "RadicalDegreeBottomRaisePercent"
1615   if not fontdata.MathConstants[dim] then
1616     fontdata.MathConstants[dim] = 60
1617   end
```

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

```
1618   local dim = "SpaceAfterScript"
1619   if not fontdata.MathConstants[dim] then
1620     fontdata.MathConstants[dim] = 0.1 * em
1621   end
```

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

```
1622   local dim = "StackBottomDisplayStyleShiftDown"
1623   if not fontdata.MathConstants[dim] then
1624     fontdata.MathConstants[dim] = sigma_11
1625   end
1626   local dim = "StackBottomShiftDown"
1627   if not fontdata.MathConstants[dim] then
1628     fontdata.MathConstants[dim] = sigma_12
1629   end
1630   local dim = "StackTopDisplayStyleShiftUp"
1631   if not fontdata.MathConstants[dim] then
1632     fontdata.MathConstants[dim] = sigma_8
1633   end
1634   local dim = "StackTopShiftUp"
1635   if not fontdata.MathConstants[dim] then
1636     fontdata.MathConstants[dim] = sigma_10
1637   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

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

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

```
1638   local dim = "StackDisplayStyleGapMin"
```

```
1639   if not fontdata.MathConstants[dim] then
1640     fontdata.MathConstants[dim] = 5 * rule_thickness + 0.5 * y_depth
1641   end
1642   local dim = "StackGapMin"
1643   if not fontdata.MathConstants[dim] then
1644     fontdata.MathConstants[dim] = 3 * rule_thickness + 0.35 * y_depth
1645   end
```

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

```
1646   local dim = "SubscriptShiftDown"
1647   if not fontdata.MathConstants[dim] then
1648     fontdata.MathConstants[dim] = sigma_16
1649   end
1650   local dim = "SubscriptBaselineDropMin"
1651   if not fontdata.MathConstants[dim] then
1652     fontdata.MathConstants[dim] = sigma_19
1653   end
1654   local dim = "SubscriptShiftDownWithSuperscript"
1655   if not fontdata.MathConstants[dim] then
1656     fontdata.MathConstants[dim] = sigma_17
1657   end
```

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

```
1658   local dim = "SubscriptTopMax"
1659   if not fontdata.MathConstants[dim] then
1660     fontdata.MathConstants[dim] = 0.5 * A_height
1661   end
```

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

```
1662   local dim = "SubSuperscriptGapMin"
1663   if not fontdata.MathConstants[dim] then
1664     fontdata.MathConstants[dim] = rule_thickness
1665   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.

```
1666   local dim = "SuperscriptBottomMin"
1667   if not fontdata.MathConstants[dim] then
1668     fontdata.MathConstants[dim] = sigma_15 - 0.7 * y_depth
1669   end
1670   local dim = "SuperscriptBaselineDropMax"
1671   if not fontdata.MathConstants[dim] then
1672     fontdata.MathConstants[dim] = sigma_18
1673   end
1674   local dim = "SuperscriptShiftUp"
1675   if not fontdata.MathConstants[dim] then
```

```
1676     fontdata.MathConstants[dim] = sigma_13
1677   end
1678   local dim = "SuperscriptShiftUpCramped"
1679   if not fontdata.MathConstants[dim] then
1680     fontdata.MathConstants[dim] = sigma_15
1681   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.

```
1682   local dim = "SuperscriptBottomMaxWithSubscript"
1683   if not fontdata.MathConstants[dim] then
1684     fontdata.MathConstants[dim] = -sigma_16 + 0.7 * A_height + rule_thickness
1685   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.

```
1686   local dim = "UnderbarExtraDescender"
1687   if not fontdata.MathConstants[dim] then
1688     fontdata.MathConstants[dim] = rule_thickness
1689   end
1690   local dim = "UnderbarRuleThickness"
1691   if not fontdata.MathConstants[dim] then
1692     fontdata.MathConstants[dim] = rule_thickness
1693   end
1694   local dim = "UnderbarVerticalGap"
1695   if not fontdata.MathConstants[dim] then
1696     fontdata.MathConstants[dim] = 2 * rule_thickness
1697   end
```

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

```
1698   local dim = "MinConnectorOverlap"
1699   if not fontdata.MathConstants[dim] then
1700     fontdata.MathConstants[dim] = 0
1701   end
1702 end
```

Time for callbacks! We create six of them.

```
1703 luatexbase.create_callback("mathfont.inspect_font", "simple", mathfont.empty)
1704 luatexbase.create_callback("mathfont.pre_adjust", "simple", mathfont.empty)
1705 luatexbase.create_callback("mathfont.disable_nomath", "simple",
1706   mathfont.set_nomath_true)
1707 luatexbase.create_callback("mathfont.add_math_constants", "simple",
```

```
1708   mathfont.math_constants)
1709 luatexbase.create_callback("mathfont.fix_character_metrics", "simple",
1710   mathfont.apply_charm_info)
1711 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.

```
1712 function mathfont.info(msg)
1713   texio.write_nl("log", "Package mathfont Info: " .. msg)
1714 end
1715 function mathfont.get_font_name(fontdata)
1716   return fontdata.fullname or fontdata.psname or fontdata.name or "<??>"
1717 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.

```
1718 function mathfont.adjust_font(fontdata)
1719   luatexbase.call_callback("mathfont.inspect_font", fontdata)
1720   if fontdata.nomath then
1721     mathfont.info("Adjusting font " .. mathfont.get_font_name(fontdata) .. ".")
1722     luatexbase.call_callback("mathfont.pre_adjust", fontdata)
1723     luatexbase.call_callback("mathfont.disable_nomath", fontdata)
1724     luatexbase.call_callback("mathfont.add_math_constants", fontdata)
1725     luatexbase.call_callback("mathfont.fix_character_metrics", fontdata)
1726     luatexbase.call_callback("mathfont.post_adjust", fontdata)
1727   else
1728     mathfont.info("No changes made to " ..
1729       mathfont.get_font_name(fontdata) .. ".")
1730   end
1731 end
```

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

```
1732 luatexbase.add_to_callback("luaotfload.patch_font", mathfont.adjust_font,
1733   "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. Lowercase 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.

```
1734 mathfont:new_type_a(97, 1044480, {50, 50, -50, 0}) % a
1735 mathfont:new_type_a(98, 1044481, {50, 50, -50, 0}) % b
1736 mathfont:new_type_a(99, 1044482, {50, 50, 0, 0}) % c
1737 mathfont:new_type_a(100, 1044483, {50, -50, -50, 0}) % d
1738 mathfont:new_type_a(101, 1044484, {50, 50, 0, 0}) % e
1739 mathfont:new_type_a(102, 1044485, {200, 0, 0, 0}) % f
```

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

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

```
1762 mathfont:new_type_a(65, 1044512, {50, 0, 150, 0}) % A
1763 mathfont:new_type_a(66, 1044513, {50, 0, 0, 0}) % B
1764 mathfont:new_type_a(67, 1044514, {0, 0, 0, 0}) % C
1765 mathfont:new_type_a(68, 1044515, {50, 0, -50, 0}) % D
1766 mathfont:new_type_a(69, 1044516, {50, 0, 0, 0}) % E
1767 mathfont:new_type_a(70, 1044517, {50, 0, 0, 0}) % F
1768 mathfont:new_type_a(71, 1044518, {0, 0, 0, 0}) % G
1769 mathfont:new_type_a(72, 1044519, {50, 0, -50, 0}) % H
1770 mathfont:new_type_a(73, 1044520, {100, 0, 0, 0}) % I
1771 mathfont:new_type_a(74, 1044521, {50, 0, 100, 0}) % J
1772 mathfont:new_type_a(75, 1044522, {50, 0, 0, 0}) % K
1773 mathfont:new_type_a(76, 1044523, {50, 0, -180, 0}) % L
1774 mathfont:new_type_a(77, 1044524, {50, 0, -50, 0}) % M
1775 mathfont:new_type_a(78, 1044525, {50, 0, -50, 0}) % N
1776 mathfont:new_type_a(79, 1044526, {0, 0, 0, 0}) % O
1777 mathfont:new_type_a(80, 1044527, {0, 0, -50, 0}) % P
1778 mathfont:new_type_a(81, 1044528, {0, 50, 0, 0}) % Q
1779 mathfont:new_type_a(82, 1044529, {50, 0, -50, 0}) % R
1780 mathfont:new_type_a(83, 1044530, {0, 0, -50, 0}) % S
1781 mathfont:new_type_a(84, 1044531, {0, 0, -50, 0}) % T
1782 mathfont:new_type_a(85, 1044532, {0, 0, -50, 0}) % U
1783 mathfont:new_type_a(86, 1044533, {0, 50, 0, 0}) % V
1784 mathfont:new_type_a(87, 1044534, {0, 50, -50, 0}) % W
1785 mathfont:new_type_a(88, 1044535, {50, 0, 0, 0}) % X
```

```
1786 mathfont:new_type_a(89, 1044536, {0, 0, -50, 0}) % Y
1787 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 becoeme type a with adjusted bounding boxes, but I don't have immediate plans for such a change.

```
1788 mathfont:new_type_u(945, {0, 0}) % \alpha
1789 mathfont:new_type_u(946, {0, 0}) % \beta
1790 mathfont:new_type_u(947, {-50, 0}) % \gamma
1791 mathfont:new_type_u(948, {0, 0}) % \delta
1792 mathfont:new_type_u(1013, {50, 0}) % \epsilon
1793 mathfont:new_type_u(950, {0, 0}) % \zeta
1794 mathfont:new_type_u(951, {-50, 0}) % \eta
1795 mathfont:new_type_u(952, {0, 0}) % \theta
1796 mathfont:new_type_u(953, {-50, 0}) % \iota
1797 mathfont:new_type_u(954, {0, 0}) % \kappa
1798 mathfont:new_type_u(955, {-150, 0}) % lambda
1799 mathfont:new_type_u(956, {0, 0}) % \mu
1800 mathfont:new_type_u(957, {-50, 0}) % \nu
1801 mathfont:new_type_u(958, {0, 0}) % \xi
1802 mathfont:new_type_u(959, {0, 0}) % \omicron
1803 mathfont:new_type_u(960, {-100, 0}) % \pi
1804 mathfont:new_type_u(961, {-50, 0}) % \rho
1805 mathfont:new_type_u(963, {-100, 0}) % \sigma
1806 mathfont:new_type_u(964, {-100, 0}) % \tau
1807 mathfont:new_type_u(965, {-50, 0}) % \upsilon
1808 mathfont:new_type_u(981, {0, 0}) % \phi
1809 mathfont:new_type_u(967, {-50, 0}) % \chi
1810 mathfont:new_type_u(968, {-50, 0}) % \psi
1811 mathfont:new_type_u(969, {0, 0}) % \omega
1812 mathfont:new_type_u(976, {0, 0}) % \varbeta
1813 mathfont:new_type_u(949, {-50, 0}) % \varepsilon
1814 mathfont:new_type_u(977, {50, 0}) % \vartheta
1815 mathfont:new_type_u(1009, {-50, 0}) % \varrho
1816 mathfont:new_type_u(962, {-50, 0}) % \varsigma
1817 mathfont:new_type_u(966, {0, 0}) % \varphi
```

Upper-case Greek characters. Same as previously.

```
1818 mathfont:new_type_u(913, {0, 0}) % \Alpha
1819 mathfont:new_type_u(914, {0, 0}) % \Beta
1820 mathfont:new_type_u(915, {0, 0}) % \Gamma
1821 mathfont:new_type_u(916, {0, 0}) % \Delta
1822 mathfont:new_type_u(917, {0, 0}) % \Epsilon
1823 mathfont:new_type_u(918, {0, 0}) % \Zeta
1824 mathfont:new_type_u(919, {0, 0}) % \Eta
1825 mathfont:new_type_u(920, {0, 0}) % \Theta
1826 mathfont:new_type_u(921, {0, 0}) % \Iota
1827 mathfont:new_type_u(922, {0, 0}) % \Kappa
1828 mathfont:new_type_u(923, {0, 0}) % \Lambda
```

```
1829 mathfont:new_type_u(924, {0, 0}) % \Mu
1830 mathfont:new_type_u(925, {0, 0}) % \Nu
1831 mathfont:new_type_u(926, {0, 0}) % \Xi
1832 mathfont:new_type_u(927, {0, 0}) % \Omicron
1833 mathfont:new_type_u(928, {0, 0}) % \Pi
1834 mathfont:new_type_u(929, {0, 0}) % \Rho
1835 mathfont:new_type_u(931, {0, 0}) % \Sigma
1836 mathfont:new_type_u(932, {0, 0}) % \Tau
1837 mathfont:new_type_u(933, {0, 0}) % \Upsilon
1838 mathfont:new_type_u(934, {0, 0}) % \Phi
1839 mathfont:new_type_u(935, {0, 0}) % \Chi
1840 mathfont:new_type_u(936, {0, 0}) % \Psi
1841 mathfont:new_type_u(937, {0, 0}) % \Omega
1842 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`.

```
1843 local delim_glyphs = {40, % (
1844    41, % )
1845    47, % /
1846    91, % [
1847    92, % backslash
1848    93, % ]
1849    123, % {
1850    125, % }
1851    8249, % \lguil
1852    8250, % \rguil
1853    171, % \llguil
1854    187, % \rrguil
1855    1044508, % \fakelangle
1856    1044509, % \fakerangle
1857    1044510, % \fakellangle
1858    1044511} % \fakerrangle
1859 local big_op_glyphs = {33, % !
1860    35, % #
1861    36, % $
1862    37, % %
1863    38, % &
1864    43, % +
1865    63, % ?
1866    64, % @
1867    167, % \S
1868    215, % \times
1869    247, % \div
1870    8719, % \prod
```

```
1871   8721, % \sum
1872   8720, % \coprod
1873   8897, % \bigvee
1874   8896, % \bigwedge
1875   8899, % \bigcup
1876   8898, % \bigcap
1877   10753, % \bigoplus
1878   10754, % \bigotimes
1879   10752, % \bigodot
1880   10757, % \bigsqcap
1881   10758} % \bigsqcup
1882 local vert_glyphs = {124, 8730} % | and \surd
1883 local int_glyphs = {8747, % \intop
1884   8748, % \iint
1885   8749, % \iiint
1886   8750, % \oint
1887   8751, % \oiint
1888   8752} % \oiiint
```

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

```
1889 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.

```
1890 local delim_scale = {}
1891 local big_op_scale = {}
1892 local vert_scale = {}
1893 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 horzontally. Vertical characters stretch vertically only, so their horizontal scale factors are all constant. Big operators stretch the same in vertical and horizoontal directions.

```
1894 for i = 1, 15, 1 do
1895   delim_scale[2*i-1] = 1000 + 100*i % horizontal - delimiters
1896   delim_scale[2*i] = 1000 + 500*i   % vertical - delimiters
1897   vert_scale[2*i-1] = 1000
1898   vert_scale[2*i] = 1000 + 500*i    % vertical - vertically scaled chars
1899   big_op_scale[2*i-1] = 1000 + 100*i   % horizontal - big operators
1900   big_op_scale[2*i] = 1000 + 100*i     % vertical - big operators
```

The integral sign is particular. Visually, we would like an integral symbol that is larger than the large operators, which means that the integral sign should have no variants between the font's value of `\Umathoperatorsize` 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 `\Umathoperatorsize` setting in `populate_math_constants`. Effectively this means that the user should take the size of the integral operator as fixed and should set

`\Umathoperatorsize` to make all other big operators the desired size.

```
1901   int_scale[2*i-1] = 1000 + 500*i      % horizontal - integral sign
1902   int_scale[2*i] = 1000 + 1500*i       % vertical - integral sign
1903 end
```

We do not modify accent placement.

```
1904 delim_scale[31] = 0
1905 delim_scale[32] = 0
1906 big_op_scale[31] = 0
1907 big_op_scale[32] = 0
1908 vert_scale[31] = 0
1909 vert_scale[32] = 0
1910 int_scale[31] = 0
1911 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.

