

Package **mathfont** v. 2.4 Implementation

Conrad Kosowsky

April 2025

`kosowsky.latex@gmail.com`

For easy, off-the-shelf use, type the following in your preamble and compile with \LaTeX or \LuaTeX :

`\usepackage[]{mathfont}`

As of version 2.0, using \LuaTeX 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 \LuaTeX , **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.

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

1 Setup

First, the package should declare itself. The first 61 lines of `mathfont.sty` are comments.

```
62 \NeedsTeXFormat{LaTeX2e}
63 \ProvidesPackage{mathfont}[2025/04/03 v. 2.4 Package mathfont]
```

Informational message.

```
64 \def\@mathfontinfo#1{\wlog{Package mathfont Info: #1}}
```

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

```
65 \newif\ifM@XeTeXLuaTeX      % is engine one of xetex or luatex?
66 \newif\ifM@Noluaotfload     % cannot find luaotfload.sty?
67 \newif\ifM@adjust@font      % should adjust fonts with lua script?
68 \newif\ifM@font@loaded      % load mathfont with font specified?
69 \newif\ifE@sterEggDecl@red  % already did easter egg?
70 \newcount\M@loader          % specifies which font-loader to use
```

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

```
71 \long\def\@gobble@brackets[#1]{}
\M@NoMathfontError 72 \def\M@NoMathfontError#1{\PackageError{mathfont}
73   {\MessageBreak Invalid command\MessageBreak
74   \string#1 on line \the\inputlineno}
75   {Your command was ignored. I couldn't\MessageBreak
76   load mathfont, so I never defined this\MessageBreak
77   control sequence.}}
```

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

```
78 \@tfor\@i:=\setfont
79   \RuleThicknessFactor
80   \IntegralItalicFactor
```

```

81 \SurdVerticalFactor
82 \SurdHorizontalFactor
83 \CharmLine
84 \CharmFile\do{%
85   \edef\@tempa{\protected\def\expandafter\noexpand\@i{%
86     \noexpand\M@NoMathfontError\expandafter\noexpand\@i
87     \noexpand\@gobble}}
88   \@tempa}

```

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

```

89 \@tfor\@i:=\newmathrm
90   \newmathit
91   \newmathbf
92   \newmathbfit
93   \newmathsc
94   \newmathscit
95   \newmathbfsc
96   \newmathbfscit\do{%
97     \edef\@tempa{\protected\def\expandafter\noexpand\@i{%
98       \noexpand\M@NoMathfontError\expandafter\noexpand\@i
99       \noexpand\@gobbletwo}}
100     \@tempa}

```

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

```

101 \protected\def\mathfont{%
102   \M@NoMathfontError\mathfont
103   \@ifnextchar[{\expandafter\@gobble\expandafter\@gobble@brackets}
104     {\@gobble}}
\mathconstantsfont 105 \protected\def\mathconstantsfont{%
106   \M@NoMathfontError\mathconstantsfont
107   \@ifnextchar[{\expandafter\@gobble\expandafter\@gobble@brackets}
108     {\@gobble}}

```

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

```

\newmathfontcommand 109 \protected\def\newmathfontcommand{%
110   \M@NoMathfontError\newmathfontcommand\@gobblefour}

```

Check that the engine is \XeTeX or \LuaTeX . If yes, set `\ifM@XeTeXLuaTeX` to true. (Otherwise the conditional will be false by default.)

```

111 \ifdefined\directlua

```

```

112 \M@XeTeXLuaTeXtrue
113 \fi
114 \ifdefined\XeTeXrevision
115 \M@XeTeXLuaTeXtrue
116 \fi

```

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

```

117 \ifM@XeTeXLuaTeX\else
118 \begingroup
119 \catcode`\ =12\relax
\M@XeTeXLuaTeXError 120 \def\M@XeTeXLuaTeXError{\GenericError{}}%
121 {\MessageBreak\MessageBreak
122 Package mathfont error:%
123 \MessageBreak\MessageBreak
124 *****\MessageBreak
125 *                               *\MessageBreak
126 *      UNABLE TO                *\MessageBreak
127 *      LOAD MATHFONT            *\MessageBreak
128 *                               *\MessageBreak
129 *      Missing XeTeX            *\MessageBreak
130 *      or LuaTeX                *\MessageBreak
131 *                               *\MessageBreak
132 *****\MessageBreak\@gobbletwo}%
133 {See the mathfont package documentation for explanation.}%
134 {I need XeTeX or LuaTeX to use mathfont. It\MessageBreak
135 looks like the current engine is something\MessageBreak
136 else, so I'm going to stop reading in the\MessageBreak
137 package file now. (You won't be able to use\MessageBreak
138 commands from mathfont in your document.) To\MessageBreak
139 load mathfont correctly, please retypeset your\MessageBreak

```

```

140 document with one of those two engines.^^J}}%
141 \expandafter\endgroup
142 \M@XeTeXLuaTeXError
143 \AtEndOfPackage{%
144   \typeout{:: mathfont :: Failed to load\on@line.}}
145 \expandafter\endinput % we \endinput with a balanced conditional
146 \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 `XYTeX`, issue a warning.

```

147 \ifdefined\directlua
148   \M@adjust@fonttrue % if engine is LuaTeX, adjust font by default
149   \IfFileExists{luaotfload.sty}
150     {\M@Noluaotfloadfalse\RequirePackage{luaotfload}}
151     {\M@Noluaotfloadtrue}
152 \else
153   \AtEndOfPackage{\PackageWarningNoLine{mathfont}{%
154     The current engine is XeTeX, but as\MessageBreak
155     of mathfont version 2.0, LuaTeX is\MessageBreak
156     recommended. Consider compiling with\MessageBreak
157     LuaLaTeX. Certain features will not\MessageBreak
158     work with XeTeX}}
159 \fi

```

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

```

160 \ifM@Noluaotfload % true if LuaTeX AND no luaotfload.sty
161 \begingroup
162 \catcode`\ =12\relax
\M@NoluaotfloadErr 163 \def\M@NoluaotfloadError{\GenericError{}%
164 {\MessageBreak\MessageBreak
165 Package mathfont error:%
166 \MessageBreak\MessageBreak
167 *****\MessageBreak
168 *
169 *          UNABLE TO          *\MessageBreak
170 *          LOAD MATHFONT      *\MessageBreak
171 *
172 *          *\MessageBreak

```

```

172 *      Cannot find the      *\MessageBreak
173 *   file luaotfload.sty *\MessageBreak
174 *                                     *\MessageBreak
175 *****\MessageBreak\@gobbletwo}%
176 {You are likely seeing this message because you haven't^^J%
177 installed luaotfload. Check your TeX distribution for a^^J%
178 list of the packages on your system.^^J^^J%
179 See the mathfont documentation for further explanation.}%
180 {You're in trouble here. It looks like the current\MessageBreak
181 engine is LuaTeX, so I need the luaotfload package\MessageBreak
182 to make mathfont work properly. However, I can't\MessageBreak
183 find luaotfload, which likely means something is\MessageBreak
184 wrong with your TeX installation. I'm going to stop\MessageBreak
185 reading in the mathfont package file. (You won't be\MessageBreak
186 able to use commands from mathfont in your document.)\MessageBreak
187 To load mathfont work correctly, make sure you have\MessageBreak
188 installed luaotfload.sty in a directory searchable\MessageBreak
189 by TeX or compile with XeLaTeX.^^J}}%
190 \expandafter\endgroup
191 \M@NoluaotfloadError
192 \AtEndOfPackage{%
193   \typeout{:: mathfont :: Failed to load\on@line.}}
194 \expandafter\endinput % we \endinput with a balanced conditional
195 \fi

```

As of version 2.4, I'm taking out the deprecated package options, which used to live here. Easter egg!!

```

196 \DeclareOption{easter-egg}{%
197   \ifE@sterEggDecl@red\else
198     \E@sterEggDecl@redtrue
199     \newcount\@easter@egg@
200     \def\EasterEggUpdate{\ProcessE@sterEgg\show\E@sterEggUpd@te}
201     \let\ProcessE@sterEgg\relax

```

Two status updates during package loading.

```

202   \edef\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
203     Okay, opening your Easter egg.^^J%
204     Type \string\EasterEggUpdate\space in your^^J%
205     document to see the status.^^J^^J}
206   \EasterEggUpdate
207   \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
208     Uh oh. It looks like your Easter^^J%
209     egg flew out the window. I don't^^J%

```

```

210         I don't suppose you know the best^^J%
211         kind of bait to lure an egg?^^J^^J}
212     \EasterEggUpdate

```

Possible updates if the user types \EasterEggUpdate. We define the status update with \ProcessE@sterEgg, which stores the current message in \E@sterEggUpd@te and changes the message as the user calls \EasterEggUpdate. The count \@easter@egg@ keeps track of how many times the user has requested a status update.

```

213     \def\ProcessE@sterEgg{%
214         \edef\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
215         \ifodd\@easter@egg@
216             \ifcase\numexpr(\@easter@egg@ - 1) / 2\relax
217                 An Easter bunny must be related to a^^J%
218                 platypus, no? Some sort of monotreme...%
219             \or
220                 Don't count your chickens before they hatch^^J%
221                 out of Easter eggs! But we don't have any^^J%
222                 chickens right now because there are no eggs,^^J%
223                 and the supply chain is sad.%
224             \or
225                 Sorry, I'm late to a meeting. Can't talk right now.%
226             \or
227                 Sunday, Monday, Tuesday, Wednesday, also^^J%
228                 known as hump day, as in camel humps, which^^J%
229                 I must say look distinctly egg-like if you^^J%
230                 squint.%
231             \or
232                 I'm calling Eggs Anonymous!%
233             \or
234                 Sorry, I'm on the phone. Can't talk right now.%
235             \or
236                 Still haven't found your Easter egg. I know^^J%
237                 it's floating around here somewhere. Like an^^J%
238                 asteroid in space, hopefully without the^^J%
239                 massive extinction event.%
240             \or
241                 Did you know eggs are used to make certain^^J%
242                 types of vaccines? PSA: get your flu shot^^J%
243                 and your covid shot!%
244             \or
245                 Three large eggs.^^J%
246                 Three large eggs.^^J%

```

```

247         See how they crack.^^J%
248         See how they crack.^^J%
249         Their broken shells are so pearly white.^^J%
250         In simmering water they catch the light.^^J%
251         Did you ever see such a sight in your life^^J%
252         As three poached eggs?%
253     \or
254         Do gnus eat eggs? Surely they must.%
255     \or
256         Okay, I have a fishing rod, some twine, and^^J%
257         a hook, but I still haven't caught your Easter^^J%
258         egg. Apparently it's harder to catch an egg^^J%
259         than a fish.%
260     \or
261         Sorry, I'm out fishing. Can't talk right now.%
262     \or
263         Is ghoti really an acceptable phonetic^^J%
264         spelling of fish? I am skeptical.%
265     \or
266         Perhaps an Easter bunny is actually a species^^J%
267         of fish. A rabbit fish.%
268     \else
269         Sorry, I'm all out of witty things to say.^^J%
270         Check back later.%
271     \fi
272     \else
273         Still wrangling. Check back later.%
274     \fi^^J^^J}%
275     \global\advance\@easter@egg@\@ne}

```

One status update \AtBeginDocument.