```
1912 function mathfont:add_extensible_variants(first_smash, glyph_list, scale_list)
1913   local variants = (\string# scale_list - 2) / 2
1914   local curr_smash = first_smash
1915   for i = 1, \string# glyph_list, 1 do
1916     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`.

```
1917     local curr_slots = {}
1918     for j = 1, variants, 1 do
1919       curr_slots[j] = curr_smash + j
1920     end
```

Add the charm information and increment `smash`.

```
1921     self:new_type_e(curr_char, curr_smash, curr_slots, scale_list)
1922     smash = smash + variants + 1
1923     curr_smash = smash
1924   end
1925 end
```

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

```
1926 mathfont:add_extensible_variants(smash, delim_glyphs, delim_scale)
1927 mathfont:add_extensible_variants(smash, big_op_glyphs, big_op_scale)
1928 mathfont:add_extensible_variants(smash, vert_glyphs, vert_scale)
1929 mathfont:add_extensible_variants(smash, int_glyphs, int_scale)
```

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

```
1930 }
1931 \fi % matches previous \ifM@adjust@font
```

12 Unicode Hex Values

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

```
1932 \ifM@adjust@font
1933   \def\M@upper@set{%
1934     \edef\M@upper@font{M\M@uppershape\@tempa}
1935     \DeclareMathSymbol{A}{\mathalpha}{\M@upper@font}{1044512}
1936     \DeclareMathSymbol{B}{\mathalpha}{\M@upper@font}{1044513}
1937     \DeclareMathSymbol{C}{\mathalpha}{\M@upper@font}{1044514}
1938     \DeclareMathSymbol{D}{\mathalpha}{\M@upper@font}{1044515}
1939     \DeclareMathSymbol{E}{\mathalpha}{\M@upper@font}{1044516}
1940     \DeclareMathSymbol{F}{\mathalpha}{\M@upper@font}{1044517}
1941     \DeclareMathSymbol{G}{\mathalpha}{\M@upper@font}{1044518}
1942     \DeclareMathSymbol{H}{\mathalpha}{\M@upper@font}{1044519}
1943     \DeclareMathSymbol{I}{\mathalpha}{\M@upper@font}{1044520}
1944     \DeclareMathSymbol{J}{\mathalpha}{\M@upper@font}{1044521}
1945     \DeclareMathSymbol{K}{\mathalpha}{\M@upper@font}{1044522}
1946     \DeclareMathSymbol{L}{\mathalpha}{\M@upper@font}{1044523}
1947     \DeclareMathSymbol{M}{\mathalpha}{\M@upper@font}{1044524}
1948     \DeclareMathSymbol{N}{\mathalpha}{\M@upper@font}{1044525}
1949     \DeclareMathSymbol{O}{\mathalpha}{\M@upper@font}{1044526}
1950     \DeclareMathSymbol{P}{\mathalpha}{\M@upper@font}{1044527}
1951     \DeclareMathSymbol{Q}{\mathalpha}{\M@upper@font}{1044528}
1952     \DeclareMathSymbol{R}{\mathalpha}{\M@upper@font}{1044529}
1953     \DeclareMathSymbol{S}{\mathalpha}{\M@upper@font}{1044530}
1954     \DeclareMathSymbol{T}{\mathalpha}{\M@upper@font}{1044531}
1955     \DeclareMathSymbol{U}{\mathalpha}{\M@upper@font}{1044532}
1956     \DeclareMathSymbol{V}{\mathalpha}{\M@upper@font}{1044533}
1957     \DeclareMathSymbol{W}{\mathalpha}{\M@upper@font}{1044534}
1958     \DeclareMathSymbol{X}{\mathalpha}{\M@upper@font}{1044535}
1959     \DeclareMathSymbol{Y}{\mathalpha}{\M@upper@font}{1044536}
1960     \DeclareMathSymbol{Z}{\mathalpha}{\M@upper@font}{1044537}}
1961 \else
1962   \def\M@upper@set{%
1963     \edef\M@upper@font{M\M@uppershape\@tempa}
1964     \DeclareMathSymbol{A}{\mathalpha}{\M@upper@font}{`A}
1965     \DeclareMathSymbol{B}{\mathalpha}{\M@upper@font}{`B}
1966     \DeclareMathSymbol{C}{\mathalpha}{\M@upper@font}{`C}
1967     \DeclareMathSymbol{D}{\mathalpha}{\M@upper@font}{`D}
1968     \DeclareMathSymbol{E}{\mathalpha}{\M@upper@font}{`E}
1969     \DeclareMathSymbol{F}{\mathalpha}{\M@upper@font}{`F}
1970     \DeclareMathSymbol{G}{\mathalpha}{\M@upper@font}{`G}
1971     \DeclareMathSymbol{H}{\mathalpha}{\M@upper@font}{`H}
1972     \DeclareMathSymbol{I}{\mathalpha}{\M@upper@font}{`I}
```

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

Set lower-case Latin characters.

```
1991 \ifM@adjust@font
1992    \def\M@lower@set{%
1993       \edef\M@lower@font{M\M@lowershape\@tempa}
1994       \DeclareMathSymbol{a}{\mathalpha}{\M@lower@font}{1044480}
1995       \DeclareMathSymbol{b}{\mathalpha}{\M@lower@font}{1044481}
1996       \DeclareMathSymbol{c}{\mathalpha}{\M@lower@font}{1044482}
1997       \DeclareMathSymbol{d}{\mathalpha}{\M@lower@font}{1044483}
1998       \DeclareMathSymbol{e}{\mathalpha}{\M@lower@font}{1044484}
1999       \DeclareMathSymbol{f}{\mathalpha}{\M@lower@font}{1044485}
2000       \DeclareMathSymbol{g}{\mathalpha}{\M@lower@font}{1044486}
2001       \DeclareMathSymbol{h}{\mathalpha}{\M@lower@font}{1044487}
2002       \DeclareMathSymbol{i}{\mathalpha}{\M@lower@font}{1044488}
2003       \DeclareMathSymbol{j}{\mathalpha}{\M@lower@font}{1044489}
2004       \DeclareMathSymbol{k}{\mathalpha}{\M@lower@font}{1044490}
2005       \DeclareMathSymbol{l}{\mathalpha}{\M@lower@font}{1044491}
2006       \DeclareMathSymbol{m}{\mathalpha}{\M@lower@font}{1044492}
2007       \DeclareMathSymbol{n}{\mathalpha}{\M@lower@font}{1044493}
2008       \DeclareMathSymbol{o}{\mathalpha}{\M@lower@font}{1044494}
2009       \DeclareMathSymbol{p}{\mathalpha}{\M@lower@font}{1044495}
2010       \DeclareMathSymbol{q}{\mathalpha}{\M@lower@font}{1044496}
2011       \DeclareMathSymbol{r}{\mathalpha}{\M@lower@font}{1044497}
2012       \DeclareMathSymbol{s}{\mathalpha}{\M@lower@font}{1044498}
2013       \DeclareMathSymbol{t}{\mathalpha}{\M@lower@font}{1044499}
2014       \DeclareMathSymbol{u}{\mathalpha}{\M@lower@font}{1044500}
2015       \DeclareMathSymbol{v}{\mathalpha}{\M@lower@font}{1044501}
2016       \DeclareMathSymbol{w}{\mathalpha}{\M@lower@font}{1044502}
2017       \DeclareMathSymbol{x}{\mathalpha}{\M@lower@font}{1044503}
2018       \DeclareMathSymbol{y}{\mathalpha}{\M@lower@font}{1044504}
```

```
2019     \DeclareMathSymbol{z}{\mathalpha}{\M@lower@font}{1044505}
2020     \DeclareMathSymbol{\imath}{\mathalpha}{\M@lower@font}{1044506}
2021     \DeclareMathSymbol{\jmath}{\mathalpha}{\M@lower@font}{1044507}
2022     \DeclareMathSymbol{\hbar}{\mathord}{\M@lower@font}{"127}}
2023 \else
2024   \def\M@lower@set{%
2025     \edef\M@lower@font{M\M@lowershape\@tempa}
2026     \DeclareMathSymbol{a}{\mathalpha}{\M@lower@font}{`a}
2027     \DeclareMathSymbol{b}{\mathalpha}{\M@lower@font}{`b}
2028     \DeclareMathSymbol{c}{\mathalpha}{\M@lower@font}{`c}
2029     \DeclareMathSymbol{d}{\mathalpha}{\M@lower@font}{`d}
2030     \DeclareMathSymbol{e}{\mathalpha}{\M@lower@font}{`e}
2031     \DeclareMathSymbol{f}{\mathalpha}{\M@lower@font}{`f}
2032     \DeclareMathSymbol{g}{\mathalpha}{\M@lower@font}{`g}
2033     \DeclareMathSymbol{h}{\mathalpha}{\M@lower@font}{`h}
2034     \DeclareMathSymbol{i}{\mathalpha}{\M@lower@font}{`i}
2035     \DeclareMathSymbol{j}{\mathalpha}{\M@lower@font}{`j}
2036     \DeclareMathSymbol{k}{\mathalpha}{\M@lower@font}{`k}
2037     \DeclareMathSymbol{l}{\mathalpha}{\M@lower@font}{`l}
2038     \DeclareMathSymbol{m}{\mathalpha}{\M@lower@font}{`m}
2039     \DeclareMathSymbol{n}{\mathalpha}{\M@lower@font}{`n}
2040     \DeclareMathSymbol{o}{\mathalpha}{\M@lower@font}{`o}
2041     \DeclareMathSymbol{p}{\mathalpha}{\M@lower@font}{`p}
2042     \DeclareMathSymbol{q}{\mathalpha}{\M@lower@font}{`q}
2043     \DeclareMathSymbol{r}{\mathalpha}{\M@lower@font}{`r}
2044     \DeclareMathSymbol{s}{\mathalpha}{\M@lower@font}{`s}
2045     \DeclareMathSymbol{t}{\mathalpha}{\M@lower@font}{`t}
2046     \DeclareMathSymbol{u}{\mathalpha}{\M@lower@font}{`u}
2047     \DeclareMathSymbol{v}{\mathalpha}{\M@lower@font}{`v}
2048     \DeclareMathSymbol{w}{\mathalpha}{\M@lower@font}{`w}
2049     \DeclareMathSymbol{x}{\mathalpha}{\M@lower@font}{`x}
2050     \DeclareMathSymbol{y}{\mathalpha}{\M@lower@font}{`y}
2051     \DeclareMathSymbol{z}{\mathalpha}{\M@lower@font}{`z}
2052     \DeclareMathSymbol{\imath}{\mathalpha}{\M@lower@font}{"131}
2053     \DeclareMathSymbol{\jmath}{\mathalpha}{\M@lower@font}{"237}
2054     \DeclareMathSymbol{\hbar}{\mathord}{\M@lower@font}{"127}}
2055 \fi
```

Set diacritics.

```
2056 \def\M@diacritics@set{%
2057   \edef\M@diacritics@font{M\M@diacriticsshape\@tempa}
2058   \DeclareMathAccent{\acute}{\mathalpha}{\M@diacritics@font}{"B4}
2059   \DeclareMathAccent{\aacute}{\mathalpha}{\M@diacritics@font}{"2DD}
2060   \DeclareMathAccent{\dot}{\mathalpha}{\M@diacritics@font}{"2D9}
2061   \DeclareMathAccent{\ddot}{\mathalpha}{\M@diacritics@font}{"A8}
2062   \DeclareMathAccent{\grave}{\mathalpha}{\M@diacritics@font}{"60}
2063   \DeclareMathAccent{\breve}{\mathalpha}{\M@diacritics@font}{"2D8}
2064   \DeclareMathAccent{\hat}{\mathalpha}{\M@diacritics@font}{"2C6}
```

```
2065  \DeclareMathAccent{\check}{\mathalpha}{\M@diacritics@font}{"2C7}
2066  \DeclareMathAccent{\bar}{\mathalpha}{\M@diacritics@font}{"2C9}
2067  \DeclareMathAccent{\mathring}{\mathalpha}{\M@diacritics@font}{"2DA}
2068  \DeclareMathAccent{\tilde}{\mathalpha}{\M@diacritics@font}{"2DC}}
```

Set capital Greek characters.

```
2069  \def\M@greekupper@set{%
2070    \edef\M@greekupper@font{M\M@greekuppershape\@tempa}
2071    \DeclareMathSymbol{\Alpha}{\mathalpha}{\M@greekupper@font}{"391}
2072    \DeclareMathSymbol{\Beta}{\mathalpha}{\M@greekupper@font}{"392}
2073    \DeclareMathSymbol{\Gamma}{\mathalpha}{\M@greekupper@font}{"393}
2074    \DeclareMathSymbol{\Delta}{\mathalpha}{\M@greekupper@font}{"394}
2075    \DeclareMathSymbol{\Epsilon}{\mathalpha}{\M@greekupper@font}{"395}
2076    \DeclareMathSymbol{\Zeta}{\mathalpha}{\M@greekupper@font}{"396}
2077    \DeclareMathSymbol{\Eta}{\mathalpha}{\M@greekupper@font}{"397}
2078    \DeclareMathSymbol{\Theta}{\mathalpha}{\M@greekupper@font}{"398}
2079    \DeclareMathSymbol{\Iota}{\mathalpha}{\M@greekupper@font}{"399}
2080    \DeclareMathSymbol{\Kappa}{\mathalpha}{\M@greekupper@font}{"39A}
2081    \DeclareMathSymbol{\Lambda}{\mathalpha}{\M@greekupper@font}{"39B}
2082    \DeclareMathSymbol{\Mu}{\mathalpha}{\M@greekupper@font}{"39C}
2083    \DeclareMathSymbol{\Nu}{\mathalpha}{\M@greekupper@font}{"39D}
2084    \DeclareMathSymbol{\Xi}{\mathalpha}{\M@greekupper@font}{"39E}
2085    \DeclareMathSymbol{\Omicron}{\mathalpha}{\M@greekupper@font}{"39F}
2086    \DeclareMathSymbol{\Pi}{\mathalpha}{\M@greekupper@font}{"3A0}
2087    \DeclareMathSymbol{\Rho}{\mathalpha}{\M@greekupper@font}{"3A1}
2088    \DeclareMathSymbol{\Sigma}{\mathalpha}{\M@greekupper@font}{"3A3}
2089    \DeclareMathSymbol{\Tau}{\mathalpha}{\M@greekupper@font}{"3A4}
2090    \DeclareMathSymbol{\Upsilon}{\mathalpha}{\M@greekupper@font}{"3A5}
2091    \DeclareMathSymbol{\Phi}{\mathalpha}{\M@greekupper@font}{"3A6}
2092    \DeclareMathSymbol{\Chi}{\mathalpha}{\M@greekupper@font}{"3A7}
2093    \DeclareMathSymbol{\Psi}{\mathalpha}{\M@greekupper@font}{"3A8}
2094    \DeclareMathSymbol{\Omega}{\mathalpha}{\M@greekupper@font}{"3A9}
2095    \DeclareMathSymbol{\varTheta}{\mathalpha}{\M@greekupper@font}{"3F4}
```