```

276     \AtBeginDocument{\bgroup
277         \let\ProcessE@sterEgg\relax
278         \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
279             If we have zero eggs^^J%
280             and zero bunnies, how^^J%
281             many gnats does it take^^J%
282             to change a lightbulb??^^J^^J}
283         \EasterEggUpdate
284     \egroup}

```

One update at the first instance of math mode, assuming another package doesn't overwrite the contents of \everymath first.


```

285 \def\math@E@sterEggUpd@te{\begingroup
286 \let\ProcessE@sterEgg\relax
287 \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
288 Scrambled, poached, or sunny side up?^^J^^J}%
289 \EasterEggUpdate
290 \endgroup
291 \global\let\math@E@sterEggUpd@te\relax}
292 \everymath\expandafter{\the\everymath\math@E@sterEggUpd@te}

```

Two status updates `\AtEndDocument`, including the egg itself. First, we disable `\ProcessE@sterEgg` since we don't need it anymore. Then inside a group, we make the control symbols `*`, `\`, and `\=` expand to their own names and do some extreme catcode sports. We convert `+` to active and make it expand to a space. Because everything has already been tokenized inside `\DeclareOption`, we have to retokenize the definition of `+` inside `\scantokens`, and we set `\everyeof` to `\noexpand` to avoid an end-of-file error.

```

293 \AtEndDocument{\let\ProcessE@sterEgg\relax
294 \begingroup
295 \edef\*{\@backslashchar*}
296 \edef\/{\@backslashchar/}
297 \edef\={\@backslashchar=}
298 \catcode`\+=\active
299 \everyeof{\noexpand}
300 \scantokens{\def+{ }}

```

At this point we are ready to make the egg message. Again, we have to retokenize everything with `\scantokens` because it was previously tokenized. However, if we write `^^J` directly inside `\scantokens`, that primitive will convert the newline to a blank space, so instead we store `^^J` in `\@tempb`. After the `\edef` expands `\scantokens`, it also expands each `\@tempb`, so `\@tempa` has the line breaks we want.

```

301 \def\@tempb{^^J}
302 \edef\@tempa{\scantokens{Easter Egg Status:\@tempb\@tempb
303 The egg has been retrieved. What\@tempb
304 pinnacle of pulchritude!\@tempb\@tempb
305 *****\@tempb
306 *****\@tempb
307 *****\@tempb
308 +---\@tempb
309 *****\@tempb
310 +---/\---/\---/\---/\---\@tempb
311 +---\/---\/---\/---\/---\@tempb
312 +---\@tempb

```

```

313 *****\@tempb
314 ***/\****/\****/\****/\****/\****\@tempb
315 ***\/\****\/\****\/\****\/\****\@tempb
316 *****\@tempb
317 *****\@tempb
318 *****/\****/\****/\****/\****\@tempb
319 +*****\/\****\/\****\/\****\@tempb
320 ++*****\@tempb
321 +++-----\@tempb
322 +++++*****\@tempb
323 +++++*****\@tempb
324 ++++++\=====/\@tempb
325 ++++++\=====/\@tempb
326 ++++++\=====/\@tempb
327 ++++++|====|\@tempb
328 ++++++/===== \@backslashchar\@tempb
329 ++++++(___)\@tempb}}

```

Then end the group and store the message in `\E@sterEggUpd@te`.

```

330 \expandafter\endgroup\expandafter
331 \def\expandafter\E@sterEggUpd@te\expandafter{\@tempa}
332 \EasterEggUpdate
333 \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
334   Happy, happy day! Happy,^^J%
335   happy day! Clap your hands,^^J%
336   and be glad your hovercraft^^J%
337   isn't full of eels!^^J^^J}
338 \EasterEggUpdate
339 \let\E@sterEggUpd@te\relax
340 \let\EasterEggUpdate\relax}
341 \fi}% my easter egg :)

```

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

```

342 \DeclareOption{default-loader}{\M@loader\z@}
343 \DeclareOption{fontspec-loader}{\M@loader\@ne}

```

The options `adjust` and `no-adjust` determine whether `mathfont` applies Lua-based font adjustments to fonts loaded in the future.

```

344 \DeclareOption{adjust}{\M@adjust@fonttrue}
345 \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`.

```

346 \DeclareOption*{\M@font@loadedtrue
347   \edef\M@font@load{\CurrentOption}}
348 \ProcessOptions*

```

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

```

349 \ifcase\M@loader
350   \@mathfontinfo{Default font-loader was
351     requested for font loading.}
352 \or
353   \@mathfontinfo{Package fontspec was
354     requested for font loading.}

```

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

```

355   \@ifpackageloaded{fontspec}
356     {\@mathfontinfo{Package fontspec detected.}
357       \csname bool_if:NTF\expandafter\endcsname
358         \csname g__fontspec_math_bool\endcsname
359         {\@mathfontinfo{Setting
360           \string\g__fontspec_math_bool to false.}
361           \csname bool_set_false:N\expandafter\endcsname
362             \csname g__fontspec_math_bool\endcsname}{\relax}}}

```

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

```

363   {\@mathfontinfo{Package fontspec not detected.}
364     \IfFileExists{fontspec.sty}
365       {\@mathfontinfo{File fontspec.sty was found.}
366         \@mathfontinfo{Loading fontspec.}
367         \RequirePackage[no-math]{fontspec}}
368     {\PackageError{mathfont}
369       {Missing package fontspec;\MessageBreak
370         using default font-loader instead}
371       {You requested fontspec as
372         the font-loader for\MessageBreak
373         mathfont. However, I can't
374         find the fontspec\MessageBreak
375         package file, so I'm going
376         to use the default\MessageBreak
377         font-loader instead. (This
378         likely means that\MessageBreak
379         something is wrong with your
380         TeX installation.)\MessageBreak

```

```

381      Check your TeX distribution
382      for a list of the\MessageBreak
383      packages installed on your
384      system. To resolve\MessageBreak
385      this error, make sure fontspec
386      is installed in\MessageBreak
387      a directory searchable by TeX or
388      load mathfont\MessageBreak
389      with the default-loader option.^^J}
390      \M@loader\z@}}
391 \fi

```

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

```

392 \ifcase\M@loader
393   \@mathfontinfo{Using default font-loader.}
394   \AtEndOfPackage{%
395     \typeout{:: mathfont :: Using default font-loader.}}
\M@otf@features 396   \def\M@otf@features{script=latin;language=DFLT;%
397     +tlig;+liga;+lnum}
398 \or
399   \@mathfontinfo{Using fontspec as font-loader.}
400   \AtEndOfPackage{%
401     \typeout{:: mathfont :: Using fontspec as font-loader.}}
\M@otf@features 402   \def\M@otf@features{Script=Latin,
403     Language=Default,
404     Ligatures={TeX,Common},
405     Numbers=Lining}
406 \fi

```

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

```

407 \ifdefined\directlua
408   \ifM@adjust@font
409     \@mathfontinfo{Enabling Lua-based font adjustments.}
410     \AtEndOfPackage{%
411       \typeout{:: mathfont :: Lua-based font adjustments
412         enabled.}}
413   \else

```

```

414 \mathfontinfo{Disabling Lua-based font adjustments.}
415 \AtEndOfPackage{%
416   \typeout{:: mathfont :: Lua-based font adjustments
417     disabled.}}
418 \fi
419 \else

```

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

```

420 \ifM@adjust@font
421   \PackageError{mathfont}{Option^^J"adjust" ignored with XeTeX}
422   {Your package option "adjust" was ignored.\MessageBreak
423   This option works only with LuaTeX, and it\MessageBreak
424   looks like the current engine is XeTeX. To\MessageBreak
425   enable Lua-based font adjustments, typeset\MessageBreak
426   with LuaLaTeX.^^J}
427   \M@adjust@fontfalse
428 \fi
429 \mathfontinfo{Disabling Lua-based font adjustments.}
430 \AtEndOfPackage{%
431   \typeout{:: mathfont :: Lua-based font adjustments disabled.}}
432 \fi

```

2 Messages and Errors

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.

```

\M@SymbolFontInfo 433 \def\M@SymbolFontInfo#1#2#3#4{\wlog{}}%
434 \mathfontinfo{Declaring new symbol font from #1!^^J%
435 NFSS Family Name: \space#2^^J%
436 Series/Shape Info: #3^^J%
437 Symbol Font Name: \space#4^^J}}
\M@FontChangeInfo 438 \def\M@FontChangeInfo#1#2{\mathfontinfo{Setting #1 chars to #2!}}
\M@NewFontCommandI 439 \def\M@NewFontCommandInfo#1#2#3#4#5{\wlog{}}%
440 \mathfontinfo{Creating \string#1 using #2!^^J%
441 NFSS Family Name: \space#3^^J%
442 Series/Shape Info: #4/#5^^J}}
\M@CharsSetWarning 443 \def\M@CharsSetWarning#1{%
444 \PackageWarning{mathfont}
445 {I already set the font for\MessageBreak

```

Table 1: Various Messages and Errors and Their Uses

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

```

446 #1 chars, so I'm ignoring\MessageBreak
447 this option for \string\mathfont\space
448 on line \the\inputlineno\@gobble}}
\M@MissingNFSSShap 449 \def\M@MissingNFSSShapesWarning#1#2{%
450 \PackageWarning{mathfont}
451 {The nfss family "#1"\MessageBreak
452 from line \the\inputlineno\space
453 is missing shapes. You\MessageBreak
454 may see some substitutions or errors.\MessageBreak
455 Missing shape(s):#2\@gobble}}
\M@NoBaseModeDetec 456 \def\M@NoBaseModeDetectedWarning#1{%
457 \PackageWarning{mathfont}
458 {I couldn't find a base-mode version of\MessageBreak
459 the nfss family "#1"\MessageBreak
460 from line \the\inputlineno,
461 so I'm using the family\MessageBreak
462 you specified for the math font. Some\MessageBreak
463 font features may not work in math\MessageBreak
464 mode\@gobble}}

```

Error messages associated with `\mathfont`.

```

\M@InvalidOptionEr 465 \def\M@InvalidOptionError#1{%
466 \PackageError{mathfont}
467 {Invalid^^Joption "#1"
468 for \string\mathfont\on@line}
469 {Hm. You used a keyword that
470 isn't actually an optional\MessageBreak
471 argument for \string\mathfont. Check
472 that you spelled the keyword\MessageBreak
473 correctly. Otherwise, I'm not sure
474 what's wrong. Is this\MessageBreak
475 option listed in the package documentation?
476 In any event,\MessageBreak
477 I'm going to ignore it.^^J}}
\M@InvalidSuboptio 478 \def\M@InvalidSuboptionError#1{%
479 \PackageError{mathfont}
480 {Invalid^^Jsuboption "#1"
481 for \string\mathfont\on@line}
482 {Hm. You used a keyword that
483 isn't actually a suboption\MessageBreak
484 for \string\mathfont. Check that you
485 spelled the keyword correctly.\MessageBreak
486 Otherwise, I'm not sure what's wrong.