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

```
2096    \ifM@adjust@font
2097      \ifM@symbols\else
2098        \DeclareMathSymbol{\increment}{\mathord}{\M@greekupper@font}{"2206}
2099        \DeclareMathSymbol{\nabla}{\mathord}{\M@greekupper@font}{"2207}
2100      \fi
2101    \else
2102      \ifM@symbols\else
2103        \DeclareMathSymbol{\increment}{\mathord}{\M@greekupper@font}{"2206}
2104      \fi
2105      \ifM@extsymbols\else
2106        \DeclareMathSymbol{\nabla}{\mathord}{\M@greekupper@font}{"2207}
2107      \fi
2108    \fi}
```

Set minuscule Greek characters.

```
2109 \def\M@greeklower@set{%
2110   \edef\M@greeklower@font{M\M@greeklowershape\@tempa}
2111   \DeclareMathSymbol{\alpha}{\mathalpha}{\M@greeklower@font}{"3B1}
2112   \DeclareMathSymbol{\beta}{\mathalpha}{\M@greeklower@font}{"3B2}
2113   \DeclareMathSymbol{\gamma}{\mathalpha}{\M@greeklower@font}{"3B3}
2114   \DeclareMathSymbol{\delta}{\mathalpha}{\M@greeklower@font}{"3B4}
2115   \DeclareMathSymbol{\epsilon}{\mathalpha}{\M@greeklower@font}{"3B5}
2116   \DeclareMathSymbol{\zeta}{\mathalpha}{\M@greeklower@font}{"3B6}
2117   \DeclareMathSymbol{\eta}{\mathalpha}{\M@greeklower@font}{"3B7}
2118   \DeclareMathSymbol{\theta}{\mathalpha}{\M@greeklower@font}{"3B8}
2119   \DeclareMathSymbol{\iota}{\mathalpha}{\M@greeklower@font}{"3B9}
2120   \DeclareMathSymbol{\kappa}{\mathalpha}{\M@greeklower@font}{"3BA}
2121   \DeclareMathSymbol{\lambda}{\mathalpha}{\M@greeklower@font}{"3BB}
2122   \DeclareMathSymbol{\mu}{\mathalpha}{\M@greeklower@font}{"3BC}
2123   \DeclareMathSymbol{\nu}{\mathalpha}{\M@greeklower@font}{"3BD}
2124   \DeclareMathSymbol{\xi}{\mathalpha}{\M@greeklower@font}{"3BE}
2125   \DeclareMathSymbol{\omicron}{\mathalpha}{\M@greeklower@font}{"3BF}
2126   \DeclareMathSymbol{\pi}{\mathalpha}{\M@greeklower@font}{"3C0}
2127   \DeclareMathSymbol{\rho}{\mathalpha}{\M@greeklower@font}{"3C1}
2128   \DeclareMathSymbol{\sigma}{\mathalpha}{\M@greeklower@font}{"3C3}
2129   \DeclareMathSymbol{\tau}{\mathalpha}{\M@greeklower@font}{"3C4}
2130   \DeclareMathSymbol{\upsilon}{\mathalpha}{\M@greeklower@font}{"3C5}
2131   \DeclareMathSymbol{\phi}{\mathalpha}{\M@greeklower@font}{"3C6}
2132   \DeclareMathSymbol{\chi}{\mathalpha}{\M@greeklower@font}{"3C7}
2133   \DeclareMathSymbol{\psi}{\mathalpha}{\M@greeklower@font}{"3C8}
2134   \DeclareMathSymbol{\omega}{\mathalpha}{\M@greeklower@font}{"3C9}
2135   \DeclareMathSymbol{\varbeta}{\mathalpha}{\M@greeklower@font}{"3D0}
2136   \DeclareMathSymbol{\varepsilon}{\mathalpha}{\M@greeklower@font}{"3F5}
2137   \DeclareMathSymbol{\varkappa}{\mathalpha}{\M@greeklower@font}{"3F0}
2138   \DeclareMathSymbol{\vartheta}{\mathalpha}{\M@greeklower@font}{"3D1}
2139   \DeclareMathSymbol{\varrho}{\mathalpha}{\M@greeklower@font}{"3F1}
2140   \DeclareMathSymbol{\varsigma}{\mathalpha}{\M@greeklower@font}{"3C2}
2141   \DeclareMathSymbol{\varphi}{\mathalpha}{\M@greeklower@font}{"3D5}}
```

Set capital ancient Greek characters.

```
2142 \def\M@agreekupper@set{%
2143   \edef\M@agreekupper@font{M\M@agreekuppershape\@tempa}
2144   \DeclareMathSymbol{\Heta}{\mathalpha}{\M@agreekupper@font}{"370}
2145   \DeclareMathSymbol{\Sampi}{\mathalpha}{\M@agreekupper@font}{"3E0}
2146   \DeclareMathSymbol{\Digamma}{\mathalpha}{\M@agreekupper@font}{"3DC}
2147   \DeclareMathSymbol{\Koppa}{\mathalpha}{\M@agreekupper@font}{"3D8}
2148   \DeclareMathSymbol{\Stigma}{\mathalpha}{\M@agreekupper@font}{"3DA}
2149   \DeclareMathSymbol{\Sho}{\mathalpha}{\M@agreekupper@font}{"3F7}
2150   \DeclareMathSymbol{\San}{\mathalpha}{\M@agreekupper@font}{"3FA}
2151   \DeclareMathSymbol{\varSampi}{\mathalpha}{\M@agreekupper@font}{"372}
2152   \DeclareMathSymbol{\varDigamma}{\mathalpha}{\M@agreekupper@font}{"376}
2153   \DeclareMathSymbol{\varKoppa}{\mathalpha}{\M@agreekupper@font}{"3DE}}
```

Set minuscule ancient Greek characters.

```
2154 \def\M@agreeklower@set{%
2155   \edef\M@agreeklower@font{M\M@agreeklowershape\@tempa}
2156   \DeclareMathSymbol{\heta}{\mathalpha}{\M@agreeklower@font}{"371}
2157   \DeclareMathSymbol{\sampi}{\mathalpha}{\M@agreeklower@font}{"3E1}
2158   \DeclareMathSymbol{\digamma}{\mathalpha}{\M@agreeklower@font}{"3DD}
2159   \DeclareMathSymbol{\koppa}{\mathalpha}{\M@agreeklower@font}{"3D9}
2160   \DeclareMathSymbol{\stigma}{\mathalpha}{\M@agreeklower@font}{"3DB}
2161   \DeclareMathSymbol{\sho}{\mathalpha}{\M@agreeklower@font}{"3F8}
2162   \DeclareMathSymbol{\san}{\mathalpha}{\M@agreeklower@font}{"3FB}
2163   \DeclareMathSymbol{\varsampi}{\mathalpha}{\M@agreeklower@font}{"373}
2164   \DeclareMathSymbol{\vardigamma}{\mathalpha}{\M@agreeklower@font}{"377}
2165   \DeclareMathSymbol{\varkoppa}{\mathalpha}{\M@agreeklower@font}{"3DF}}
```

Set capital Cyrillic characters.

```
2166 \def\M@cyrillicupper@set{%
2167   \edef\M@cyrillicupper@font{M\M@cyrillicuppershape\@tempa}
2168   \DeclareMathSymbol{\cyrA}{\mathalpha}{\M@cyrillicupper@font}{"410}
2169   \DeclareMathSymbol{\cyrBe}{\mathalpha}{\M@cyrillicupper@font}{"411}
2170   \DeclareMathSymbol{\cyrVe}{\mathalpha}{\M@cyrillicupper@font}{"412}
2171   \DeclareMathSymbol{\cyrGhe}{\mathalpha}{\M@cyrillicupper@font}{"413}
2172   \DeclareMathSymbol{\cyrDe}{\mathalpha}{\M@cyrillicupper@font}{"414}
2173   \DeclareMathSymbol{\cyrIe}{\mathalpha}{\M@cyrillicupper@font}{"415}
2174   \DeclareMathSymbol{\cyrZhe}{\mathalpha}{\M@cyrillicupper@font}{"416}
2175   \DeclareMathSymbol{\cyrZe}{\mathalpha}{\M@cyrillicupper@font}{"417}
2176   \DeclareMathSymbol{\cyrI}{\mathalpha}{\M@cyrillicupper@font}{"418}
2177   \DeclareMathSymbol{\cyrKa}{\mathalpha}{\M@cyrillicupper@font}{"41A}
2178   \DeclareMathSymbol{\cyrEl}{\mathalpha}{\M@cyrillicupper@font}{"41B}
2179   \DeclareMathSymbol{\cyrEm}{\mathalpha}{\M@cyrillicupper@font}{"41C}
2180   \DeclareMathSymbol{\cyrEn}{\mathalpha}{\M@cyrillicupper@font}{"41D}
2181   \DeclareMathSymbol{\cyrO}{\mathalpha}{\M@cyrillicupper@font}{"41E}
2182   \DeclareMathSymbol{\cyrPe}{\mathalpha}{\M@cyrillicupper@font}{"41F}
2183   \DeclareMathSymbol{\cyrEr}{\mathalpha}{\M@cyrillicupper@font}{"420}
2184   \DeclareMathSymbol{\cyrEs}{\mathalpha}{\M@cyrillicupper@font}{"421}
2185   \DeclareMathSymbol{\cyrTe}{\mathalpha}{\M@cyrillicupper@font}{"422}
2186   \DeclareMathSymbol{\cyrU}{\mathalpha}{\M@cyrillicupper@font}{"423}
2187   \DeclareMathSymbol{\cyrEf}{\mathalpha}{\M@cyrillicupper@font}{"424}
2188   \DeclareMathSymbol{\cyrHa}{\mathalpha}{\M@cyrillicupper@font}{"425}
2189   \DeclareMathSymbol{\cyrTse}{\mathalpha}{\M@cyrillicupper@font}{"426}
2190   \DeclareMathSymbol{\cyrChe}{\mathalpha}{\M@cyrillicupper@font}{"427}
2191   \DeclareMathSymbol{\cyrSha}{\mathalpha}{\M@cyrillicupper@font}{"428}
2192   \DeclareMathSymbol{\cyrShcha}{\mathalpha}{\M@cyrillicupper@font}{"429}
2193   \DeclareMathSymbol{\cyrHard}{\mathalpha}{\M@cyrillicupper@font}{"42A}
2194   \DeclareMathSymbol{\cyrYeru}{\mathalpha}{\M@cyrillicupper@font}{"42B}
2195   \DeclareMathSymbol{\cyrSoft}{\mathalpha}{\M@cyrillicupper@font}{"42C}
2196   \DeclareMathSymbol{\cyrE}{\mathalpha}{\M@cyrillicupper@font}{"42D}
2197   \DeclareMathSymbol{\cyrYu}{\mathalpha}{\M@cyrillicupper@font}{"42E}
2198   \DeclareMathSymbol{\cyrYa}{\mathalpha}{\M@cyrillicupper@font}{"42F}
```

2199   \DeclareMathSymbol{\cyrvarI}{\mathalpha}{\M@cyrillicupper@font}{"419}}

Set minuscule Cyrillic characters.

2200 \def\M@cyrilliclower@set{%
2201   \edef\M@cyrilliclower@font{M\M@cyrilliclowershape\@tempa}
2202   \DeclareMathSymbol{\cyra}{\mathalpha}{\M@cyrilliclower@font}{"430}
2203   \DeclareMathSymbol{\cyrbe}{\mathalpha}{\M@cyrilliclower@font}{"431}
2204   \DeclareMathSymbol{\cyrve}{\mathalpha}{\M@cyrilliclower@font}{"432}
2205   \DeclareMathSymbol{\cyrghe}{\mathalpha}{\M@cyrilliclower@font}{"433}
2206   \DeclareMathSymbol{\cyrde}{\mathalpha}{\M@cyrilliclower@font}{"434}
2207   \DeclareMathSymbol{\cyrie}{\mathalpha}{\M@cyrilliclower@font}{"435}
2208   \DeclareMathSymbol{\cyrzhe}{\mathalpha}{\M@cyrilliclower@font}{"436}
2209   \DeclareMathSymbol{\cyrze}{\mathalpha}{\M@cyrilliclower@font}{"437}
2210   \DeclareMathSymbol{\cyri}{\mathalpha}{\M@cyrilliclower@font}{"438}
2211   \DeclareMathSymbol{\cyrka}{\mathalpha}{\M@cyrilliclower@font}{"43A}
2212   \DeclareMathSymbol{\cyrel}{\mathalpha}{\M@cyrilliclower@font}{"43B}
2213   \DeclareMathSymbol{\cyrem}{\mathalpha}{\M@cyrilliclower@font}{"43C}
2214   \DeclareMathSymbol{\cyren}{\mathalpha}{\M@cyrilliclower@font}{"43D}
2215   \DeclareMathSymbol{\cyro}{\mathalpha}{\M@cyrilliclower@font}{"43E}
2216   \DeclareMathSymbol{\cyrpe}{\mathalpha}{\M@cyrilliclower@font}{"43F}
2217   \DeclareMathSymbol{\cyrer}{\mathalpha}{\M@cyrilliclower@font}{"440}
2218   \DeclareMathSymbol{\cyres}{\mathalpha}{\M@cyrilliclower@font}{"441}
2219   \DeclareMathSymbol{\cyrte}{\mathalpha}{\M@cyrilliclower@font}{"442}
2220   \DeclareMathSymbol{\cyru}{\mathalpha}{\M@cyrilliclower@font}{"443}
2221   \DeclareMathSymbol{\cyref}{\mathalpha}{\M@cyrilliclower@font}{"444}
2222   \DeclareMathSymbol{\cyrha}{\mathalpha}{\M@cyrilliclower@font}{"445}
2223   \DeclareMathSymbol{\cyrtse}{\mathalpha}{\M@cyrilliclower@font}{"446}
2224   \DeclareMathSymbol{\cyrche}{\mathalpha}{\M@cyrilliclower@font}{"447}
2225   \DeclareMathSymbol{\cyrsha}{\mathalpha}{\M@cyrilliclower@font}{"448}
2226   \DeclareMathSymbol{\cyrshcha}{\mathalpha}{\M@cyrilliclower@font}{"449}
2227   \DeclareMathSymbol{\cyrhard}{\mathalpha}{\M@cyrilliclower@font}{"44A}
2228   \DeclareMathSymbol{\cyryeru}{\mathalpha}{\M@cyrilliclower@font}{"44B}
2229   \DeclareMathSymbol{\cyrsoft}{\mathalpha}{\M@cyrilliclower@font}{"44C}
2230   \DeclareMathSymbol{\cyre}{\mathalpha}{\M@cyrilliclower@font}{"44D}
2231   \DeclareMathSymbol{\cyryu}{\mathalpha}{\M@cyrilliclower@font}{"44E}
2232   \DeclareMathSymbol{\cyrya}{\mathalpha}{\M@cyrilliclower@font}{"44F}
2233   \DeclareMathSymbol{\cyrvari}{\mathalpha}{\M@cyrilliclower@font}{"439}}

Set Hebrew characters.