```

```

487   Is this suboption\MessageBreak
488   listed in the package documentation?
489   In any event, I'm\MessageBreak
490   going to ignore it.^^J}}
\M@MissingOptionEr 491 \def\M@MissingOptionError{%
492   \PackageError{mathfont}
493   {Missing^^Joption for
494   \string\mathfont\on@line}
495   {It looks like you
496   included a , or = in\MessageBreak
497   the optional argument of
498   \string\mathfont\space but\MessageBreak
499   didn't put anything before it.^^J}}
\M@MissingSuboptio 500 \def\M@MissingSuboptionError{%
501   \PackageError{mathfont}
502   {Missing^^Jsuboption for
503   \string\mathfont\on@line}
504   {It looks like you
505   included an = somewhere\MessageBreak
506   in the optional argument of
507   \string\mathfont\space but\MessageBreak
508   didn't put the suboption
509   after it. Either\MessageBreak
510   that or you typed == instead of =.^^J}}
Error messages for \mathconstantsfont.
\M@BadMathConstant 511 \def\M@BadMathConstantsFontError#1{%
512   \PackageError{mathfont}
513   {Invalid font specifier\MessageBreak
514   for \string\mathconstantsfont:\MessageBreak
515   "#1"}
516   {Your command was ignored---I
517   can't parse your argument.\MessageBreak
518   Please make sure to use text
519   that you have previously\MessageBreak
520   fed to \string\mathfont\space
521   for the argument of
522   \string\mathconstantsfont.^^J}}
\M@BadMathConstant 523 \def\M@BadMathConstantsFontTypeError#1{%
524   \PackageError{mathfont}
525   {Invalid\MessageBreak font specifier for
526   \string\mathconstantsfont:\MessageBreak"#1"}
527   {The optional argument of

```



```

528 \string\mathconstantsfont\MessageBreak
529 should be "upright" or "italic."
530 Right now,\MessageBreak
531 it's "#1."^^J}}

```

```

\M@MathConstantsNo 532 \def\M@MathConstantsNoAdjustWarning{%
533 \PackageWarning{mathfont}
534 {Your \string\mathconstantsfont\space
535 on line \the\inputlineno\MessageBreak
536 is mainly for use in
537 LuaTeX with font\MessageBreak
538 adjustments enabled.
539 In the current\MessageBreak
540 situation, it is
541 probably not doing\MessageBreak
542 anything\@gobble}}

```

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

```

\M@MissingControlS 543 \def\M@MissingControlSequenceError#1#2{%
544 \PackageError{mathfont}
545 {Missing control sequence\MessageBreak
546 for\string#1\MessageBreak
547 on input line \the\inputlineno}
548 {Your command was ignored.
549 Right now the\MessageBreak
550 first argument of
551 \string#1\space is "#2." \MessageBreak
552 Please use a control
553 sequence instead.^^J}}

```

```

\M@DoubleArgError 554 \def\M@DoubleArgError#1#2{%
555 \PackageError{mathfont}
556 {Multiple characters in\MessageBreak
557 first argument of \string#1\MessageBreak
558 on input line \the\inputlineno}
559 {Your command was ignored.
560 Right now the\MessageBreak
561 first argument of
562 \string#1\space is "#2," \MessageBreak
563 which is multiple characters.
564 Please use \MessageBreak
565 a single character instead.^^J}}

```

```

\M@HModeError 566 \def\M@HModeError#1{%
567 \PackageError{mathfont}
568 {Missing \string$ inserted\MessageBreak

```

```

569 on input line line \the\inputlineno}
570 {I generated an error because
571 you used \string#1\space
572 outside of\MessageBreak
573 math mode. I inserted a \string$
574 before your \string#1,
575 so we\MessageBreak
576 should be all good now.^^J}}

```

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

```

\M@ForbiddenCharmL 577 \def\M@ForbiddenCharmLine#1{%
578   \PackageError{mathfont}
579   {Forbidden charm info contains #1}
580   {The argument of your \string\CharmLine\space
581   macro on line \the\inputlineno\MessageBreak
582   contains the character #1,
583   which will mess me up\MessageBreak
584   if I try to read it, so I'm
585   ignoring this call\MessageBreak
586   to \string\CharmLine. To resolve
587   this error, make sure\MessageBreak
588   your charm information
589   contains only integers,\MessageBreak
590   floats, asterisks, commas, and spaces.^^J}}
\M@ForbiddenCharmF 591 \def\M@ForbiddenCharmFile#1{%
592   \PackageError{mathfont}
593   {Forbidden charm info contains #1}
594   {One of the lines in your
595   \string\CharmFile\space
596   from line \the\inputlineno\MessageBreak
597   contains the character #1,
598   which will mess me up\MessageBreak
599   if I try to read it, so I'm
600   ignoring this line\MessageBreak
601   from your file. To resolve
602   this error, make sure\MessageBreak
603   your charm information
604   contains only integers,\MessageBreak
605   floats, asterisks, commas, and spaces.^^J}}
\M@NoFontAdjustErr 606 \def\M@NoFontAdjustError#1{%
607   \PackageError{mathfont}
608   {Your command \MessageBreak
609   \string#1 is invalid\MessageBreak

```

```

610 without Lua-based font adjustments}
611 {You haven't enabled Lua-based
612 font adjustments,\MessageBreak
613 but the macro you called won't
614 do anything without\MessageBreak
615 them. I'm going to ignore your
616 command for now. To\MessageBreak
617 resolve this error, load mathfont
618 with the package\MessageBreak
619 option "adjust" or compile with LuaLaTeX.^^J}}
\M@BadIntegerError 620 \def\M@BadIntegerError#1#2{%
621 \PackageError{mathfont}
622 {Bad argument for\MessageBreak\string#1}
623 {Your command was ignored.
624 Please make sure\MessageBreak
625 that your argument of
626 \string#1\space\MessageBreak
627 is a nonnegative integer.
628 Right now it's\MessageBreak
629 "#2".^^J}}

```

3 Default Settings

We save four macros from the L^AT_EX kernel for safe-keeping, and then we change their definitions. To make the symbol declaration macros from the kernel compatible with unicode fonts, we convert the hexadecimal digits in `\count0` and `\count2` back to decimal and change the `\math` primitive to `\Umath`.

```

630 \let\@@set@mathchar\set@mathchar
631 \let\@@set@mathsymbol\set@mathsymbol
632 \let\@@set@mathaccent\set@mathaccent
633 \let\@@DeclareSymbolFont\DeclareSymbolFont
634 \let\@@DeclareSymbolFont@m@dropped\DeclareSymbolFont@m@dropped
635 \@onlypreamble\@@set@mathchar
636 \@onlypreamble\@@set@mathsymbol
637 \@onlypreamble\@@set@mathaccent
638 \@onlypreamble\@@DeclareSymbolFont
639 \@onlypreamble\@@DeclareSymbolFont@m@dropped
640 \@mathfontinfo{Adapting \noexpand\set@mathchar for unicode.}
641 \@mathfontinfo{Adapting \noexpand\set@mathsymbol for unicode.}
642 \@mathfontinfo{Adapting \noexpand\set@mathaccent for unicode.}
643 \@mathfontinfo{Increasing upper bound on

```

```
644 \noexpand\DeclareSymbolFont to 256.}
```

Kernel command to set math characters from keystrokes.

```
645 \def\set@mathchar#1#2#3#4{%
646   \multiply\count\z@ by 16\relax
647   \advance\count\z@\count\tw@
648   \global\Umathcode`#2=\mathchar@type#3+#1+\count\z@\relax}
```

Kernel command to set math characters from control sequences.

```
649 \def\set@mathsymbol#1#2#3#4{%
650   \multiply\count\z@ by 16\relax
651   \advance\count\z@\count\tw@
652   \global\Umathchardef#2=\mathchar@type#3+#1+\count\z@\relax}
```

Kernel command to set accents.

```
653 \def\set@mathaccent#1#2#3#4{%
654   \multiply\count\z@ by 16\relax
655   \advance\count\z@\count\tw@
656   \protected\xdef#2{%
657     \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 XeTeX allow up to 256 math families, but the L^AT_EX 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, which is 256 in XeTeX and LuaTeX. We get a sanitized definition with `\meaning` and `\strip@prefix`, implement the patch by expanding `\M@p@tch@decl@re`, and retokenize the whole thing. A simpler approach, such as calling `\M@p@tch@decl@re` directly on the expansion of `\DeclareSymbolFont`, won't work because of how TeX stores and expands parameter symbols inside macros.

As of November 2022, the L^AT_EX team renamed `\DeclareSymbolFont` to `\DeclareSymbolFont@m@dropped`, and now `\DeclareSymbolFont` is a wrapper around the old version of itself. 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`.

```
658 \ifx\DeclareSymbolFont@m@dropped\@undefined
659   \edef\@tempa{\expandafter
660     \strip@prefix\meaning\DeclareSymbolFont}
661   \def\@tempb{\def\DeclareSymbolFont##1##2##3##4##5}
662 \else
663   \edef\@tempa{\expandafter
```

```

664 \strip@prefix\meaning\DeclareSymbolFont@m@dropped}
665 \def\@tempb{\def\DeclareSymbolFont@m@dropped##1##2##3##4##5}
666 \fi
\M@p@tch@decl@re 667 \def\M@p@tch@decl@re#1<15#2\@nil{#1<\e@mathgroup@top#2}
\M@DecSymDef 668 \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 XeTeX. 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`.

```

669 \ifdefined\directlua
670 \expandafter\@tempb\expandafter{\scantextokens
671 \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 as a result, `\scantokens` can produce an end-of-file error. We can make the command usable by putting a `\noexpand` inside `\everyeof`. In order to prevent the `\edef` from also expanding our retokenized definition of `\DeclareSymbolFont`, we put the definition inside `\unexpanded`.

```

672 \else
673 \begingroup
674 \everyeof{\noexpand}

```

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` before the `\endgroup` and redefine `\DeclareSymbolFont` by calling `\@tempa`.

```

\M@retokenize 675 \edef\M@retokenize{\noexpand\scantokens
676 {\noexpand\unexpanded{\M@DecSymDef}}}
\M@retokenize 677 \edef\M@retokenize{\M@retokenize}
678 \expandafter\endgroup
679 \expandafter\@tempb\expandafter{\M@retokenize}
680 \fi

```

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

```

681 \newbox\surdbox
682 \newcount\M@count
683 \newcount\M@num@localfonts
684 \newcount\M@RuleThicknessFactor
685 \newcount\M@IntegralItalicFactor
686 \newcount\M@SurdVerticalFactor
687 \newcount\M@SurdHorizontalFactor
688 \newmuskip\radicandoffset
689 \newread\M@Charm
690 \newtoks\M@toks
691 \M@count\z@
692 \M@RuleThicknessFactor\@m
693 \M@IntegralItalicFactor=400\relax
694 \M@SurdHorizontalFactor\@m
695 \M@SurdVerticalFactor\@m
696 \radicandoffset=3mu\relax

```

Macro that we use later to store font names for local font changes.

```
\M@localfonts 697 \let\M@localfonts\@empty
```

Necessary booleans and default math font shapes.

```

698 \newif\ifM@upper
699 \newif\ifM@lower
700 \newif\ifM@diacritics
701 \newif\ifM@greekupper
702 \newif\ifM@greeklower
703 \newif\ifM@agreekupper
704 \newif\ifM@agreeklower
705 \newif\ifM@cyrillicupper
706 \newif\ifM@cyrilliclower
707 \newif\ifM@hebrew
708 \newif\ifM@digits
709 \newif\ifM@operator
710 \newif\ifM@symbols
711 \newif\ifM@extsymbols
712 \newif\ifM@delimiters
713 \newif\ifM@radical
714 \newif\ifM@arrows
715 \newif\ifM@bigops
716 \newif\ifM@extbigops
717 \newif\ifM@bb
718 \newif\ifM@cal
719 \newif\ifM@frak

```

```

720 \newif\ifM@bcal
721 \newif\ifM@bfrak
722 \newif\if@optionpresent
723 \newif\if@suboptionpresent
724 \newif\ifM@arg@good
725 \newif\ifM@Decl@reF@mily
726 \newif\ifM@fromCharmFile

```

Default shapes.

```

727 \def\M@uppershape{italic}           % latin upper
728 \def\M@lowershape{italic}           % latin lower
729 \def\M@diacriticsshape{upright}     % diacritics
730 \def\M@greekuppershape{upright}     % greek upper
731 \def\M@greeklowershape{italic}      % greek lower
732 \def\M@agreekuppershape{upright}    % ancient greek upper
733 \def\M@agreeklowershape{italic}     % ancient greek lower
734 \def\M@cyrillicuppershape{upright}  % cyrillic upper
735 \def\M@cyrilliclowershape{italic}   % cyrillic lower
736 \def\M@hebrewshape{upright}        % hebrew
737 \def\M@digitsshape{upright}         % numerals
738 \def\M@operatorshape{upright}       % operator font
739 \def\M@delimitersshape{upright}    % delimiters
740 \def\M@radicalshape{upright}        % surd
741 \def\M@bigopssshape{upright}        % big operators
742 \def\M@extbigopssshape{upright}     % extended big operators
743 \def\M@symbolssshape{upright}       % basic symbols
744 \def\M@extsymbolssshape{upright}    % extended symbols
745 \def\M@arrowssshape{upright}        % arrows
746 \def\M@bbshape{upright}             % blackboard bold
747 \def\M@calshape{upright}            % caligraphic
748 \def\M@frakshape{upright}           % fraktur
749 \def\M@bcalshape{upright}           % bold caligraphic
750 \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.

```

\M@keys 751 \def\M@keys{upper,lower,diacritics,greekupper,%
752   greeklower,agreekupper,agreeklower,cyrillicupper,%
753   cyrilliclower,hebrew,digits,operator,delimiters,%
754   radical,bigops,extbigops,symbols,extsymbols,arrows,%
755   bb,cal,frak,bcal,bfrak}
\M@defaultkeys 756 \def\M@defaultkeys{upper,lower,diacritics,greekupper,%
757   greeklower,digits,operator,symbols}

```

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

```

758 \ifM@adjust@font
\M@defaultkeys 759 \edef\M@defaultkeys{\M@defaultkeys,delimiters,radical,bigops}
760 \fi

```

4 Fontloader

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

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

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

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

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

```
\M@declare@shape 761 \def\M@declare@shape#1#2#3#4#5#6{%
762   \ifcsname TU/#1/#5/#6\endcsname
763   \else
764     \DeclareFontShape{TU}{#1}{#5}{#6}{<->"\@tempbase#2:#3;#4"}{}
765   \fi}
```

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

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

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

```
\M@fill@nfss@shape 766 \def\M@fill@nfss@shapes#1#2#3{%
767   \@for\@i:={#1}{#2;-smcp}{#3}{\mddefault}{\shapedefault},%
768   {#1}{/I}{#2;-smcp}{#3}{\mddefault}{\itdefault},%
769   {#1}{/B}{#2;-smcp}{#3}{\bfdefault}{\shapedefault},%
770   {#1}{/BI}{#2;-smcp}{#3}{\bfdefault}{\itdefault},%
771   {#1}{#2;+smcp}{#3}{\mddefault}{\scdefault},%
772   {#1}{/I}{#2;+smcp}{#3}{\mddefault}{\scdefault\itdefault},%
773   {#1}{/B}{#2;+smcp}{#3}{\bfdefault}{\shapedefault},%
774   {#1}{/BI}{#2;+smcp}{#3}{\bfdefault}{\scdefault\itdefault}%
775   \do{\expandafter\M@declare@shape\@i}}
```

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

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

```
\M@check@nfss@shap 776 \def\M@check@nfss@shapes#1{%
777   \let\@tempb\@empty
778   \let\@tempwarning\@gobble
779   \@for\@i:=\mddefault/\shapedefault,%
780     \mddefault/\itdefault,%
781     \bfdefault/\shapedefault,%
782     \bfdefault/\itdefault\do{%
783     \expandafter\ifx\csname TU/#1/\@i\endcsname\relax
784       \def\@tempwarning{\M@MissingNFSSShapesWarning{#1}}
785       \edef\@tempb{\@tempb, \@i}
786     \fi}
```

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

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

We use `\M@split@colon` and `\M@strip@colon` for parsing the argument of `\mathfont`. If the user calls `\mathfont{⟨name⟩:⟨features⟩}`, we store the name in `\@tempbase` and the features in `\@tempfeatures`. If the user specifies a name only, then `\@tempfeatures` will be empty. Syntactically, we use `\M@strip@colon` to remove a final `:` the same way we remove a final `=` when we parse the optional argument in the next section.

```
\M@split@colon 788 \def\M@split@colon#1:#2\@nil{%
\@tempbase 789   \def\@tempbase{#1}
\@tempfeatures 790   \def\@tempfeatures{#2}}
\M@strip@colon 791 \def\M@strip@colon#1:{#1}
```

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

```
\M@newfont 792 \def\M@newfont#1{%
793   \edef\@tempa{#1}
794   \expandafter\M@split@colon\@tempa:\@nil
```

```

795 \ifx\@tempfeatures\@empty\else
\@tempfeatures 796 \edef\@tempfeatures{\expandafter\M@strip@colon\@tempfeatures}
797 \fi

```

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

```

798 \ifcsname TU+\@tempbase\endcsname % is font family in the nfss?
799 \let\M@f@ntn@me\@tempbase
800 \M@check@nfss@shapes\M@f@ntn@me
801 \ifx\directlua\@undefined % if XeTeX
802 \expandafter\edef
803 \csname\M@f@ntn@me-base\endcsname{\M@f@ntn@me}
804 \let\M@f@ntn@meb@se\M@f@ntn@me
805 \else % if LuaTeX

```

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

```

806 \ifcsname\M@f@ntn@me-base\endcsname % if base-mode version
807 \edef\M@f@ntn@meb@se
808 {\csname\M@f@ntn@me-base\endcsname}
809 \M@check@nfss@shapes\M@f@ntn@meb@se
810 \else
811 \M@NoBaseModeDetectedWarning{\M@f@ntn@me}
812 \edef\M@f@ntn@meb@se{\M@f@ntn@me}
813 \expandafter\edef
814 \csname\M@f@ntn@me-base\endcsname{\M@f@ntn@me}
815 \fi
816 \fi

```

Now save the font families for reference later.

```

817 \expandafter\edef
818 \csname M@fontfamily@\@tempbase\endcsname
819 {\M@f@ntn@me}
820 \expandafter\edef
821 \csname M@fontfamily@base@\@tempbase\endcsname
822 {\M@f@ntn@meb@se}
823 \else

```

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

```

824     \edef\nospace\@tempa{\@tempa}
825     \ifcsname M@fontfamily@\@tempa\endcsname
826       \edef\M@f@ntn@me{\csname M@fontfamily@\@tempa\endcsname}
827       \edef\M@f@ntn@meb@se
828         {\csname M@fontfamily@base@\@tempa\endcsname}
829     \else

```

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

```

830     \edef\nospace\@tempa{\@tempa}
831     \ifcase\M@loader % are we using default font-loader?
832       \let\M@f@ntn@me\@tempa
833       \@mathfontinfo{^^JAdding \@tempbase\space
834         to the nfss!}
835       \wlog{Family name: \M@f@ntn@me}
836       \DeclareFontFamily{TU}{\M@f@ntn@me}{}
837       \M@fill@nfss@shapes{\M@f@ntn@me}{\M@otf@features}
838       {\@tempfeatures}

```

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

```

839     \ifdefined\directlua
840       \edef\M@f@ntn@meb@se{\M@f@ntn@me-base}
841       \@mathfontinfo{Adding \@tempbase\space with mode=base
842         to the nfss!}
843       \wlog{Family name: \M@f@ntn@meb@se}
844       \DeclareFontFamily{TU}{\M@f@ntn@meb@se}{}
845       \M@fill@nfss@shapes{\M@f@ntn@meb@se}{\M@otf@features}
846       {\@tempfeatures;mode=base}
847     \else
848       \edef\M@f@ntn@meb@se{\M@f@ntn@me}
849     \fi
850     \or % are we using fontspec as font-loader?