2234 \def\M@hebrew@set{%
2235   \edef\M@hebrew@font{M\M@hebrewshape\@tempa}
2236   \DeclareMathSymbol{\aleph}{\mathalpha}{\M@hebrew@font}{"5D0}
2237   \DeclareMathSymbol{\beth}{\mathalpha}{\M@hebrew@font}{"5D1}
2238   \DeclareMathSymbol{\gimel}{\mathalpha}{\M@hebrew@font}{"5D2}
2239   \DeclareMathSymbol{\daleth}{\mathalpha}{\M@hebrew@font}{"5D3}
2240   \DeclareMathSymbol{\he}{\mathalpha}{\M@hebrew@font}{"5D4}
2241   \DeclareMathSymbol{\vav}{\mathalpha}{\M@hebrew@font}{"5D5}
2242   \DeclareMathSymbol{\zayin}{\mathalpha}{\M@hebrew@font}{"5D6}
2243   \DeclareMathSymbol{\het}{\mathalpha}{\M@hebrew@font}{"5D7}

```
2244  \DeclareMathSymbol{\tet}{\mathalpha}{\M@hebrew@font}{"5D8}
2245  \DeclareMathSymbol{\yod}{\mathalpha}{\M@hebrew@font}{"5D9}
2246  \DeclareMathSymbol{\kaf}{\mathalpha}{\M@hebrew@font}{"5DB}
2247  \DeclareMathSymbol{\lamed}{\mathalpha}{\M@hebrew@font}{"5DC}
2248  \DeclareMathSymbol{\mem}{\mathalpha}{\M@hebrew@font}{"5DE}
2249  \DeclareMathSymbol{\nun}{\mathalpha}{\M@hebrew@font}{"5E0}
2250  \DeclareMathSymbol{\samekh}{\mathalpha}{\M@hebrew@font}{"5E1}
2251  \DeclareMathSymbol{\ayin}{\mathalpha}{\M@hebrew@font}{"5E2}
2252  \DeclareMathSymbol{\pe}{\mathalpha}{\M@hebrew@font}{"5E4}
2253  \DeclareMathSymbol{\tsadi}{\mathalpha}{\M@hebrew@font}{"5E6}
2254  \DeclareMathSymbol{\qof}{\mathalpha}{\M@hebrew@font}{"5E7}
2255  \DeclareMathSymbol{\resh}{\mathalpha}{\M@hebrew@font}{"5E8}
2256  \DeclareMathSymbol{\shin}{\mathalpha}{\M@hebrew@font}{"5E9}
2257  \DeclareMathSymbol{\tav}{\mathalpha}{\M@hebrew@font}{"5EA}
2258  \DeclareMathSymbol{\varkaf}{\mathalpha}{\M@hebrew@font}{"5DA}
2259  \DeclareMathSymbol{\varmem}{\mathalpha}{\M@hebrew@font}{"5DD}
2260  \DeclareMathSymbol{\varnun}{\mathalpha}{\M@hebrew@font}{"5DF}
2261  \DeclareMathSymbol{\varpe}{\mathalpha}{\M@hebrew@font}{"5E3}
2262  \DeclareMathSymbol{\vartsadi}{\mathalpha}{\M@hebrew@font}{"5E5}}
```

Set digits.

```
2263  \def\M@digits@set{%
2264    \edef\M@digits@font{M\M@digitsshape\@tempa}
2265    \DeclareMathSymbol{0}{\mathalpha}{\M@digits@font}{`0}
2266    \DeclareMathSymbol{1}{\mathalpha}{\M@digits@font}{`1}
2267    \DeclareMathSymbol{2}{\mathalpha}{\M@digits@font}{`2}
2268    \DeclareMathSymbol{3}{\mathalpha}{\M@digits@font}{`3}
2269    \DeclareMathSymbol{4}{\mathalpha}{\M@digits@font}{`4}
2270    \DeclareMathSymbol{5}{\mathalpha}{\M@digits@font}{`5}
2271    \DeclareMathSymbol{6}{\mathalpha}{\M@digits@font}{`6}
2272    \DeclareMathSymbol{7}{\mathalpha}{\M@digits@font}{`7}
2273    \DeclareMathSymbol{8}{\mathalpha}{\M@digits@font}{`8}
2274    \DeclareMathSymbol{9}{\mathalpha}{\M@digits@font}{`9}}
```

Set new operator font. If mathfont is set to adjust fonts, we will have a problem when type-
setting operators because the \operator@font will pull modified (lengthened) letters from
the operator font. Traditional TEX addressed this problem by storing the Latin letters for
math in the same endoding 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.

```
2275  \def\M@operator@set{%
2276    \ifM@adjust@font
2277      \edef\M@operator@num{\number\csname symM\M@operatorshape\@tempa\endcsname}
2278      \protected\edef\M@operator@mathcodes{%
2279        \Umathcode`A=7+\M@operator@num+`A\relax
2280        \Umathcode`B=7+\M@operator@num+`B\relax
```

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

```
2328        \Umathcode`x=7+\M@operator@num+`x\relax
2329        \Umathcode`y=7+\M@operator@num+`y\relax
2330        \Umathcode`z=7+\M@operator@num+`z\relax
2331        \Umathchardef\imath=7+\M@operator@num+1044506\relax
2332        \Umathchardef\jmath=7+\M@operator@num+1044500\relax}
2333    \else
2334      \let\M@operator@mathcodes\@empty
2335    \fi
```

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

```
2336    \xdef\operator@font{\noexpand\mathgroup
2337        \csname symM\M@operatorshape\@tempa\endcsname\M@operator@mathcodes}}
```

Set delimiters.

```
2338  \ifM@adjust@font
2339    \def\M@delimiters@set{%
2340      \edef\M@delimiters@font{M\M@delimitersshape\@tempa}
2341      \DeclareMathSymbol{(}{\mathopen}{\M@delimiters@font}{"28}
2342      \DeclareMathSymbol{)}{\mathclose}{\M@delimiters@font}{"29}
2343      \DeclareMathSymbol{[}{\mathopen}{\M@delimiters@font}{"5B}
2344      \DeclareMathSymbol{]}{\mathclose}{\M@delimiters@font}{"5D}
2345      \ifM@symbols\else
2346        \DeclareMathSymbol{|}{\mathord}{\M@delimiters@font}{"7C}
2347      \fi
2348      \DeclareMathSymbol{\leftbrace}{\mathopen}{\M@delimiters@font}{"7B}
2349      \DeclareMathSymbol{\rightbrace}{\mathclose}{\M@delimiters@font}{"7D}
2350      \global\Udelcode40=+\csname sym\M@delimiters@font\endcsname+40\relax % (
2351      \global\Udelcode41=+\csname sym\M@delimiters@font\endcsname+41\relax % )
2352      \global\Udelcode47=+\csname sym\M@delimiters@font\endcsname+47\relax % /
2353      \global\Udelcode91=+\csname sym\M@delimiters@font\endcsname+91\relax % [
2354      \global\Udelcode93=+\csname sym\M@delimiters@font\endcsname+93\relax % ]
2355      \global\Udelcode124=+\csname sym\M@delimiters@font\endcsname+124\relax % |
2356      \global\let\vert=|
2357      \protected\gdef\backslash{\ifmmode\mathbackslash\else\textbackslash\fi}
2358      \protected\xdef\mathbackslash{%
2359        \Udelimiter+2+\number\csname sym\M@delimiters@font\endcsname
2360          +92\relax} % backslash
2361      \protected\xdef\lbrace{%
2362        \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname
2363          +123\relax} % {
2364      \protected\xdef\rbrace{%
2365        \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
2366          +125\relax} % }
2367      \protected\xdef\lguil{%
2368        \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname
2369          +8249\relax} % single left guilement
2370      \protected\xdef\rguil{%
2371        \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
2372          +8250\relax} % single right guilement
```

```
2373      \protected\xdef\llguil{%
2374        \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname
2375          +171\relax} % double left guilement
2376      \protected\xdef\rrguil{%
2377        \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
2378          +187\relax} % double right guilement
2379      \protected\xdef\fakelangle{%
2380        \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname
2381          +1044508\relax} % fake left angle
2382      \protected\xdef\fakerangle{%
2383        \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
2384          +1044509\relax} % fake right angle
2385      \protected\xdef\fakellangle{%
2386        \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname
2387          +1044510\relax} % fake double left angle
2388      \protected\xdef\fakerrangle{%
2389        \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
2390          +1044511\relax} % fake double right angle
2391      }
2392  \else
2393    \def\M@delimiters@set{%
2394      \edef\M@delimiters@font{M\M@delimitersshape\@tempa}
2395      \DeclareMathSymbol{(}{\mathopen}{\M@delimiters@font}{"28}
2396      \DeclareMathSymbol{)}{\mathclose}{\M@delimiters@font}{"29}
2397      \DeclareMathSymbol{[}{\mathopen}{\M@delimiters@font}{"5B}
2398      \DeclareMathSymbol{]}{\mathclose}{\M@delimiters@font}{"5D}
2399      \DeclareMathSymbol{\lguil}{\mathopen}{\M@delimiters@font}{"2039}
2400      \DeclareMathSymbol{\rguil}{\mathclose}{\M@delimiters@font}{"203A}
2401      \DeclareMathSymbol{\llguil}{\mathopen}{\M@delimiters@font}{"AB}
2402      \DeclareMathSymbol{\rrguil}{\mathclose}{\M@delimiters@font}{"BB}
2403      \DeclareMathSymbol{\leftbrace}{\mathopen}{\M@delimiters@font}{"7B}
2404      \DeclareMathSymbol{\rightbrace}{\mathclose}{\M@delimiters@font}{"7D}}
2405  \fi
```

Radicals.

```
2406  \ifM@adjust@font
2407    \def\M@radical@set{%
2408      \edef\M@radical@font{M\M@radicalshape\@tempa}
2409      \DeclareMathSymbol{\surd}{\mathord}{\M@radical@font}{"221A}
2410      \xdef\@sqrts@gn##1{%
2411        \Uradical+\number\csname sym\M@radical@font\endcsname+8730\relax{##1}}
```

We redefine \r@@t, which typesets the degree symbol on an nth 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.

```
2412      \gdef\r@@t##1##2{%
2413        \setbox\z@\hbox{$\m@th##1\sqrtsign{##2}$}%
2414        \setbox\surdbox\hbox{$\m@th##1\@sqrts@gn{%
```

```
2415        \hbox{\vphantom{$\m@th##1##2$}}}}$}
2416      \dimen@\ht\surdbox
2417      \advance\dimen@\dp\surdbox
2418      \dimen@=0.6\dimen@
2419      \advance\dimen@-\dp\surdbox
2420      \ifdim\wd\rootbox<0.6\wd\surdbox
2421        \kern0.6\wd\surdbox
2422      \else
2423        \kern\wd\rootbox
2424      \fi
2425      \raise\dimen@\hbox{\llap{\copy\rootbox}}
2426      \kern-0.6\wd\surdbox
2427      \box\z@}
2428    \gdef\sqrtsign##1{\@sqrts@gn{\mkern\radicandoffset##1}}}}
2429 \else
2430   \def\M@radical@set{%
2431     \edef\M@radical@font{M\M@radicalshape\@tempa}
2432     \DeclareMathSymbol{\surd}{\mathord}{\M@radical@font}{"221A}}
2433 \fi
```
Big operators.
```
2434 \def\M@bigops@set{%
2435   \edef\M@bigops@font{M\M@bigopsshape\@tempa}
2436   \let\sum\@undefined
2437   \let\prod\@undefined
2438   \DeclareMathSymbol{\sum}{\mathop}{\M@bigops@font}{"2211}
2439   \DeclareMathSymbol{\prod}{\mathop}{\M@bigops@font}{"220F}
2440   \DeclareMathSymbol{\intop}{\mathop}{\M@bigops@font}{"222B}}
```
Extended big operators.
```
2441 \def\M@extbigops@set{%
2442   \edef\M@extbigops@font{M\M@extbigopsshape\@tempa}
2443   \let\coprod\@undefined
2444   \let\bigvee\@undefined
2445   \let\bigwedge\@undefined
2446   \let\bigcup\@undefined
2447   \let\bigcap\@undefined
2448   \let\bigoplus\@undefined
2449   \let\bigotimes\@undefined
2450   \let\bigodot\@undefined
2451   \let\bigsqcup\@undefined
2452   \DeclareMathSymbol{\coprod}{\mathop}{\M@extbigops@font}{"2210}
2453   \DeclareMathSymbol{\bigvee}{\mathop}{\M@extbigops@font}{"22C1}
2454   \DeclareMathSymbol{\bigwedge}{\mathop}{\M@extbigops@font}{"22C0}
2455   \DeclareMathSymbol{\bigcup}{\mathop}{\M@extbigops@font}{"22C3}
2456   \DeclareMathSymbol{\bigcap}{\mathop}{\M@extbigops@font}{"22C2}
2457   \DeclareMathSymbol{\iintop}{\mathop}{\M@extbigops@font}{"222C}
2458     \protected\gdef\iint{\iintop\nolimits}
2459   \DeclareMathSymbol{\iiintop}{\mathop}{\M@extbigops@font}{"222D}
```

```
2460      \protected\gdef\iiint{\iiintop\nolimits}
2461   \DeclareMathSymbol{\ointop}{\mathop}{\M@extbigops@font}{"222E}
2462      \protected\gdef\oint{\ointop\nolimits}
2463   \DeclareMathSymbol{\oiintop}{\mathop}{\M@extbigops@font}{"222F}
2464      \protected\gdef\oiint{\oiintop\nolimits}
2465   \DeclareMathSymbol{\oiiintop}{\mathop}{\M@extbigops@font}{"2230}
2466      \protected\gdef\oiiint{\oiiintop\nolimits}
2467   \DeclareMathSymbol{\bigoplus}{\mathop}{\M@extbigops@font}{"2A01}
2468   \DeclareMathSymbol{\bigotimes}{\mathop}{\M@extbigops@font}{"2A02}
2469   \DeclareMathSymbol{\bigodot}{\mathop}{\M@extbigops@font}{"2A00}
2470   \DeclareMathSymbol{\bigsqcap}{\mathop}{\M@extbigops@font}{"2A05}
2471   \DeclareMathSymbol{\bigsqcup}{\mathop}{\M@extbigops@font}{"2A06}}
```