```

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

```

851      \@mathfontinfo{^^JPassing \@tempbase\space
852      to fontspec for handling!}
853      \csname fontspec_set_family:Nnn\endcsname\M@f@ntn@me
854      {\M@otf@features,\@tempfeatures}{\@tempbase}
855      \ifdefined\directlua
856      \@mathfontinfo{Passing \@tempbase\space
857      with Renderer=Base to fontspec for handling!}
858      \csname fontspec_set_family:Nnn\endcsname\M@f@ntn@meb@se
859      {\M@otf@features,\@tempfeatures,Renderer=Base}
860      {\@tempbase}
861      \else
862      \edef\M@f@ntn@meb@se{\M@f@ntn@me}
863      \fi
864      \fi

```

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

```

865      \expandafter\edef\csname\M@f@ntn@me-base\endcsname
866      {\M@f@ntn@meb@se}
867      \expandafter\edef\csname M@fontfamily@\@tempa\endcsname
868      {\M@f@ntn@me}
869      \expandafter\edef\csname M@fontfamily@base@\@tempa\endcsname
870      {\M@f@ntn@meb@se}
871      \fi
872      \fi}

```

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

```

873 \@onlypreamble\M@declare@shape
874 \@onlypreamble\M@fill@nfss@shapes
875 \@onlypreamble\M@newfont

```

At this point, the font information is stored in the NFSS, but nothing has been loaded. For text fonts, that happens during a call to `\selectfont`, and for math fonts, that happens the first time entering math mode. I've considered forcing some fonts to load now, but I'm hesitant to change `LATEX`'s standard font-loading behavior. On this issue, I plan to leave `mathfont` as is for the foreseeable future. (Which probably means forever because backwards compatibility.)

5 Parse Input

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

```
\M@check@option@va 876 \def\M@check@option@valid#1{%
877   \let\@temperror\M@InvalidOptionError % error by default
878   \@for\@j:=\M@keys\do{%
879     \ifx\@j#1
880       \let\@temperror\@gobble % eliminate error
881       \@optionpresenttrue      % set switch to true
882     \fi}
883   \def\@j{empty} % if option is "empty," we do nothing
884   \ifx\@j#1
885     \let\@temperror\@gobble
886     \@optionpresentfalse
887   \fi
888   \@temperror{#1}}
```

Do the same thing for the suboption.

```
\M@check@suboption 889 \def\M@check@suboption@valid#1{%
890   \let\@temperror\M@InvalidSuboptionError % error by default
891   \@for\@j:=roman,upright,italic\do{%
892     \ifx\@j#1
893       \let\@temperror\@gobble % eliminate error
894       \@suboptionpresenttrue  % set switch to true
895     \fi}
896     \@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 spec-

ifies a suboption, `\@tempb` contains $\langle suboption \rangle =$, and we use `\M@strip@equals` to get rid of the extra `=`. If the user does not specify a suboption, `\@tempb` will be empty.

```
\M@strip@equals 897 \def\M@strip@equals#1={#1}
\M@parse@option 898 \def\M@parse@option#1=#2\@nil{%
899   \@optionpresentfalse    % set switch to false by default
900   \@suboptionpresentfalse % set switch to false by default
901   \def\@temp@opt{#1}      % store option
902   \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.

```
903   \ifx\@temp@opt\@empty
904     \M@MissingOptionError
905   \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 `\egroups` in both branches of `\ifx`.

```
906     \M@check@option@valid\@temp@opt
907     \bgroup\def\@tempa{=}
908     \ifx\@temp@sub\@tempa
909       \egroup % first branch \egroup
910       \M@MissingSuboptionError
911     \else
912       \egroup % second branch \egroup
```

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

```
913     \ifx\@temp@sub\@empty
914     \else
915       \edef\@temp@sub{\expandafter\M@strip@equals\@temp@sub}
916       \M@check@suboption@valid\@temp@sub % is suboption legit?
917     \fi
918   \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.

```
919     \bgroup\def\@tempa{roman}
920     \ifx\@temp@sub\@tempa
921       \egroup % first branch \egroup
```

```

922     \def\@temp@sub{upright}
923   \else
924     \egroup % second branch \egroup
925   \fi
926 \fi}

```

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

```

927 \long\def\edef@nospace#1#2{%
928   \edef#1{#2}%
929   \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`.

```

930 \protected\def\mathfont{\@testopt{\@mathfont}}{\M@defaultkeys}}

```

The internal font-changing command. As of version 2.4, I'm taking out `\restoremathinternals`, so we don't need to check for restored primitives anymore.

```

931 \def\@mathfont[#1]#2{%
932   \M@toks{ }

```

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

```

933   \M@newfont{#2}
934   \ifcsname M@fontid@\M@fontn@me\endcsname % need new \M@count?

```



```

935 \else
936   \expandafter\edef
937   \csname M@fontid@\M@fontn@me\endcsname{\the\M@count}
938   \expandafter\edef
939   \csname M@fontid@\M@fontn@meb@se\endcsname{\the\M@count}
940   \advance\M@count\@ne
941 \fi
942 \edef\@tempa{\csname M@fontid@\M@fontn@me\endcsname}

```

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

```

943 \edef\nospace\@tempb{#1}
944 \@for\@i:=\@tempb\do{\expandafter\M@parse@option\@i=\@nil
945   \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!)

```

946   \expandafter\ifx % next lines are two cs to be compared
947   \csname ifM@\@temp@opt\expandafter\endcsname
948   \csname iftrue\endcsname
949   \M@CharsSetWarning{\@temp@opt}
950 \else

```

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

```

951   \edef\@tempc{\the\M@toks^^J\@temp@opt}
952   \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`.

```

953   \if@suboptionpresent
954   \expandafter\edef
955   \csname M@\@temp@opt shape\endcsname{\@temp@sub}
956 \fi

```

```

957      \edef\@tempc{\the\M@toks\space
958      (\csname M@\@temp@opt shape\endcsname)}
959      \M@toks\expandafter{\@tempc}
960      \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.

```

961      \def\@tempb{upright}
962      \ifx\@tempb\@tempc
963          \let\@tempb\shapedefault
964      \else
965          \let\@tempb\itdefault
966      \fi

```

At this point we have the information we need to declare the symbol font, namely the NFSS family (`\M@f@ntn@me`), series (`\mddefault`), and shape (`\@tempb`). 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.

```

967      \ifcsname symM\@tempc\@tempa\endcsname\else
968          \M@SymbolFontInfo{\@tempbase}{\M@f@ntn@meb@se}
969          {\mddefault/\@tempb}{M\@tempc\@tempa}
970          \DeclareSymbolFont{M\@tempc\@tempa}{TU}
971          {\M@f@ntn@meb@se}{\mddefault}{\@tempb}
972      \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.

```

973      \expandafter\edef
974      \csname M@\@temp@opt @fontinfo\endcsname{\@tempbase}
975      \M@FontChangeInfo{\@temp@opt}{\@tempbase}

```

We have extra information to keep track of when `\@temp@opt` is `bb`, `cal`, `frak`, `bcal`, or `bfrak` because then `mathfont` effectively creates a new local font-change command, and we want to make sure that information gets added to `\M@localfonts`.

```

976      \@tfor\@j:={bb}{cal}{frak}{bcal}{bfrak}\do{%
977      \ifx\@temp@opt\@j
978          \@expandtwoargs\M@addto@localfonts
979          {\expandafter\string
980          \csname math\@temp@opt\endcsname}

```

```

981          {\@tempbase}
982          \@break@tfor
983          \fi}

```

And now the magic happens!

```

984          \csname M@\@temp@opt @set\endcsname % set default font
985          \csname M@\@temp@opt true\endcsname % set switch to true
986          \fi
987          \fi}

```

Display concluding messages for the user.

```

988 \edef\@tempa{\the\M@toks}
989 \ifx\@tempa\@empty
990   \wlog{The \string\mathfont\space command on line
991     \the\inputlineno\space did not change the font
992     for any characters!}
993 \else
994   \wlog{}
995   \typeout{:: mathfont :: Using font \@tempbase\space
996     on line \the\inputlineno.}
997   \wlog{Character classes changed:\the\M@toks}
998 \fi}
999 \@onlypreamble\mathfont
1000 \@onlypreamble\mathfont
1001 \@onlypreamble\mathfont

```

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

```

\setfont 1002 \protected\def\setfont#1{%
1003   \mathfont{#1}
1004   \mathconstantsfont{#1}
1005   \setmathfontcommands{#1}
1006   \let\rmdefault\M@fontn@me}
1007 \@onlypreamble\setfont

```

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

```

\M@SetMathConstant 1008 \let\M@SetMathConstants\relax
\mathconstantsfont 1009 \protected\def\mathconstantsfont{%
1010   \@testopt{\@mathconstantsfont}{upright}}
\@mathconstantsfon 1011 \def\@mathconstantsfont[#1]#2{%

```

```

1012 \edef\nospace\@tempa{#2}
1013 \edef\@tempa{\csname M@fontfamily@base@\@tempa\endcsname}
1014 \expandafter\ifx\@tempa\relax
1015   \M@BadMathConstantsFontError{#2}
1016 \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.

```

1017   \def\@tempb{#1}
1018   \def\@tempc{upright}
1019   \ifx\@tempb\@tempc
\m@th@const@nts@fo 1020     \let\m@th@const@nts@font@sh@pe\shapedefault
1021   \else
1022     \def\@tempc{italic}
1023     \ifx\@tempb\@tempc
\m@th@const@nts@fo 1024       \let\m@th@const@nts@font@sh@pe\itdefault
1025     \else
1026       \M@BadMathConstantsFontTypeError{#1}
1027     \fi
1028   \fi
1029   \ifcsname symM#1\csname M@fontid@\@tempa\endcsname\endcsname
1030   \else
1031     \DeclareSymbolFont{M#1\csname M@fontid@\@tempa\endcsname}
1032       {TU}{\@tempa}{\mddefault}{\m@th@const@nts@font@sh@pe}
1033   \fi
\m@th@const@nts@fo 1034   \let\m@th@const@nts@font\@tempa

```

We come to the tricky problem of making sure to use the correct MathConstants table. LuaTeX automatically initializes all math parameters based on the most recent \textfont, etc. assignment, so we want to tell L^AT_EX to reassign whatever default font we're using to the correct math family whenever we load new math fonts. This is possible, but the implementation is super hacky. When L^AT_EX enters math mode, it checks whether it needs to redo any math family assignments, typically because of a change in font size, and if so, it calls \getanddefine@fonts repeatedly to append \textfont, etc. assignments onto the macro \math@fonts. Usually \math@fonts is empty because this process always happens inside a group, so we can hook into the code by defining \math@font to be \aftergroup<extra code>. In this case, the extra code will be another call to \getanddefine@fonts.

We initialize \M@SetMathConstants to be \relax, and we define it the first time the user calls \mathconstantsfont. When that happens, mathfont

begins by calling `\getanddefine@fonts` inside a group and uses as arguments the upright face of the font corresponding to `#1`. That puts the `\textfont`, `\scriptfont`, and `\scriptscriptfont` assignments corresponding to `#1` inside `\math@fonts`. 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`.

```

\M@SetMathConstant 1035   \protected\def\M@SetMathConstants{%
1036       \begingroup
1037       \escapechar\m@ne
1038       \expandafter\getanddefine@fonts
1039       \csname symM#1%
1040           \csname M@fontid@\m@th@const@nts@font\endcsname
1041       \expandafter
1042       \endcsname % expands to \symMupright<id>
1043       \csname TU/\m@th@const@nts@font
1044           /\seriesdefault
1045           /\m@th@const@nts@font@sh@pe
1046       \endcsname % expands to \TU/<nfss family name>/m/<shape>
1047       \globaldefs\@ne
1048       \expandafter\@gobbletwo\math@fonts % avoid infinite loop
1049       \endgroup}
1050   \fi
1051   \ifM@adjust@font\else
1052       \M@MathConstantsNoAdjustWarning
1053   \fi}
\math@fonts 1054 \def\math@fonts{\aftergroup\M@SetMathConstants}
1055 \@onlypreamble\mathconstantsfont

```

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

7 Local Font Changes

This section deals with local font changes. The `\newmathfontcommand` creates macros that change the font for math alphabet characters and is basically a wrapper around `\DeclareMathAlphabet`. First we code `\M@check@csarg`, which accepts two arguments. The `#1` argument is the user-level command that

called `\M@check@csarg`, which we use for error messaging, and `#2` should be a single control sequence. The way `\M@check@csarg` scans the following tokens is a bit tricky: (1) check the length of the argument by seeing if `\@gobble` eats it completely; and (2) check that the argument is a control sequence. If the user specifies an argument of the form `{...}`, i.e. extra text inside braces, the `\ifcat` will catch it and issue an error. If `\M@check@csarg` likes the input, it sets `\ifM@good@arg` to true, and otherwise, it sets `\ifM@arg@good` to false.

```
\M@check@csarg 1056 \def\M@check@csarg#1#2{%
1057   \expandafter\ifx\expandafter\@nnil\@gobble#2\@nnil % good
1058   \ifcat\relax\noexpand#2 % good
1059   \M@arg@goodtrue
1060   \else % if #2 not a control sequence
1061   \M@MissingControlSequenceError#1{#2}
1062   \M@arg@goodfalse
1063   \fi
1064   \else % if #2 is multiple tokens
1065   \M@DoubleArgError#1{#2}
1066   \M@arg@goodfalse
1067   \fi}
```

The macro `\M@check@specials` accepts a control sequence as its `#1` argument and a font name as its `#2` argument, and it checks whether `#1` is `\mathbb` or a related command. If yes, we assume that the user is using some variant of `\newmathrm` instead of `\mathfont[bb]`, so we do some processing analogous to what we do inside `\mathfont`.

```
\M@check@specials 1068 \def\M@check@specials#1#2{%
1069   \in@#1{\mathbb\mathcal\mathfrak\mathbbcal\mathbbfrak}
1070   \ifin@
```

We set `\escapechar` to `-1` and use `\@gobblefour` to remove the `\math` from the start of `#1`. The string of `\expandafters` hits the `\string` inside `\@tempa`, and then the `\edef` expands the `\@gobblefour`. We are left with just the keyword inside `\@tempa`.

```
1071   \begingroup
1072   \escapechar\m@ne
1073   \expandafter
1074   \endgroup
1075   \expandafter\edef\expandafter\@tempa\expandafter{%
1076   \expandafter\@gobblefour\string#1}
```

Then write a message to the log file and set the corresponding boolean to true.

```
1077   \@mathfontinfo{Interpreting your new macro \string#1\space
1078   as \@tempa\space chars.}
```

```

1079 \mathfontinfo{Setting \expandafter\string
1080 \csname ifM@\@tempa\endcsname\space to true.}
1081 \csname M@\@tempa true\endcsname

```

And store the information to write to the log file `\AtBeginDocument`. The structure of `\M@<keyword>shape` is unusual because we want the macro to gobble “`\space shape`” in `\M@keyword@info@begin`.

```

1082 \expandafter\edef\csname M@\@tempa @fontinfo\endcsname{#2}
1083 \expandafter\edef\csname M@\@tempa shape\endcsname
1084 \space shape{inferred from \string#1}
1085 \fi}

```

Now declare the math alphabet. This macro first checks that its #1 argument is a control sequence using `\M@check@csarg`. If yes, load the #2 argument with `\M@newfont`, call `\DeclareMathAlphabet`, and check whether #1 is `\mathbb` or a related command, and finally add #1 and #2 to the list of local font-change commands.

```

\newmathfontcommand 1086 \protected\def\newmathfontcommand#1#2#3#4{%
1087 \M@check@csarg\newmathfontcommand{#1}
1088 \ifM@arg@good
1089 \M@newfont{#2}
1090 \M@NewFontCommandInfo{#1}{\@tempbase}{\M@font@meb@se}{#3}{#4}
1091 \DeclareMathAlphabet{#1}{TU}{\M@font@meb@se}{#3}{#4}
1092 \M@checkspecials{#1}{\@tempbase}
1093 \M@addto@localfonts{\string#1}{\@tempbase}
1094 \fi}
1095 \@onlypreamble\newmathfontcommand

```

A helper macro that accepts two arguments. Its #1 argument is a control sequence (from `\string`), and the #2 argument is a font name. This macro checks whether #2 is a new font, increments `\M@num@localfonts` if so, and then adds both #1 and #2 to `\M@localfonts`. If `\M@localfonts` is `\@empty`, that means we haven’t added any fonts to the list yet, so in that case, we obviously increase `\M@num@localfonts`. Otherwise we loop through `\M@localfonts`, and if any of them are equal to `\@tempbase`, we do not need to increase `\M@num@localfonts`. We have to use `\@tempb` instead of `\@tempa` here because we use `\M@addto@localfonts` inside `\mathfont` where we need to preserve the definition of `\@tempb`.

```

\M@addto@localfont 1096 \def\M@addto@localfonts#1#2{%
1097 \ifx\M@localfonts\@empty
1098 \advance\M@num@localfonts\@ne
\M@localfonts 1099 \edef\M@localfonts{{#1}{#2}}
1100 \else

```

```

1101 \tempswatrue      % increase by default
1102 \for\@j:=\M@localfonts\do{%
1103   \edef\@tempb{\expandafter\@secondoftwo\@j}
1104   \ifx\@tempbase\@tempb
1105     \tempswafalse % if \@tempbase is in list, don't add
1106   \fi}
1107 \if@tempswa
1108   \advance\M@num@localfonts\@ne
1109 \fi
\M@localfonts 1110 \edef\M@localfonts{\M@localfonts,{#1}{#2}}
1111 \fi}

```

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

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

We feed ##1, ##2, #2, and #3 to `\newmathfontcommand`, and we load ##2 with `\M@newfont`. Each `\newmath<key>` macro will check its first argument using `\M@check@csarg` and then call `\newmathfontcommand` on both of its two arguments. We store the list of `\newmath<key>` commands that we want to define with their series and shape information in `\M@default@newmath@cmds`, and we loop through it with `\@for`.

```

\M@define@newmath@ 1112 \def\M@define@newmath@cmd#1#2#3{%
1113   \protected\def#1##1##2{%
\M@check@csarg 1114     \M@check@csarg{#1}{##1}
1115     \newmathfontcommand{##1}{##2}{#2}{#3}}}
\M@default@newmath 1116 \def\M@default@newmath@cmds{%
1117   \newmathrm{\mddefault}{\shapedefault},%
1118   \newmathit{\mddefault}{\itdefault},%
1119   \newmathbf{\bfdefault}{\shapedefault},%
1120   \newmathbfit{\bfdefault}{\itdefault},%
1121   \newmathsc{\mddefault}{\scdefault},%
1122   \newmathscit{\mddefault}{\scdefault\itdefault},%
1123   \newmathbfsc{\bfdefault}{\scdefault},%
1124   \newmathbfscit{\bfdefault}{\scdefault\itdefault}}
1125 \@for\@i:=\M@default@newmath@cmds\do{%

```



```

1126 \expandafter\M@define@newmath@cmd\@i}
1127 \@onlypreamble\newmathrm
1128 \@onlypreamble\newmathit
1129 \@onlypreamble\newmathbf
1130 \@onlypreamble\newmathbfit
1131 \@onlypreamble\newmathsc
1132 \@onlypreamble\newmathscit
1133 \@onlypreamble\newmathbfsc
1134 \@onlypreamble\newmathbfscit
1135 \@onlypreamble\M@define@newmath@cmd