Set symbols.

```
2472 \def\M@symbols@set{%
2473   \edef\M@symbols@font{M\M@symbolsshape\@tempa}
2474   \let\colon\@undefined
2475   \let\mathellipsis\@undefined
2476   \DeclareMathSymbol{.}{\mathord}{\M@symbols@font}{"2E}
2477   \DeclareMathSymbol{@}{\mathord}{\M@symbols@font}{"40}
2478   \DeclareMathSymbol{'}{\mathord}{\M@symbols@font}{"2032}
2479   \DeclareMathSymbol{\prime}{\mathord}{\M@symbols@font}{"2032}
2480   \DeclareMathSymbol{"}{\mathord}{\M@symbols@font}{"2033}
2481   \DeclareMathSymbol{\mathhash}{\mathord}{\M@symbols@font}{"23}
2482   \DeclareMathSymbol{\mathdollar}{\mathord}{\M@symbols@font}{"24}
2483   \DeclareMathSymbol{\mathpercent}{\mathord}{\M@symbols@font}{"25}
2484   \DeclareMathSymbol{\mathand}{\mathord}{\M@symbols@font}{"26}
2485   \DeclareMathSymbol{\mathparagraph}{\mathord}{\M@symbols@font}{"B6}
2486   \DeclareMathSymbol{\mathsection}{\mathord}{\M@symbols@font}{"A7}
2487   \DeclareMathSymbol{\mathsterling}{\mathord}{\M@symbols@font}{"A3}
2488   \DeclareMathSymbol{\neg}{\mathord}{\M@symbols@font}{"AC}
2489   \DeclareMathSymbol{|}{\mathord}{\M@symbols@font}{"7C}
2490   \DeclareMathSymbol{\infty}{\mathord}{\M@symbols@font}{"221E}
2491   \DeclareMathSymbol{\partial}{\mathord}{\M@symbols@font}{"2202}
2492   \DeclareMathSymbol{\degree}{\mathord}{\M@symbols@font}{"B0}
2493   \DeclareMathSymbol{\increment}{\mathord}{\M@symbols@font}{"2206}
2494   \DeclareMathSymbol{\comma}{\mathord}{\M@symbols@font}{"2C}
2495   \DeclareMathSymbol{+}{\mathbin}{\M@symbols@font}{"2B}
2496   \DeclareMathSymbol{-}{\mathbin}{\M@symbols@font}{"2212}
2497   \DeclareMathSymbol{*}{\mathbin}{\M@symbols@font}{"2A}
2498   \DeclareMathSymbol{\times}{\mathbin}{\M@symbols@font}{"D7}
2499   \DeclareMathSymbol{/}{\mathbin}{\M@symbols@font}{"2F}
2500   \DeclareMathSymbol{\fractionslash}{\mathbin}{\M@symbols@font}{"2215}
2501   \DeclareMathSymbol{\div}{\mathbin}{\M@symbols@font}{"F7}
2502   \DeclareMathSymbol{\pm}{\mathbin}{\M@symbols@font}{"B1}
2503   \DeclareMathSymbol{\bullet}{\mathbin}{\M@symbols@font}{"2022}
2504   \DeclareMathSymbol{\dagger}{\mathbin}{\M@symbols@font}{"2020}
2505   \DeclareMathSymbol{\ddagger}{\mathbin}{\M@symbols@font}{"2021}
```

```
2506    \DeclareMathSymbol{\cdot}{\mathbin}{\M@symbols@font}{"2219}
2507    \DeclareMathSymbol{\setminus}{\mathbin}{\M@symbols@font}{"5C}
2508    \DeclareMathSymbol{=}{\mathrel}{\M@symbols@font}{"3D}
2509    \DeclareMathSymbol{<}{\mathrel}{\M@symbols@font}{"3C}
2510    \DeclareMathSymbol{>}{\mathrel}{\M@symbols@font}{"3E}
2511    \DeclareMathSymbol{\leq}{\mathrel}{\M@symbols@font}{"2264}
2512    \DeclareMathSymbol{\geq}{\mathrel}{\M@symbols@font}{"2265}
2513    \DeclareMathSymbol{\sim}{\mathrel}{\M@symbols@font}{"7E}
2514    \DeclareMathSymbol{\approx}{\mathrel}{\M@symbols@font}{"2248}
2515    \DeclareMathSymbol{\equiv}{\mathrel}{\M@symbols@font}{"2261}
2516    \DeclareMathSymbol{\mid}{\mathrel}{\M@symbols@font}{"7C}
2517    \DeclareMathSymbol{\parallel}{\mathrel}{\M@symbols@font}{"2016}
2518    \DeclareMathSymbol{:}{\mathrel}{\M@symbols@font}{"3A}
2519    \DeclareMathSymbol{?}{\mathclose}{\M@symbols@font}{"3F}
2520    \DeclareMathSymbol{!}{\mathclose}{\M@symbols@font}{"21}
2521    \DeclareMathSymbol{,}{\mathpunct}{\M@symbols@font}{"2C}
2522    \DeclareMathSymbol{;}{\mathpunct}{\M@symbols@font}{"3B}
2523    \DeclareMathSymbol{\colon}{\mathord}{\M@symbols@font}{"3A}
2524    \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 charcters used in several math-mode symbols.

```
2525    \protected\gdef\#{\ifmmode\mathhash\else\char"23\relax\fi}
2526    \protected\gdef\%{\ifmmode\mathpercent\else\char"25\relax\fi}
2527    \protected\gdef\&{\ifmmode\mathand\else\char"26\relax\fi}
2528    \DeclareMathSymbol{\@relbar}{\mathbin}{symbols}{"00}
2529    \DeclareMathSymbol{\@Relbar}{\mathrel}{operators}{"3D}
2530    \DeclareMathSymbol{\@verticalbar}{\mathord}{symbols}{"6A}
2531    \ifM@extsymbols\else
2532        \protected\gdef\simeq{\mathrel{\mathpalette\stack@flatrel{{-}{\sim}}}}
2533        \protected\gdef\cong{\mathrel{\mathpalette\stack@flatrel{{=}{\sim}}}}
2534    \fi
2535    \protected\gdef\relbar{\mathrel{\smash\@relbar}}
2536    \protected\gdef\Relbar{\mathrel{\@Relbar}}
2537    \protected\gdef\models{\mathrel{\@verticalbar}\joinrel\Relbar}
```

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

```
2538    \ifM@adjust@font
2539        \DeclareMathSymbol{\bigat}{\mathop}{\M@symbols@font}{"40}
2540        \DeclareMathSymbol{\bighash}{\mathop}{\M@symbols@font}{"23}
2541        \DeclareMathSymbol{\bigdollar}{\mathop}{\M@symbols@font}{"24}
2542        \DeclareMathSymbol{\bigpercent}{\mathop}{\M@symbols@font}{"25}
2543        \DeclareMathSymbol{\bigand}{\mathop}{\M@symbols@font}{"26}
```

```
2544    \DeclareMathSymbol{\bigplus}{\mathop}{\M@symbols@font}{"2B}
2545    \DeclareMathSymbol{\bigp}{\mathop}{\M@symbols@font}{"21}
2546    \DeclareMathSymbol{\bigq}{\mathop}{\M@symbols@font}{"3F}
2547    \DeclareMathSymbol{\bigS}{\mathop}{\M@symbols@font}{"A7}
2548    \DeclareMathSymbol{\bigtimes}{\mathop}{\M@symbols@font}{"D7}
2549    \DeclareMathSymbol{\bigdiv}{\mathop}{\M@symbols@font}{"F7}
```

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

```
2550    \DeclareMathSymbol{\nabla}{\mathord}{\M@symbols@font}{"2207}
2551  \fi}
```

Set extended symbols.

```
2552 \def\M@extsymbols@set{%
2553   \edef\M@extsymbols@font{M\M@extsymbolsshape\@tempa}
2554   \let\angle\@undefined
2555   \let\simeq\@undefined
2556   \let\sqsubset\@undefined
2557   \let\sqsupset\@undefined
2558   \let\bowtie\@undefined
2559   \let\doteq\@undefined
2560   \let\neq\@undefined
2561   \let\ng\@undefined
2562   \DeclareMathSymbol{\wp}{\mathord}{\M@extsymbols@font}{"2118}
2563   \DeclareMathSymbol{\Re}{\mathord}{\M@extsymbols@font}{"211C}
2564   \DeclareMathSymbol{\Im}{\mathord}{\M@extsymbols@font}{"2111}
2565   \DeclareMathSymbol{\ell}{\mathord}{\M@extsymbols@font}{"2113}
2566   \DeclareMathSymbol{\forall}{\mathord}{\M@extsymbols@font}{"2200}
2567   \DeclareMathSymbol{\exists}{\mathord}{\M@extsymbols@font}{"2203}
2568   \DeclareMathSymbol{\emptyset}{\mathord}{\M@extsymbols@font}{"2205}
2569   \DeclareMathSymbol{\in}{\mathord}{\M@extsymbols@font}{"2208}
2570   \DeclareMathSymbol{\ni}{\mathord}{\M@extsymbols@font}{"220B}
2571   \DeclareMathSymbol{\mp}{\mathord}{\M@extsymbols@font}{"2213}
2572   \DeclareMathSymbol{\angle}{\mathord}{\M@extsymbols@font}{"2220}
2573   \DeclareMathSymbol{\top}{\mathord}{\M@extsymbols@font}{"22A4}
2574   \DeclareMathSymbol{\bot}{\mathord}{\M@extsymbols@font}{"22A5}
2575   \DeclareMathSymbol{\vdash}{\mathord}{\M@extsymbols@font}{"22A2}
2576   \DeclareMathSymbol{\dashv}{\mathord}{\M@extsymbols@font}{"22A3}
2577   \DeclareMathSymbol{\flat}{\mathord}{\M@extsymbols@font}{"266D}
2578   \DeclareMathSymbol{\natural}{\mathord}{\M@extsymbols@font}{"266E}
2579   \DeclareMathSymbol{\sharp}{\mathord}{\M@extsymbols@font}{"266F}
2580   \DeclareMathSymbol{\fflat}{\mathord}{\M@extsymbols@font}{"1D12B}
2581   \DeclareMathSymbol{\ssharp}{\mathord}{\M@extsymbols@font}{"1D12A}
2582   \DeclareMathSymbol{\bclubsuit}{\mathord}{\M@extsymbols@font}{"2663}
2583   \DeclareMathSymbol{\bdiamondsuit}{\mathord}{\M@extsymbols@font}{"2666}
2584   \DeclareMathSymbol{\bheartsuit}{\mathord}{\M@extsymbols@font}{"2665}
2585   \DeclareMathSymbol{\bspadesuit}{\mathord}{\M@extsymbols@font}{"2660}
2586   \DeclareMathSymbol{\wclubsuit}{\mathord}{\M@extsymbols@font}{"2667}
2587   \DeclareMathSymbol{\wdiamondsuit}{\mathord}{\M@extsymbols@font}{"2662}
2588   \DeclareMathSymbol{\wheartsuit}{\mathord}{\M@extsymbols@font}{"2661}
```

```
2589  \DeclareMathSymbol{\wspadesuit}{\mathord}{\M@extsymbols@font}{"2664}
2590    \global\let\spadesuit\bspadesuit
2591    \global\let\heartsuit\wheartsuit
2592    \global\let\diamondsuit\wdiamondsuit
2593    \global\let\clubsuit\bclubsuit
2594  \DeclareMathSymbol{\wedge}{\mathbin}{\M@extsymbols@font}{"2227}
2595  \DeclareMathSymbol{\vee}{\mathbin}{\M@extsymbols@font}{"2228}
2596  \DeclareMathSymbol{\cap}{\mathord}{\M@extsymbols@font}{"2229}
2597  \DeclareMathSymbol{\cup}{\mathbin}{\M@extsymbols@font}{"222A}
2598  \DeclareMathSymbol{\sqcap}{\mathbin}{\M@extsymbols@font}{"2293}
2599  \DeclareMathSymbol{\sqcup}{\mathbin}{\M@extsymbols@font}{"2294}
2600  \DeclareMathSymbol{\amalg}{\mathbin}{\M@extsymbols@font}{"2A3F}
2601  \DeclareMathSymbol{\wr}{\mathbin}{\M@extsymbols@font}{"2240}
2602  \DeclareMathSymbol{\ast}{\mathbin}{\M@extsymbols@font}{"2217}
2603  \DeclareMathSymbol{\star}{\mathbin}{\M@extsymbols@font}{"22C6}
2604  \DeclareMathSymbol{\diamond}{\mathbin}{\M@extsymbols@font}{"22C4}
2605  \DeclareMathSymbol{\varcdot}{\mathbin}{\M@extsymbols@font}{"22C5}
2606  \DeclareMathSymbol{\varsetminus}{\mathbin}{\M@extsymbols@font}{"2216}
2607  \DeclareMathSymbol{\oplus}{\mathbin}{\M@extsymbols@font}{"2295}
2608  \DeclareMathSymbol{\otimes}{\mathbin}{\M@extsymbols@font}{"2297}
2609  \DeclareMathSymbol{\ominus}{\mathbin}{\M@extsymbols@font}{"2296}
2610  \DeclareMathSymbol{\odiv}{\mathbin}{\M@extsymbols@font}{"2A38}
2611  \DeclareMathSymbol{\oslash}{\mathbin}{\M@extsymbols@font}{"2298}
2612  \DeclareMathSymbol{\odot}{\mathbin}{\M@extsymbols@font}{"2299}
2613  \DeclareMathSymbol{\sqplus}{\mathbin}{\M@extsymbols@font}{"229E}
2614  \DeclareMathSymbol{\sqtimes}{\mathbin}{\M@extsymbols@font}{"22A0}
2615  \DeclareMathSymbol{\sqminus}{\mathbin}{\M@extsymbols@font}{"229F}
2616  \DeclareMathSymbol{\sqdot}{\mathbin}{\M@extsymbols@font}{"22A1}
2617  \DeclareMathSymbol{\in}{\mathrel}{\M@extsymbols@font}{"2208}
2618  \DeclareMathSymbol{\ni}{\mathrel}{\M@extsymbols@font}{"220B}
2619  \DeclareMathSymbol{\subset}{\mathrel}{\M@extsymbols@font}{"2282}
2620  \DeclareMathSymbol{\supset}{\mathrel}{\M@extsymbols@font}{"2283}
2621  \DeclareMathSymbol{\subseteq}{\mathrel}{\M@extsymbols@font}{"2286}
2622  \DeclareMathSymbol{\supseteq}{\mathrel}{\M@extsymbols@font}{"2287}
2623  \DeclareMathSymbol{\sqsubset}{\mathrel}{\M@extsymbols@font}{"228F}
2624  \DeclareMathSymbol{\sqsupset}{\mathrel}{\M@extsymbols@font}{"2290}
2625  \DeclareMathSymbol{\sqsubseteq}{\mathrel}{\M@extsymbols@font}{"2291}
2626  \DeclareMathSymbol{\sqsupseteq}{\mathrel}{\M@extsymbols@font}{"2292}
2627  \DeclareMathSymbol{\triangleleft}{\mathrel}{\M@extsymbols@font}{"22B2}
2628  \DeclareMathSymbol{\triangleright}{\mathrel}{\M@extsymbols@font}{"22B3}
2629  \DeclareMathSymbol{\trianglelefteq}{\mathrel}{\M@extsymbols@font}{"22B4}
2630  \DeclareMathSymbol{\trianglerighteq}{\mathrel}{\M@extsymbols@font}{"22B5}
2631  \DeclareMathSymbol{\propto}{\mathrel}{\M@extsymbols@font}{"221D}
2632  \DeclareMathSymbol{\bowtie}{\mathrel}{\M@extsymbols@font}{"22C8}
2633  \DeclareMathSymbol{\hourglass}{\mathrel}{\M@extsymbols@font}{"29D6}
2634  \DeclareMathSymbol{\therefore}{\mathrel}{\M@extsymbols@font}{"2234}
2635  \DeclareMathSymbol{\because}{\mathrel}{\M@extsymbols@font}{"2235}
```

```
2636    \DeclareMathSymbol{\ratio}{\mathrel}{\M@extsymbols@font}{"2236}
2637    \DeclareMathSymbol{\proportion}{\mathrel}{\M@extsymbols@font}{"2237}
2638    \DeclareMathSymbol{\ll}{\mathrel}{\M@extsymbols@font}{"226A}
2639    \DeclareMathSymbol{\gg}{\mathrel}{\M@extsymbols@font}{"226B}
2640    \DeclareMathSymbol{\lll}{\mathrel}{\M@extsymbols@font}{"22D8}
2641    \DeclareMathSymbol{\ggg}{\mathrel}{\M@extsymbols@font}{"22D9}
2642    \DeclareMathSymbol{\leqq}{\mathrel}{\M@extsymbols@font}{"2266}
2643    \DeclareMathSymbol{\geqq}{\mathrel}{\M@extsymbols@font}{"2267}
2644    \DeclareMathSymbol{\lapprox}{\mathrel}{\M@extsymbols@font}{"2A85}
2645    \DeclareMathSymbol{\gapprox}{\mathrel}{\M@extsymbols@font}{"2A86}
2646    \DeclareMathSymbol{\simeq}{\mathrel}{\M@extsymbols@font}{"2243}
2647    \DeclareMathSymbol{\eqsim}{\mathrel}{\M@extsymbols@font}{"2242}
2648    \DeclareMathSymbol{\simeqq}{\mathrel}{\M@extsymbols@font}{"2245}
2649      \global\let\cong\simeqq
2650    \DeclareMathSymbol{\approxeq}{\mathrel}{\M@extsymbols@font}{"224A}
2651    \DeclareMathSymbol{\sssim}{\mathrel}{\M@extsymbols@font}{"224B}
2652    \DeclareMathSymbol{\seq}{\mathrel}{\M@extsymbols@font}{"224C}
2653    \DeclareMathSymbol{\doteq}{\mathrel}{\M@extsymbols@font}{"2250}
2654    \DeclareMathSymbol{\coloneq}{\mathrel}{\M@extsymbols@font}{"2254}
2655    \DeclareMathSymbol{\eqcolon}{\mathrel}{\M@extsymbols@font}{"2255}
2656    \DeclareMathSymbol{\ringeq}{\mathrel}{\M@extsymbols@font}{"2257}
2657    \DeclareMathSymbol{\arceq}{\mathrel}{\M@extsymbols@font}{"2258}
2658    \DeclareMathSymbol{\wedgeeq}{\mathrel}{\M@extsymbols@font}{"2259}
2659    \DeclareMathSymbol{\veeeq}{\mathrel}{\M@extsymbols@font}{"225A}
2660    \DeclareMathSymbol{\stareq}{\mathrel}{\M@extsymbols@font}{"225B}
2661    \DeclareMathSymbol{\triangleeq}{\mathrel}{\M@extsymbols@font}{"225C}
2662    \DeclareMathSymbol{\defeq}{\mathrel}{\M@extsymbols@font}{"225D}
2663    \DeclareMathSymbol{\qeq}{\mathrel}{\M@extsymbols@font}{"225F}
2664    \DeclareMathSymbol{\lsim}{\mathrel}{\M@extsymbols@font}{"2272}
2665    \DeclareMathSymbol{\gsim}{\mathrel}{\M@extsymbols@font}{"2273}
2666    \DeclareMathSymbol{\prec}{\mathrel}{\M@extsymbols@font}{"227A}
2667    \DeclareMathSymbol{\succ}{\mathrel}{\M@extsymbols@font}{"227B}
2668    \DeclareMathSymbol{\preceq}{\mathrel}{\M@extsymbols@font}{"227C}
2669    \DeclareMathSymbol{\succeq}{\mathrel}{\M@extsymbols@font}{"227D}
2670    \DeclareMathSymbol{\preceqq}{\mathrel}{\M@extsymbols@font}{"2AB3}
2671    \DeclareMathSymbol{\succeqq}{\mathrel}{\M@extsymbols@font}{"2AB4}
2672    \DeclareMathSymbol{\precsim}{\mathrel}{\M@extsymbols@font}{"227E}
2673    \DeclareMathSymbol{\succsim}{\mathrel}{\M@extsymbols@font}{"227F}
2674    \DeclareMathSymbol{\precapprox}{\mathrel}{\M@extsymbols@font}{"2AB7}
2675    \DeclareMathSymbol{\succapprox}{\mathrel}{\M@extsymbols@font}{"2AB8}
2676    \DeclareMathSymbol{\precprec}{\mathrel}{\M@extsymbols@font}{"2ABB}
2677    \DeclareMathSymbol{\succsucc}{\mathrel}{\M@extsymbols@font}{"2ABC}
2678    \DeclareMathSymbol{\asymp}{\mathrel}{\M@extsymbols@font}{"224D}
2679    \DeclareMathSymbol{\nin}{\mathrel}{\M@extsymbols@font}{"2209}
2680    \DeclareMathSymbol{\nni}{\mathrel}{\M@extsymbols@font}{"220C}
2681    \DeclareMathSymbol{\nsubset}{\mathrel}{\M@extsymbols@font}{"2284}
2682    \DeclareMathSymbol{\nsupset}{\mathrel}{\M@extsymbols@font}{"2285}
```

```
2683    \DeclareMathSymbol{\nsubseteq}{\mathrel}{\M@extsymbols@font}{"2288}
2684    \DeclareMathSymbol{\nsupseteq}{\mathrel}{\M@extsymbols@font}{"2289}
2685    \DeclareMathSymbol{\subsetneq}{\mathrel}{\M@extsymbols@font}{"228A}
2686    \DeclareMathSymbol{\supsetneq}{\mathrel}{\M@extsymbols@font}{"228B}
2687    \DeclareMathSymbol{\nsqsubseteq}{\mathrel}{\M@extsymbols@font}{"22E2}
2688    \DeclareMathSymbol{\nsqsupseteq}{\mathrel}{\M@extsymbols@font}{"22E3}
2689    \DeclareMathSymbol{\sqsubsetneq}{\mathrel}{\M@extsymbols@font}{"22E4}
2690    \DeclareMathSymbol{\sqsupsetneq}{\mathrel}{\M@extsymbols@font}{"22E5}
2691    \DeclareMathSymbol{\neq}{\mathrel}{\M@extsymbols@font}{"2260}
2692    \DeclareMathSymbol{\nl}{\mathrel}{\M@extsymbols@font}{"226E}
2693    \DeclareMathSymbol{\ng}{\mathrel}{\M@extsymbols@font}{"226F}
2694    \DeclareMathSymbol{\nleq}{\mathrel}{\M@extsymbols@font}{"2270}
2695    \DeclareMathSymbol{\ngeq}{\mathrel}{\M@extsymbols@font}{"2271}
2696    \DeclareMathSymbol{\lneq}{\mathrel}{\M@extsymbols@font}{"2A87}
2697    \DeclareMathSymbol{\gneq}{\mathrel}{\M@extsymbols@font}{"2A88}
2698    \DeclareMathSymbol{\lneqq}{\mathrel}{\M@extsymbols@font}{"2268}
2699    \DeclareMathSymbol{\gneqq}{\mathrel}{\M@extsymbols@font}{"2269}
2700    \DeclareMathSymbol{\ntriangleleft}{\mathrel}{\M@extsymbols@font}{"22EA}
2701    \DeclareMathSymbol{\ntriangleright}{\mathrel}{\M@extsymbols@font}{"22EB}
2702    \DeclareMathSymbol{\ntrianglelefteq}{\mathrel}{\M@extsymbols@font}{"22EC}
2703    \DeclareMathSymbol{\ntrianglerighteq}{\mathrel}{\M@extsymbols@font}{"22ED}
2704    \DeclareMathSymbol{\nsim}{\mathrel}{\M@extsymbols@font}{"2241}
2705    \DeclareMathSymbol{\napprox}{\mathrel}{\M@extsymbols@font}{"2249}
2706    \DeclareMathSymbol{\nsimeq}{\mathrel}{\M@extsymbols@font}{"2244}
2707    \DeclareMathSymbol{\nsimeqq}{\mathrel}{\M@extsymbols@font}{"2247}
2708    \DeclareMathSymbol{\simneqq}{\mathrel}{\M@extsymbols@font}{"2246}
2709    \DeclareMathSymbol{\nlsim}{\mathrel}{\M@extsymbols@font}{"2274}
2710    \DeclareMathSymbol{\ngsim}{\mathrel}{\M@extsymbols@font}{"2275}
2711    \DeclareMathSymbol{\lnsim}{\mathrel}{\M@extsymbols@font}{"22E6}
2712    \DeclareMathSymbol{\gnsim}{\mathrel}{\M@extsymbols@font}{"22E7}
2713    \DeclareMathSymbol{\lnapprox}{\mathrel}{\M@extsymbols@font}{"2A89}
2714    \DeclareMathSymbol{\gnapprox}{\mathrel}{\M@extsymbols@font}{"2A8A}
2715    \DeclareMathSymbol{\nprec}{\mathrel}{\M@extsymbols@font}{"2280}
2716    \DeclareMathSymbol{\nsucc}{\mathrel}{\M@extsymbols@font}{"2281}
2717    \DeclareMathSymbol{\npreceq}{\mathrel}{\M@extsymbols@font}{"22E0}
2718    \DeclareMathSymbol{\nsucceq}{\mathrel}{\M@extsymbols@font}{"22E1}
2719    \DeclareMathSymbol{\precneq}{\mathrel}{\M@extsymbols@font}{"2AB1}
2720    \DeclareMathSymbol{\succneq}{\mathrel}{\M@extsymbols@font}{"2AB2}
2721    \DeclareMathSymbol{\precneqq}{\mathrel}{\M@extsymbols@font}{"2AB5}
2722    \DeclareMathSymbol{\succneqq}{\mathrel}{\M@extsymbols@font}{"2AB6}
2723    \DeclareMathSymbol{\precnsim}{\mathrel}{\M@extsymbols@font}{"22E8}
2724    \DeclareMathSymbol{\succnsim}{\mathrel}{\M@extsymbols@font}{"22E9}
2725    \DeclareMathSymbol{\precnapprox}{\mathrel}{\M@extsymbols@font}{"2AB9}
2726    \DeclareMathSymbol{\succnapprox}{\mathrel}{\M@extsymbols@font}{"2ABA}
2727    \DeclareMathSymbol{\nequiv}{\mathrel}{\M@extsymbols@font}{"2262}
```

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

```
2728    \ifM@adjust@font\else
```

```
2729    \DeclareMathSymbol{\nabla}{\mathord}{\M@extsymbols@font}{"2207}
2730    \fi}
```
Set arrows.
```
2731 \def\M@arrows@set{%
2732    \edef\M@arrows@font{M\M@arrowsshape\@tempa}
2733    \let\uparrow\@undefined
2734    \let\Uparrow\@undefined
2735    \let\downarrow\@undefined
2736    \let\Downarrow\@undefined
2737    \let\updownarrow\@undefined
2738    \let\Updownarrow\@undefined
2739    \let\longrightarrow\@undefined
2740    \let\longleftarrow\@undefined
2741    \let\longleftrightarrow\@undefined
2742    \let\hookrightarrow\@undefined
2743    \let\hookleftarrow\@undefined
2744    \let\Longrightarrow\@undefined
2745    \let\Longleftarrow\@undefined
2746    \let\Longleftrightarrow\@undefined
2747    \let\rightleftharpoons\@undefined
2748    \DeclareMathSymbol{\rightarrow}{\mathrel}{\M@arrows@font}{"2192}
2749      \global\let\to\rightarrow
2750    \DeclareMathSymbol{\nrightarrow}{\mathrel}{\M@arrows@font}{"219B}
2751    \DeclareMathSymbol{\Rightarrow}{\mathrel}{\M@arrows@font}{"21D2}
2752    \DeclareMathSymbol{\nRightarrow}{\mathrel}{\M@arrows@font}{"21CF}
2753    \DeclareMathSymbol{\Rrightarrow}{\mathrel}{\M@arrows@font}{"21DB}
2754    \DeclareMathSymbol{\longrightarrow}{\mathrel}{\M@arrows@font}{"27F6}
2755    \DeclareMathSymbol{\Longrightarrow}{\mathrel}{\M@arrows@font}{"27F9}
2756    \DeclareMathSymbol{\rightbararrow}{\mathrel}{\M@arrows@font}{"21A6}
2757      \global\let\mapsto\rightbararrow
2758    \DeclareMathSymbol{\Rightbararrow}{\mathrel}{\M@arrows@font}{"2907}
2759    \DeclareMathSymbol{\longrightbararrow}{\mathrel}{\M@arrows@font}{"27FC}
2760      \global\let\longmapsto\longrightbararrow
2761    \DeclareMathSymbol{\Longrightbararrow}{\mathrel}{\M@arrows@font}{"27FE}
2762    \DeclareMathSymbol{\hookrightarrow}{\mathrel}{\M@arrows@font}{"21AA}
2763    \DeclareMathSymbol{\rightdasharrow}{\mathrel}{\M@arrows@font}{"21E2}
2764    \DeclareMathSymbol{\rightharpoonup}{\mathrel}{\M@arrows@font}{"21C0}
2765    \DeclareMathSymbol{\rightharpoondown}{\mathrel}{\M@arrows@font}{"21C1}
2766    \DeclareMathSymbol{\rightarrowtail}{\mathrel}{\M@arrows@font}{"21A3}
2767    \DeclareMathSymbol{\rightoplusarrow}{\mathrel}{\M@arrows@font}{"27F4}
2768    \DeclareMathSymbol{\rightwavearrow}{\mathrel}{\M@arrows@font}{"219D}
2769    \DeclareMathSymbol{\rightsquigarrow}{\mathrel}{\M@arrows@font}{"21DD}
2770    \DeclareMathSymbol{\longrightsquigarrow}{\mathrel}{\M@arrows@font}{"27FF}
2771    \DeclareMathSymbol{\looparrowright}{\mathrel}{\M@arrows@font}{"21AC}
2772    \DeclareMathSymbol{\curvearrowright}{\mathrel}{\M@arrows@font}{"293B}
2773    \DeclareMathSymbol{\circlearrowright}{\mathrel}{\M@arrows@font}{"21BB}
2774    \DeclareMathSymbol{\twoheadrightarrow}{\mathrel}{\M@arrows@font}{"21A0}
```