```

```
\M@default@newmath 1136 \let\M@default@newmath@cmds\relax
```

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

```

\setmathfontcommand 1137 \protected\def\setmathfontcommands#1{%
1138   \newmathrm\mathrm{#1}
1139   \newmathit\mathit{#1}
1140   \newmathbf\mathbf{#1}
1141   \newmathbfit\mathbfit{#1}
1142   \newmathsc\mathsc{#1}
1143   \newmathscit\mathscit{#1}
1144   \newmathbfsc\mathbfsc{#1}
1145   \newmathbfscit\mathbfscit{#1}}
1146 \@onlypreamble\setmathfontcommands

```

8 Miscellaneous

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 checks whether `#1` is a nonnegative integer. Depending on the result, `mathfont` sets `\ifM@arg@good` to true or false. The use of `\afterassignment` is inspired by `\@defaultunits` from the L^AT_EX kernel.

```

1147 \ifM@adjust@font
\M@check@int 1148 \def\M@check@int#1{%
1149   \def\@tempa##1\relax{\def\@tempb{##1}}
1150   \afterassignment\@tempa\count@=0#1\relax
1151   \expandafter\ifx\expandafter\@nnil\@tempb\@nnil
1152   \M@arg@goodtrue
1153   \else
1154   \M@arg@goodfalse
1155   \fi}

```

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

```

1156 \let\@tempa~
1157 \let~\noexpand
1158 \count@\escapechar
1159 \escapechar\m@ne
1160 \@tfor\@i:=\RuleThicknessFactor\IntegralItalicFactor
1161 \SurdHorizontalFactor\SurdVerticalFactor\do{%
1162 \protected\expandafter\edef\@i#1{%
1163 ~\M@check@int{#1}%
1164 ~\ifM@arg@good
1165 ~\global\expandafter
1166 ~\csname
1167 M@\expandafter\string\@i
1168 \endcsname=#1\relax
1169 ~\else
1170 ~\M@BadIntegerError\expandafter~\@i{#1}%
1171 ~\fi}}
1172 \let~\@tempa
1173 \escapechar\count@

```

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

```

1174 \else
1175 \@tfor\@i:=\RuleThicknessFactor\IntegralItalicFactor
1176 \SurdHorizontalFactor\SurdVerticalFactor\CharmLine\CharmFile
1177 \do{%
1178 \protected\expandafter\edef\@i{%
1179 \noexpand\M@NoFontAdjustError\expandafter\noexpand\@i
1180 \noexpand\@gobble}}
1181 \fi

```

These commands should appear in the preamble only.

```

1182 \@onlypreamble\RuleThicknessFactor
1183 \@onlypreamble\IntegralItalicFactor
1184 \@onlypreamble\SurdHorizontalFactor
1185 \@onlypreamble\SurdVerticalFactor
1186 \@onlypreamble\CharmLine
1187 \@onlypreamble\CharmFile

```

As of version 2.4, I'm taking out `\restoremathinternals`. We use the next three macros 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. It feels like it should be expandable, but it isn't. Instead, 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, but this control sequence isn't always accurate. In fact, the LuaTeX manual suggests that it is impossible to have an expandable version of `\mathchoice` that is always correct because of how TeX's math mode algorithm works..

```

1188 \protected\gdef\clap#1{\hb@xt@{z@{\hss#1\hss}}
\stack@flatrel 1189 \protected\def\stack@flatrel#1#2{\expandafter
1190   \st@ck@fl@trel\expandafter#1\@firstofone#2}
\st@ck@fl@trel 1191 \protected\gdef\st@ck@fl@trel#1#2#3{%
1192   {\setbox0\hbox{${\m@th#1#2$}}% contains \mathrel symbol
1193    \setbox1\hbox{${\m@th#1#3$}}% gets raised over \box0
1194    \if\wd0>\wd1\relax
1195      \hb@xt@{\wd0}{%
1196        \hfil
1197        \clap{\raise0.7\ht0\box1}%
1198        \clap{\box0}\hfil}%
1199    \else
1200      \hb@xt@{\wd1}{%
1201        \hfil
1202        \clap{\raise0.7\ht0\box1}%
1203        \clap{\box0}\hfil}%
1204    \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.

```

1205 \ifnum\tracinglostchars<\tw@
1206   \tracinglostchars\tw@
1207 \fi

```

If `mathfont` is adjusting fonts, it also changes the `\Umathcodes` for Latin letters. As a result, math alphabet commands will no longer work for fonts where `mathfont` has not added new Latin letters in the correct encoding slots. The `\M@normal@mathcodes` macro addresses this issue by resetting all Latin letters to their usual encoding slots in math mode. This macro should always appear inside a group, and I am intending for it to be used alongside math alphabet commands from other packages. In particular, we use this macro to make `mathfont` compatible with any `\mathbb` or related commands that don't come from `mathfont`.

```

1208 \ifM@adjust@font
\M@normal@mathcode 1209 \protected\def\M@normal@mathcodes{%
1210     \ifM@upper
1211         \Umathcode`A=7+0+`A\relax
1212         \Umathcode`B=7+0+`B\relax
1213         \Umathcode`C=7+0+`C\relax
1214         \Umathcode`D=7+0+`D\relax
1215         \Umathcode`E=7+0+`E\relax
1216         \Umathcode`F=7+0+`F\relax
1217         \Umathcode`G=7+0+`G\relax
1218         \Umathcode`H=7+0+`H\relax
1219         \Umathcode`I=7+0+`I\relax
1220         \Umathcode`J=7+0+`J\relax
1221         \Umathcode`K=7+0+`K\relax
1222         \Umathcode`L=7+0+`L\relax
1223         \Umathcode`M=7+0+`M\relax
1224         \Umathcode`N=7+0+`N\relax
1225         \Umathcode`O=7+0+`O\relax
1226         \Umathcode`P=7+0+`P\relax
1227         \Umathcode`Q=7+0+`Q\relax
1228         \Umathcode`R=7+0+`R\relax
1229         \Umathcode`S=7+0+`S\relax
1230         \Umathcode`T=7+0+`T\relax
1231         \Umathcode`U=7+0+`U\relax
1232         \Umathcode`V=7+0+`V\relax
1233         \Umathcode`W=7+0+`W\relax
1234         \Umathcode`X=7+0+`X\relax
1235         \Umathcode`Y=7+0+`Y\relax
1236         \Umathcode`Z=7+0+`Z\relax
1237     \fi
1238     \ifM@lower
1239         \Umathcode`a=7+0+`a\relax
1240         \Umathcode`b=7+0+`b\relax
1241         \Umathcode`c=7+0+`c\relax
1242         \Umathcode`d=7+0+`d\relax
1243         \Umathcode`e=7+0+`e\relax
1244         \Umathcode`f=7+0+`f\relax
1245         \Umathcode`g=7+0+`g\relax
1246         \Umathcode`h=7+0+`h\relax
1247         \Umathcode`i=7+0+`i\relax
1248         \Umathcode`j=7+0+`j\relax
1249         \Umathcode`k=7+0+`k\relax

```

```

1250      \Umathcode`l=7+0+`l\relax
1251      \Umathcode`m=7+0+`m\relax
1252      \Umathcode`n=7+0+`n\relax
1253      \Umathcode`o=7+0+`o\relax
1254      \Umathcode`p=7+0+`p\relax
1255      \Umathcode`q=7+0+`q\relax
1256      \Umathcode`r=7+0+`r\relax
1257      \Umathcode`s=7+0+`s\relax
1258      \Umathcode`t=7+0+`t\relax
1259      \Umathcode`u=7+0+`u\relax
1260      \Umathcode`v=7+0+`v\relax
1261      \Umathcode`w=7+0+`w\relax
1262      \Umathcode`x=7+0+`x\relax
1263      \Umathcode`y=7+0+`y\relax
1264      \Umathcode`z=7+0+`z\relax
1265      \fi}

```

We loop through each `bb`, etc. keyword and check whether the corresponding `\math<keyword>` is defined. For convenience, we store `\math<keyword>` inside `\@tempa`. If the definition didn't come from `mathfont`, we wrap a call to `\M@normal@mathcodes` around the macro. This has to happen `\AtBeginDocument` because the user could load a package after `mathfont` that defines `\mathbb`.

```

1266      \AtBeginDocument{\@tfor\@i:={bb}{cal}{frac}{bcal}{bfrac}\do{%
1267        \ifcsname math\@i\endcsname

```

If `\math<keyword>` is defined, check whether the definition came from `mathfont`. Once again, we use `\ifx` with two booleans inside `\csname/\endcsname` pairs because otherwise `TEX` will get confused and think we have an unbalanced boolean inside `\@tfor`.

```

1268      \expandafter\ifx % next two lines are cs to be compared
1269        \csname ifM@\@i\expandafter\endcsname
1270        \csname iffalse\endcsname % is \ifM@<keyword> false?

```

If `mathfont` did not define the control sequence, we have to do things. First an informational message in the log file.

```

1271      \@mathfontinfo{Restoring \string\Umathcode\space values
1272      for \expandafter\string\csname math\@i\endcsname.}

```

Then save `\math<keyword>` as `\@@math<keyword>`, and make `\math<keyword>` into a wrapper around its original definition.

```

1273      \expandafter\let\csname @@math\@i\expandafter\endcsname
1274      \csname math\@i\endcsname
1275      \protected\expandafter\edef\csname math\@i\endcsname#1{%

```

```

1276         \begingroup
1277         \noexpand\M@normal@mathcodes
1278         \expandafter\noexpand\csname @@math\@i\endcsname{#1}
1279         \endgroup}
1280     \fi
1281 \fi}}

```

If `mathfont` is not adjusting fonts, then we do not need to change any `\Umathcodes`.

```

1282 \else
\M@normal@mathcode 1283 \let\M@normal@mathcodes\relax
1284 \fi

```

Now a messages about local font changes. We `\typeout` a message about local font-change commands.

```

1285 \AtBeginDocument{%
1286   \ifcase\M@num@localfonts
1287   \or
1288     \def\@tempa#1#2#3\@nil{#2}
1289     \wlog{}
1290     \typeout{:: mathfont :: Using
1291       \expandafter\@tempa\M@localfonts\@nil\space
1292       for local font changes.}
1293   \else
1294     \wlog{}
1295     \typeout{:: mathfont :: Using \the\M@num@localfonts\space
1296       fonts for local font changes.}
1297   \fi}

```

Write to the log file `\AtBeginDocument` all font changes carried out by `mathfont`. The command `\M@keyword@info@begin` accepts two arguments and prints an informational message about default font changes for keyword options for `\mathfont`. The `#1` argument is a keyword, and the `#2` argument is a number of spaces. The spaces make the messages line up with each other in the log file.