```
2775   \DeclareMathSymbol{\rightarrowtobar}{\mathrel}{\M@arrows@font}{"21E5}
2776   \DeclareMathSymbol{\rightwhitearrow}{\mathrel}{\M@arrows@font}{"21E8}
2777   \DeclareMathSymbol{\rightrightarrows}{\mathrel}{\M@arrows@font}{"21C9}
2778   \DeclareMathSymbol{\rightrightrightarrows}{\mathrel}{\M@arrows@font}{"21F6}
2779   \DeclareMathSymbol{\leftarrow}{\mathrel}{\M@arrows@font}{"2190}
2780     \global\let\from\leftarrow
2781   \DeclareMathSymbol{\nleftarrow}{\mathrel}{\M@arrows@font}{"219A}
2782   \DeclareMathSymbol{\Leftarrow}{\mathrel}{\M@arrows@font}{"21D0}
2783   \DeclareMathSymbol{\nLeftarrow}{\mathrel}{\M@arrows@font}{"21CD}
2784   \DeclareMathSymbol{\Lleftarrow}{\mathrel}{\M@arrows@font}{"21DA}
2785   \DeclareMathSymbol{\longleftarrow}{\mathrel}{\M@arrows@font}{"27F5}
2786   \DeclareMathSymbol{\Longleftarrow}{\mathrel}{\M@arrows@font}{"27F8}
2787   \DeclareMathSymbol{\leftbararrow}{\mathrel}{\M@arrows@font}{"21A4}
2788     \global\let\mapsfrom\leftbararrow
2789   \DeclareMathSymbol{\Leftbararrow}{\mathrel}{\M@arrows@font}{"2906}
2790   \DeclareMathSymbol{\longleftbararrow}{\mathrel}{\M@arrows@font}{"27FB}
2791     \global\let\longmapsfrom\longleftbararrow
2792   \DeclareMathSymbol{\Longleftbararrow}{\mathrel}{\M@arrows@font}{"27FD}
2793   \DeclareMathSymbol{\hookleftarrow}{\mathrel}{\M@arrows@font}{"21A9}
2794   \DeclareMathSymbol{\leftdasharrow}{\mathrel}{\M@arrows@font}{"21E0}
2795   \DeclareMathSymbol{\leftharpoonup}{\mathrel}{\M@arrows@font}{"21BC}
2796   \DeclareMathSymbol{\leftharpoondown}{\mathrel}{\M@arrows@font}{"21BD}
2797   \DeclareMathSymbol{\leftarrowtail}{\mathrel}{\M@arrows@font}{"21A2}
2798   \DeclareMathSymbol{\leftoplusarrow}{\mathrel}{\M@arrows@font}{"2B32}
2799   \DeclareMathSymbol{\leftwavearrow}{\mathrel}{\M@arrows@font}{"219C}
2800   \DeclareMathSymbol{\leftsquigarrow}{\mathrel}{\M@arrows@font}{"21DC}
2801   \DeclareMathSymbol{\longleftsquigarrow}{\mathrel}{\M@arrows@font}{"2B33}
2802   \DeclareMathSymbol{\looparrowleft}{\mathrel}{\M@arrows@font}{"21AB}
2803   \DeclareMathSymbol{\curvearrowleft}{\mathrel}{\M@arrows@font}{"293A}
2804   \DeclareMathSymbol{\circlearrowleft}{\mathrel}{\M@arrows@font}{"21BA}
2805   \DeclareMathSymbol{\twoheadleftarrow}{\mathrel}{\M@arrows@font}{"219E}
2806   \DeclareMathSymbol{\leftarrowtobar}{\mathrel}{\M@arrows@font}{"21E4}
2807   \DeclareMathSymbol{\leftwhitearrow}{\mathrel}{\M@arrows@font}{"21E6}
2808   \DeclareMathSymbol{\leftleftarrows}{\mathrel}{\M@arrows@font}{"21C7}
2809   \DeclareMathSymbol{\leftleftleftarrows}{\mathrel}{\M@arrows@font}{"2B31}
2810   \DeclareMathSymbol{\leftrightarrow}{\mathrel}{\M@arrows@font}{"2194}
2811   \DeclareMathSymbol{\Leftrightarrow}{\mathrel}{\M@arrows@font}{"21D4}
2812   \DeclareMathSymbol{\nLeftrightarrow}{\mathrel}{\M@arrows@font}{"21CE}
2813   \DeclareMathSymbol{\longleftrightarrow}{\mathrel}{\M@arrows@font}{"27F7}
2814   \DeclareMathSymbol{\Longleftrightarrow}{\mathrel}{\M@arrows@font}{"27FA}
2815   \DeclareMathSymbol{\leftrightwavearrow}{\mathrel}{\M@arrows@font}{"21AD}
2816   \DeclareMathSymbol{\leftrightarrows}{\mathrel}{\M@arrows@font}{"21C6}
2817   \DeclareMathSymbol{\leftrightharpoons}{\mathrel}{\M@arrows@font}{"21CB}
2818   \DeclareMathSymbol{\leftrightarrowstobar}{\mathrel}{\M@arrows@font}{"21B9}
2819   \DeclareMathSymbol{\rightleftarrows}{\mathrel}{\M@arrows@font}{"21C4}
2820   \DeclareMathSymbol{\rightleftharpoons}{\mathrel}{\M@arrows@font}{"21CC}
2821   \DeclareMathSymbol{\uparrow}{\mathrel}{\M@arrows@font}{"2191}
```

```
2822  \DeclareMathSymbol{\Uparrow}{\mathrel}{\M@arrows@font}{"21D1}
2823  \DeclareMathSymbol{\Uuparrow}{\mathrel}{\M@arrows@font}{"290A}
2824  \DeclareMathSymbol{\upbararrow}{\mathrel}{\M@arrows@font}{"21A5}
2825  \DeclareMathSymbol{\updasharrow}{\mathrel}{\M@arrows@font}{"21E1}
2826  \DeclareMathSymbol{\upharpoonleft}{\mathrel}{\M@arrows@font}{"21BF}
2827  \DeclareMathSymbol{\upharpoonright}{\mathrel}{\M@arrows@font}{"21BE}
2828  \DeclareMathSymbol{\twoheaduparrow}{\mathrel}{\M@arrows@font}{"219F}
2829  \DeclareMathSymbol{\uparrowtobar}{\mathrel}{\M@arrows@font}{"2912}
2830  \DeclareMathSymbol{\upwhitearrow}{\mathrel}{\M@arrows@font}{"21E7}
2831  \DeclareMathSymbol{\upwhitebararrow}{\mathrel}{\M@arrows@font}{"21EA}
2832  \DeclareMathSymbol{\upuparrows}{\mathrel}{\M@arrows@font}{"21C8}
2833  \DeclareMathSymbol{\downarrow}{\mathrel}{\M@arrows@font}{"2193}
2834  \DeclareMathSymbol{\Downarrow}{\mathrel}{\M@arrows@font}{"21D3}
2835  \DeclareMathSymbol{\Ddownarrow}{\mathrel}{\M@arrows@font}{"290B}
2836  \DeclareMathSymbol{\downbararrow}{\mathrel}{\M@arrows@font}{"21A7}
2837  \DeclareMathSymbol{\downdasharrow}{\mathrel}{\M@arrows@font}{"21E3}
2838  \DeclareMathSymbol{\zigzagarrow}{\mathrel}{\M@arrows@font}{"21AF}
2839    \global\let\lightningboltarrow\zigzagarrow
2840  \DeclareMathSymbol{\downharpoonleft}{\mathrel}{\M@arrows@font}{"21C3}
2841  \DeclareMathSymbol{\downharpoonright}{\mathrel}{\M@arrows@font}{"21C2}
2842  \DeclareMathSymbol{\twoheaddownarrow}{\mathrel}{\M@arrows@font}{"21A1}
2843  \DeclareMathSymbol{\downarrowtobar}{\mathrel}{\M@arrows@font}{"2913}
2844  \DeclareMathSymbol{\downwhitearrow}{\mathrel}{\M@arrows@font}{"21E9}
2845  \DeclareMathSymbol{\downdownarrows}{\mathrel}{\M@arrows@font}{"21CA}
2846  \DeclareMathSymbol{\updownarrow}{\mathrel}{\M@arrows@font}{"2195}
2847  \DeclareMathSymbol{\Updownarrow}{\mathrel}{\M@arrows@font}{"21D5}
2848  \DeclareMathSymbol{\updownarrows}{\mathrel}{\M@arrows@font}{"21C5}
2849  \DeclareMathSymbol{\downuparrows}{\mathrel}{\M@arrows@font}{"21F5}
2850  \DeclareMathSymbol{\updownharpoons}{\mathrel}{\M@arrows@font}{"296E}
2851  \DeclareMathSymbol{\downupharpoons}{\mathrel}{\M@arrows@font}{"296F}
2852  \DeclareMathSymbol{\nearrow}{\mathrel}{\M@arrows@font}{"2197}
2853  \DeclareMathSymbol{\Nearrow}{\mathrel}{\M@arrows@font}{"21D7}
2854  \DeclareMathSymbol{\nwarrow}{\mathrel}{\M@arrows@font}{"2196}
2855  \DeclareMathSymbol{\Nwarrow}{\mathrel}{\M@arrows@font}{"21D6}
2856  \DeclareMathSymbol{\searrow}{\mathrel}{\M@arrows@font}{"2198}
2857  \DeclareMathSymbol{\Searrow}{\mathrel}{\M@arrows@font}{"21D8}
2858  \DeclareMathSymbol{\swarrow}{\mathrel}{\M@arrows@font}{"2199}
2859  \DeclareMathSymbol{\Swarrow}{\mathrel}{\M@arrows@font}{"21D9}
2860  \DeclareMathSymbol{\nwsearrow}{\mathrel}{\M@arrows@font}{"2921}
2861  \DeclareMathSymbol{\neswarrow}{\mathrel}{\M@arrows@font}{"2922}
2862  \DeclareMathSymbol{\lcirclearrow}{\mathrel}{\M@arrows@font}{"27F2}
2863  \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 sumbol 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.

```
2864 \def\M@bb@set{%
2865   \protected\def\mathbb##1{\relax
2866     \ifmmode\else
2867       \M@HModeError\mathbb
2868       $%
2869     \fi
2870     \begingroup
2871       \M@bb@mathcodes
2872       ##1%
2873     \endgroup}
2874   \edef\M@bb@num{\number\csname symM\M@bbshape\@tempa\endcsname}
2875   \protected\edef\M@bb@mathcodes{%
2876   \Umathcode`A=0+\M@bb@num"1D538\relax
2877   \Umathcode`B=0+\M@bb@num"1D539\relax
2878   \Umathcode`C=0+\M@bb@num"2102\relax
2879   \Umathcode`D=0+\M@bb@num"1D53B\relax
2880   \Umathcode`E=0+\M@bb@num"1D53C\relax
2881   \Umathcode`F=0+\M@bb@num"1D53D\relax
2882   \Umathcode`G=0+\M@bb@num"1D53E\relax
2883   \Umathcode`H=0+\M@bb@num"210D\relax
2884   \Umathcode`I=0+\M@bb@num"1D540\relax
2885   \Umathcode`J=0+\M@bb@num"1D541\relax
2886   \Umathcode`K=0+\M@bb@num"1D542\relax
2887   \Umathcode`L=0+\M@bb@num"1D543\relax
2888   \Umathcode`M=0+\M@bb@num"1D544\relax
2889   \Umathcode`N=0+\M@bb@num"2115\relax
2890   \Umathcode`O=0+\M@bb@num"1D546\relax
2891   \Umathcode`P=0+\M@bb@num"2119\relax
2892   \Umathcode`Q=0+\M@bb@num"211A\relax
2893   \Umathcode`R=0+\M@bb@num"211D\relax
2894   \Umathcode`S=0+\M@bb@num"1D54A\relax
2895   \Umathcode`T=0+\M@bb@num"1D54B\relax
2896   \Umathcode`U=0+\M@bb@num"1D54C\relax
2897   \Umathcode`V=0+\M@bb@num"1D54D\relax
2898   \Umathcode`W=0+\M@bb@num"1D54E\relax
2899   \Umathcode`X=0+\M@bb@num"1D54F\relax
2900   \Umathcode`Y=0+\M@bb@num"1D550\relax
2901   \Umathcode`Z=0+\M@bb@num"2124\relax
2902   \Umathcode`a=0+\M@bb@num"1D552\relax
2903   \Umathcode`b=0+\M@bb@num"1D553\relax
2904   \Umathcode`c=0+\M@bb@num"1D554\relax
2905   \Umathcode`d=0+\M@bb@num"1D555\relax
2906   \Umathcode`e=0+\M@bb@num"1D556\relax
2907   \Umathcode`f=0+\M@bb@num"1D557\relax
2908   \Umathcode`g=0+\M@bb@num"1D558\relax
```

```
2909   \Umathcode`h=0+\M@bb@num"1D559\relax
2910   \Umathcode`i=0+\M@bb@num"1D55A\relax
2911   \Umathcode`j=0+\M@bb@num"1D55B\relax
2912   \Umathcode`k=0+\M@bb@num"1D55C\relax
2913   \Umathcode`l=0+\M@bb@num"1D55D\relax
2914   \Umathcode`m=0+\M@bb@num"1D55E\relax
2915   \Umathcode`n=0+\M@bb@num"1D55F\relax
2916   \Umathcode`o=0+\M@bb@num"1D560\relax
2917   \Umathcode`p=0+\M@bb@num"1D561\relax
2918   \Umathcode`q=0+\M@bb@num"1D562\relax
2919   \Umathcode`r=0+\M@bb@num"1D563\relax
2920   \Umathcode`s=0+\M@bb@num"1D564\relax
2921   \Umathcode`t=0+\M@bb@num"1D565\relax
2922   \Umathcode`u=0+\M@bb@num"1D566\relax
2923   \Umathcode`v=0+\M@bb@num"1D567\relax
2924   \Umathcode`w=0+\M@bb@num"1D568\relax
2925   \Umathcode`x=0+\M@bb@num"1D569\relax
2926   \Umathcode`y=0+\M@bb@num"1D56A\relax
2927   \Umathcode`z=0+\M@bb@num"1D56B\relax
2928   \Umathcode`0=0+\M@bb@num"1D7D8\relax
2929   \Umathcode`1=0+\M@bb@num"1D7D9\relax
2930   \Umathcode`2=0+\M@bb@num"1D7DA\relax
2931   \Umathcode`3=0+\M@bb@num"1D7DB\relax
2932   \Umathcode`4=0+\M@bb@num"1D7DC\relax
2933   \Umathcode`5=0+\M@bb@num"1D7DD\relax
2934   \Umathcode`6=0+\M@bb@num"1D7DE\relax
2935   \Umathcode`7=0+\M@bb@num"1D7DF\relax
2936   \Umathcode`8=0+\M@bb@num"1D7E0\relax
2937   \Umathcode`9=0+\M@bb@num"1D7E1\relax}}
```

Set caligraphic letters.

```
2938 \def\M@cal@set{%
2939   \protected\def\mathcal##1{\relax
2940     \ifmmode\else
2941       \M@HModeError\mathcal
2942       $%
2943     \fi
2944     \begingroup
2945       \M@cal@mathcodes
2946       ##1%
2947     \endgroup}
2948   \edef\M@cal@num{\number\csname symM\M@calshape\@tempa\endcsname}
2949   \protected\edef\M@cal@mathcodes{%
2950   \Umathcode`A=0+\M@cal@num"1D49C\relax
2951   \Umathcode`B=0+\M@cal@num"212C\relax
2952   \Umathcode`C=0+\M@cal@num"1D49E\relax
2953   \Umathcode`D=0+\M@cal@num"1D49F\relax
2954   \Umathcode`E=0+\M@cal@num"2130\relax
```

```
2955   \Umathcode`F=0+\M@cal@num"2131\relax
2956   \Umathcode`G=0+\M@cal@num"1D4A2\relax
2957   \Umathcode`H=0+\M@cal@num"210B\relax
2958   \Umathcode`I=0+\M@cal@num"2110\relax
2959   \Umathcode`J=0+\M@cal@num"1D4A5\relax
2960   \Umathcode`K=0+\M@cal@num"1D4A6\relax
2961   \Umathcode`L=0+\M@cal@num"2112\relax
2962   \Umathcode`M=0+\M@cal@num"2133\relax
2963   \Umathcode`N=0+\M@cal@num"1D4A9\relax
2964   \Umathcode`O=0+\M@cal@num"1D4AA\relax
2965   \Umathcode`P=0+\M@cal@num"1D4AB\relax
2966   \Umathcode`Q=0+\M@cal@num"1D4AC\relax
2967   \Umathcode`R=0+\M@cal@num"211B\relax
2968   \Umathcode`S=0+\M@cal@num"1D4AE\relax
2969   \Umathcode`T=0+\M@cal@num"1D4AF\relax
2970   \Umathcode`U=0+\M@cal@num"1D4B0\relax
2971   \Umathcode`V=0+\M@cal@num"1D4B1\relax
2972   \Umathcode`W=0+\M@cal@num"1D4B2\relax
2973   \Umathcode`X=0+\M@cal@num"1D4B3\relax
2974   \Umathcode`Y=0+\M@cal@num"1D4B4\relax
2975   \Umathcode`Z=0+\M@cal@num"1D4B5\relax
2976   \Umathcode`a=0+\M@cal@num"1D4B6\relax
2977   \Umathcode`b=0+\M@cal@num"1D4B7\relax
2978   \Umathcode`c=0+\M@cal@num"1D4B8\relax
2979   \Umathcode`d=0+\M@cal@num"1D4B9\relax
2980   \Umathcode`e=0+\M@cal@num"212F\relax
2981   \Umathcode`f=0+\M@cal@num"1D4BB\relax
2982   \Umathcode`g=0+\M@cal@num"210A\relax
2983   \Umathcode`h=0+\M@cal@num"1D4BD\relax
2984   \Umathcode`i=0+\M@cal@num"1D4BE\relax
2985   \Umathcode`j=0+\M@cal@num"1D4BF\relax
2986   \Umathcode`k=0+\M@cal@num"1D4C0\relax
2987   \Umathcode`l=0+\M@cal@num"1D4C1\relax
2988   \Umathcode`m=0+\M@cal@num"1D4C2\relax
2989   \Umathcode`n=0+\M@cal@num"1D4C3\relax
2990   \Umathcode`o=0+\M@cal@num"2134\relax
2991   \Umathcode`p=0+\M@cal@num"1D4C5\relax
2992   \Umathcode`q=0+\M@cal@num"1D4C6\relax
2993   \Umathcode`r=0+\M@cal@num"1D4C7\relax
2994   \Umathcode`s=0+\M@cal@num"1D4C8\relax
2995   \Umathcode`t=0+\M@cal@num"1D4C9\relax
2996   \Umathcode`u=0+\M@cal@num"1D4CA\relax
2997   \Umathcode`v=0+\M@cal@num"1D4CB\relax
2998   \Umathcode`w=0+\M@cal@num"1D4CC\relax
2999   \Umathcode`x=0+\M@cal@num"1D4CD\relax
3000   \Umathcode`y=0+\M@cal@num"1D4CE\relax
3001   \Umathcode`z=0+\M@cal@num"1D4CF\relax}}
```

Set fraktur letters.

```
3002 \def\M@frak@set{%
3003   \protected\def\mathfrak##1{\relax
3004     \ifmmode\else
3005       \M@HModeError\mathfrak
3006       $%
3007     \fi
3008     \begingroup
3009       \M@frak@mathcodes
3010       ##1%
3011     \endgroup}
3012   \edef\M@frak@num{\number\csname symM\M@frakshape\@tempa\endcsname}
3013   \protected\edef\M@frak@mathcodes{%
3014   \Umathcode`A=0+\M@frak@num"1D504\relax
3015   \Umathcode`B=0+\M@frak@num"1D505\relax
3016   \Umathcode`C=0+\M@frak@num"212D\relax
3017   \Umathcode`D=0+\M@frak@num"1D507\relax
3018   \Umathcode`E=0+\M@frak@num"1D508\relax
3019   \Umathcode`F=0+\M@frak@num"1D509\relax
3020   \Umathcode`G=0+\M@frak@num"1D50A\relax
3021   \Umathcode`H=0+\M@frak@num"210C\relax
3022   \Umathcode`I=0+\M@frak@num"2111\relax
3023   \Umathcode`J=0+\M@frak@num"1D50D\relax
3024   \Umathcode`K=0+\M@frak@num"1D50E\relax
3025   \Umathcode`L=0+\M@frak@num"1D50F\relax
3026   \Umathcode`M=0+\M@frak@num"1D510\relax
3027   \Umathcode`N=0+\M@frak@num"1D511\relax
3028   \Umathcode`O=0+\M@frak@num"1D512\relax
3029   \Umathcode`P=0+\M@frak@num"1D513\relax
3030   \Umathcode`Q=0+\M@frak@num"1D514\relax
3031   \Umathcode`R=0+\M@frak@num"211C\relax
3032   \Umathcode`S=0+\M@frak@num"1D516\relax
3033   \Umathcode`T=0+\M@frak@num"1D517\relax
3034   \Umathcode`U=0+\M@frak@num"1D518\relax
3035   \Umathcode`V=0+\M@frak@num"1D519\relax
3036   \Umathcode`W=0+\M@frak@num"1D51A\relax
3037   \Umathcode`X=0+\M@frak@num"1D51B\relax
3038   \Umathcode`Y=0+\M@frak@num"1D51C\relax
3039   \Umathcode`Z=0+\M@frak@num"2128\relax
3040   \Umathcode`a=0+\M@frak@num"1D51E\relax
3041   \Umathcode`b=0+\M@frak@num"1D51F\relax
3042   \Umathcode`c=0+\M@frak@num"1D520\relax
3043   \Umathcode`d=0+\M@frak@num"1D521\relax
3044   \Umathcode`e=0+\M@frak@num"1D522\relax
3045   \Umathcode`f=0+\M@frak@num"1D523\relax
3046   \Umathcode`g=0+\M@frak@num"1D524\relax
3047   \Umathcode`h=0+\M@frak@num"1D525\relax
```

```
3048   \Umathcode`i=0+\M@frak@num"1D526\relax
3049   \Umathcode`j=0+\M@frak@num"1D527\relax
3050   \Umathcode`k=0+\M@frak@num"1D528\relax
3051   \Umathcode`l=0+\M@frak@num"1D529\relax
3052   \Umathcode`m=0+\M@frak@num"1D52A\relax
3053   \Umathcode`n=0+\M@frak@num"1D52B\relax
3054   \Umathcode`o=0+\M@frak@num"1D52C\relax
3055   \Umathcode`p=0+\M@frak@num"1D52D\relax
3056   \Umathcode`q=0+\M@frak@num"1D52E\relax
3057   \Umathcode`r=0+\M@frak@num"1D52F\relax
3058   \Umathcode`s=0+\M@frak@num"1D530\relax
3059   \Umathcode`t=0+\M@frak@num"1D531\relax
3060   \Umathcode`u=0+\M@frak@num"1D532\relax
3061   \Umathcode`v=0+\M@frak@num"1D533\relax
3062   \Umathcode`w=0+\M@frak@num"1D534\relax
3063   \Umathcode`x=0+\M@frak@num"1D535\relax
3064   \Umathcode`y=0+\M@frak@num"1D536\relax
3065   \Umathcode`z=0+\M@frak@num"1D537\relax}}
```

Set bold caligraphic letters.

```
3066 \def\M@bcal@set{%
3067   \protected\def\mathbcal##1{\relax
3068     \ifmmode\else
3069       \M@HModeError\mathbcal
3070       $%
3071     \fi
3072     \begingroup
3073       \M@bcal@mathcodes
3074       ##1%
3075     \endgroup}
3076   \edef\M@bcal@num{\number\csname symM\M@bcalshape\@tempa\endcsname}
3077   \protected\edef\M@bcal@mathcodes{%
3078   \Umathcode`A=0+\M@bcal@num"1D4D0\relax
3079   \Umathcode`B=0+\M@bcal@num"1D4D1\relax
3080   \Umathcode`C=0+\M@bcal@num"1D4D2\relax
3081   \Umathcode`D=0+\M@bcal@num"1D4D3\relax
3082   \Umathcode`E=0+\M@bcal@num"1D4D4\relax
3083   \Umathcode`F=0+\M@bcal@num"1D4D5\relax
3084   \Umathcode`G=0+\M@bcal@num"1D4D6\relax
3085   \Umathcode`H=0+\M@bcal@num"1D4D7\relax
3086   \Umathcode`I=0+\M@bcal@num"1D4D8\relax
3087   \Umathcode`J=0+\M@bcal@num"1D4D9\relax
3088   \Umathcode`K=0+\M@bcal@num"1D4DA\relax
3089   \Umathcode`L=0+\M@bcal@num"1D4DB\relax
3090   \Umathcode`M=0+\M@bcal@num"1D4DC\relax
3091   \Umathcode`N=0+\M@bcal@num"1D4DD\relax
3092   \Umathcode`O=0+\M@bcal@num"1D4DE\relax
3093   \Umathcode`P=0+\M@bcal@num"1D4DF\relax
```

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

Set bold fraktur letters.

```
3130 \def\M@bfrak@set{%
3131    \protected\def\mathbfrak##1{\relax
3132      \ifmmode\else
3133        \M@HModeError\mathbfrak
3134        $%
3135      \fi
3136      \begingroup
3137        \M@bfrak@mathcodes
3138        ##1%
3139      \endgroup}
```

```
3140    \edef\M@bfrak@num{\number\csname symM\M@bfrakshape\@tempa\endcsname}
3141    \protected\edef\M@bfrak@mathcodes{%
3142    \Umathcode`A=0+\M@bfrak@num"1D56C\relax
3143    \Umathcode`B=0+\M@bfrak@num"1D56D\relax
3144    \Umathcode`C=0+\M@bfrak@num"1D56E\relax
3145    \Umathcode`D=0+\M@bfrak@num"1D56F\relax
3146    \Umathcode`E=0+\M@bfrak@num"1D570\relax
3147    \Umathcode`F=0+\M@bfrak@num"1D571\relax
3148    \Umathcode`G=0+\M@bfrak@num"1D572\relax
3149    \Umathcode`H=0+\M@bfrak@num"1D573\relax
3150    \Umathcode`I=0+\M@bfrak@num"1D574\relax
3151    \Umathcode`J=0+\M@bfrak@num"1D575\relax
3152    \Umathcode`K=0+\M@bfrak@num"1D576\relax
3153    \Umathcode`L=0+\M@bfrak@num"1D577\relax
3154    \Umathcode`M=0+\M@bfrak@num"1D578\relax
3155    \Umathcode`N=0+\M@bfrak@num"1D579\relax
3156    \Umathcode`O=0+\M@bfrak@num"1D57A\relax
3157    \Umathcode`P=0+\M@bfrak@num"1D57B\relax
3158    \Umathcode`Q=0+\M@bfrak@num"1D57C\relax
3159    \Umathcode`R=0+\M@bfrak@num"1D57D\relax
3160    \Umathcode`S=0+\M@bfrak@num"1D57E\relax
3161    \Umathcode`T=0+\M@bfrak@num"1D57F\relax
3162    \Umathcode`U=0+\M@bfrak@num"1D580\relax
3163    \Umathcode`V=0+\M@bfrak@num"1D581\relax
3164    \Umathcode`W=0+\M@bfrak@num"1D582\relax
3165    \Umathcode`X=0+\M@bfrak@num"1D583\relax
3166    \Umathcode`Y=0+\M@bfrak@num"1D584\relax
3167    \Umathcode`Z=0+\M@bfrak@num"1D585\relax
3168    \Umathcode`a=0+\M@bfrak@num"1D586\relax
3169    \Umathcode`b=0+\M@bfrak@num"1D587\relax
3170    \Umathcode`c=0+\M@bfrak@num"1D588\relax
3171    \Umathcode`d=0+\M@bfrak@num"1D589\relax
3172    \Umathcode`e=0+\M@bfrak@num"1D58A\relax
3173    \Umathcode`f=0+\M@bfrak@num"1D58B\relax
3174    \Umathcode`g=0+\M@bfrak@num"1D58C\relax
3175    \Umathcode`h=0+\M@bfrak@num"1D58D\relax
3176    \Umathcode`i=0+\M@bfrak@num"1D58E\relax
3177    \Umathcode`j=0+\M@bfrak@num"1D58F\relax
3178    \Umathcode`k=0+\M@bfrak@num"1D590\relax
3179    \Umathcode`l=0+\M@bfrak@num"1D591\relax
3180    \Umathcode`m=0+\M@bfrak@num"1D592\relax
3181    \Umathcode`n=0+\M@bfrak@num"1D593\relax
3182    \Umathcode`o=0+\M@bfrak@num"1D594\relax
3183    \Umathcode`p=0+\M@bfrak@num"1D595\relax
3184    \Umathcode`q=0+\M@bfrak@num"1D596\relax
3185    \Umathcode`r=0+\M@bfrak@num"1D597\relax
3186    \Umathcode`s=0+\M@bfrak@num"1D598\relax
```

```
3187    \Umathcode`t=0+\M@bfrak@num"1D599\relax
3188    \Umathcode`u=0+\M@bfrak@num"1D59A\relax
3189    \Umathcode`v=0+\M@bfrak@num"1D59B\relax
3190    \Umathcode`w=0+\M@bfrak@num"1D59C\relax
3191    \Umathcode`x=0+\M@bfrak@num"1D59D\relax
3192    \Umathcode`y=0+\M@bfrak@num"1D59E\relax
3193    \Umathcode`z=0+\M@bfrak@num"1D59F\relax}}
```

And that's everything!

Version History

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

1.1b . . . . July 2018
—initial release

1.2 . . . . August 2018
—minor bug fix for `\mathfrak`
—eliminated redundant batchfile

1.3 . . . . January 2019
—added `symbols` keyword
—created `mathfont_example.pdf`
—corrected the description of the `mathastext` package
—font-change `\message` added to `\mathfont`

1.4 . . . . April 2019
—`\setfont` command added
—`\mathfont` optional argument can parse spaces
—`no-operators` now default package optional argument
—added `\comma` command
—new fancy fatal error message
—improved messaging for `\mathfont`
—internal command `\mathpound` changed to `\mathhash`
—added a missing `#1` after `\char`\"` in the example code redefining " in the user guide

1.5 . . . . April 2019
—separated `\increment` and `\Delta`
—version history added
—initial off-the-shelf use insert added

1.6 . . . . December 2019
—separated implementation and user documentation
—created `mathfont_heading.tex`
—created `mathfont_doc_patch.tex` for use with the index
—changed `mathfont_greek.pdf` to `mathfont_symbol_list.pdf`

—eliminated `mathfont_example.pdf`
—eliminated `operators` package option
—eliminated `packages` package option
—font name can be package option
—added Hebrew and Cyrillic characters
—separated ancient Greek from modern Greek characters
—created new keywords: `extsymbols`, `delimiters`, `arrows`, `diacritics`, `bigops`, `extbigops`
—improved messaging
—improved internal code for local font-change commands
—improved space parsing for the optional argument of `\mathfont`
—bug fix for `\#`, etc. commands
—bad input for `\mathbb`, etc. now gives a warning
—improved error checking for `\newmathrm`, etc. commands
—`\mathfont` now ignores bad options (on top of issuing an error)
—iternal commands now begin with `\M@…`
—added Easter Egg!
—improved indexing
—`mathfont.dtx` renamed as `mathfont_code.dtx`
—`\newmathbold` renamed as `\newmathbf`
—default local font changes now use `\updefault`, etc.
—added fatal error for missing `fontspec`
—fatal errors result in `\endinput` rather than `\@@end`

2.0 . . . . December 2021

> **Big Change:** Font adjustments for LuaT_EX: new glyph boundaries for Latin letters in math mode, resizable delimiters, actual big operators, MathConstants table based on font metrics.

—added `\CharmLine` and `\CharmFile`

—added `\mathconstantsfont`

—certain dimensions in equations are now adjustable when typesetting with LuaTeX

—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 `\Umathcode`s 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

—calling Plain TeX on `mathfont_code.dtx` produces sty file and no pdf file

Index

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