```

\M@keyword@info@be 1298 \def\M@keyword@info@begin#1:#2\@nil{%
1299   \expandafter\ifx % next lines are two cs to be compared
1300     \csname ifM@#1\expandafter\endcsname
1301     \csname iftrue\endcsname
1302     \wlog{#1#2\@spaces\space\space\space Set to
1303       \csname M@#1@fontinfo\endcsname,
1304       \csname M@#1shape\endcsname\space shape}
1305   \else
1306     \wlog{#1#2\@spaces\space\space\space No change}

```

1307 `\fi}`

The macro `\M@local@info@begin` does the same thing except for the local font-change commands. The `#1` argument is a control sequence from `\string`, and the `#2` argument is a font name. We use `\count@` and `\@tfor` to count out the number of characters in `#1`, and if we get to 13, we cut it off and add three dots. We need to use this approach here because the control sequences in `\M@localfonts` have an unknown number of characters. We could have coded up an algorithm like this for `\M@keyword@info@begin` too, but I didn't do it before, and it won't make any practical difference to do it now.

```
\M@local@info@begi 1308 \def\M@local@info@begin#1#2{%
1309   \let\@tempa\@empty
1310   \count@\z@
1311   \@tfor\@i:=#1\do{%
1312     \edef\@tempa{\@tempa\@i}
1313     \advance\count@\@ne
1314     \ifnum\count@=15\relax
1315       \edef\@tempa{\@tempa...}
1316       \advance\count@\thr@@
1317       \@break@tfor
1318   \fi}
```

Now add spaces to `\@tempa` until we get up to 20 characters. At that point, we can write the message to the log file.

```
1319   \@whilenum\count@<20\do{%
1320     \edef\@tempa{\@tempa\space}
1321     \advance\count@\@ne}
1322   \wlog{\@tempa#2}}
```

Now print the messages.

```
1323 \AtBeginDocument{%
1324   \wlog{}
1325   \wlog{*****~J%
1326     * Changes made by mathfont in the preamble *~J%
1327     *****~J}
1328   \wlog{Fonts for character classes:}
1329   \def\@tempa{%    <-- everything should be 14 characters long
1330     upper:\@spaces\@spaces,%
1331     lower:\@spaces\@spaces,%
1332     diacritics:\space\space\space,%
1333     greekupper:\space\space\space,%
1334     greeklower:\space\space\space,%
1335     agreekupper:\space\space,%
1336     agreeklower:\space\space,%
```

```

1337   cyrillicupper:,%
1338   cyrilliclower:,%
1339   hebrew:\@spaces\space\space\space,%
1340   digits:\@spaces\space\space\space,%
1341   operator:\@spaces\space,%
1342   delimiters:\space\space\space,%
1343   radical:\@spaces\space\space,%
1344   bigops:\@spaces\space\space\space,%
1345   extbigops:\@spaces,%
1346   symbols:\@spaces\space\space,%
1347   extsymbols:\space\space\space,%
1348   arrows:\@spaces\space\space\space,%
1349   bb:\@spaces\@spaces\space\space\space,%
1350   cal:\@spaces\@spaces\space\space,%
1351   frac:\@spaces\@spaces\space,%
1352   bcal:\@spaces\@spaces\space,%
1353   bfrac:\@spaces\@spaces}
1354   \@for\@i:=\@tempa\do{%
1355     \expandafter\M@keyword@info@begin\@i\@nil}
1356   \wlog{}
```

And information in the log file about local font-change commands.

```

1357   \wlog{Local font-change commands:}
1358   \ifnum\M@num@localfonts=\z@
1359     \wlog{No local font change commands declared.}
1360   \else
1361     \@for\@j:=\M@localfonts\do{\expandafter\M@local@info@begin\@j}
1362   \fi
1363   \wlog{}}
```

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

```

1364 \ifdefined\XeTeXrevision
1365   \ifM@font@loaded
1366     \AtEndOfPackage{%
1367       \PackageWarningNoLine{mathfont}
1368       {XeTeX detected. It looks like you\MessageBreak
1369       specified a font when you loaded\MessageBreak
1370       mathfont. If you run into problems\MessageBreak
1371       with a font whose name is multiple\MessageBreak
1372       words, try compiling with LuaLaTeX\MessageBreak
1373       or call \string\setfont\space or \string\mathfont\MessageBreak
1374       manually}}}
```



```
1375 \fi
```

```
1376 \fi
```

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

```
1377 \ifM@font@loaded
```

```
1378 \AtEndOfPackage{\setfont\M@font@load}
```

```
1379 \fi
```

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

```
1380 \@onlypreamble\M@upper@set
```

```
1381 \@onlypreamble\M@lower@set
```

```
1382 \@onlypreamble\M@diacritics@set
```

```
1383 \@onlypreamble\M@greekupper@set
```

```
1384 \@onlypreamble\M@greeklower@set
```

```
1385 \@onlypreamble\M@agreekupper@set
```

```
1386 \@onlypreamble\M@agreeklower@set
```

```
1387 \@onlypreamble\M@cyrillicupper@set
```

```
1388 \@onlypreamble\M@cyrilliclower@set
```

```
1389 \@onlypreamble\M@hebrew@set
```

```
1390 \@onlypreamble\M@digits@set
```

```
1391 \@onlypreamble\M@operator@set
```

```
1392 \@onlypreamble\M@delimiters@set
```

```
1393 \@onlypreamble\M@radical@set
```

```
1394 \@onlypreamble\M@bigops@set
```

```
1395 \@onlypreamble\M@extbigops@set
```

```
1396 \@onlypreamble\M@symbols@set
```

```
1397 \@onlypreamble\M@extsymbols@set
```

```
1398 \@onlypreamble\M@arrows@set
```

```
1399 \@onlypreamble\M@bb@set
```

```
1400 \@onlypreamble\M@cal@set
```

```
1401 \@onlypreamble\M@frak@set
```

```
1402 \@onlypreamble\M@bcal@set
```

```
1403 \@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. Put a wrapper around them, and insert the wrapper-function into `luaotfload.patch_font`.

Step 2 happens on the T_EX 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 T_EX 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 T_EX loads a text font, we store character metric information about that glyph as a subtable in `mathfont`. The entries of the subtable describe how to stretch the glyph bounds, scale the glyph itself, or determine math accent placement. For characters of type `u`, we only specify accent placement. For characters of type `a`, which is the upper and lower-case Latin letters, we stretch the bounding box of the glyph horizontally to widen the letters slightly. When we load a text font, we create 52 virtual characters in the Unicode Supplementary Private Use Area-A that typeset the Latin letter glyphs in elongated bounding boxes, and later in `mathfont`, we set the mathcodes of Latin letters to be these virtual characters. For type `e`, we do the same thing except that for each character, we create an ensemble of scaled versions, which we use as a family of large variants.

Here's how to think about the dynamics of our approach. We use character metric information at three different times: pre-processing, interim processing,

Table 2: Fields of Character Subtables in `mathfont`

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

and post-processing. In pre-processing, which we implement in this section, we assemble initial character metric information into entries in `mathfont`. In other words, pre-processing means creating the initial `mathfont` subtables and happens during package loading. Interim processing means the user altering entries in `mathfont` and happens through `\CharmLine` and `\CharmFile`. This can occur at any point in the preamble. In post-processing, which we implement in the next section, `mathfont` extracts information from the current state of the `mathfont` table and uses it to alter a `fontdata` object. Post-processing happens through the `luaotfload.patch_font` callback and occurs once at the point when \TeX loads the font file. As a rule, \LaTeX does not like to load fonts before it uses them, so post-processing typically happens `\AtBeginDocument` in the case of the main text font or whenever the user calls a `\text{}` 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

Table 3: Functions in `mathfont`

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

least nicer than they would otherwise.

```
1404 \ifM@adjust@font
1405 \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.

```
1406 \bgroup
1407   \catcode`\~=0
1408   ~catcode`~\=12
1409   ~@firstofone{
1410 ~egroup
1411 ~def~M@number@assert{"\n%
1412   Package mathfont error: Nonnumeric charm value.\n\n%
1413   I'm having trouble with a character metric.\n%
1414   Your \\CharmLine or \\CharmFile contains \"
1415   .. temp_string .. "\"\n%
1416   which is not a number. Make sure that your\n%
1417   charm information is all integers, floats,\n%
1418   or asterisks separated by commas or spaces.\n"}
1419 ~def~M@index@assert{"\n%
1420   Package mathfont error: Invalid unicode index.\n\n%
1421   The unicode index \"
1422   .. split_string[1] .. "\" is invalid. Make sure\n%
1423   that the first number in your \\CharmLine and in each\n%
1424   line of your \\CharmFile is an integer between 0 and\n%
1425   1,114,111.\n"}
1426 ~def~M@entries@assert{"\n%
1427   Package mathfont error: Charm values too short.\n\n%
1428   Your charm information for U+\" .. index .. " needs more\n%
1429   entries. Right now you have \"
1430   .. number_of_entries .. " entries, and\n%
1431   you need at least \"
1432   .. entries_needed .. ". If you aren't sure what\n%
1433   to do, try adding asterisks to your \\CharmLine\n%
1434   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.

```
\CharmLine 1435 \protected\def\CharmLine#1{%
```

```

1436 \begingroup
1437 \edef\@tempa{#1}
1438 \edef\@tempa{\detokenize\expandafter{\@tempa}}
1439 \@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.

```

1440 \ifin@ % is " in #1?
1441   \ifM@fromCharmFile
1442     \M@ForbiddenCharmFile{"}
1443   \else
1444     \M@ForbiddenCharmLine{"}
1445   \fi
1446 \else
1447   \@expandtwoargs\in@{\@backslashchar}{\@tempa}
1448   \ifin@ % is backslash in #1?
1449     \ifM@fromCharmFile
1450       \M@ForbiddenCharmFile{\@backslashchar}
1451     \else
1452       \M@ForbiddenCharmLine{\@backslashchar}
1453     \fi
1454   \else

```

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

```

1455     \directlua{mathfont:add_to_charm("\@tempa")}
1456   \fi
1457 \fi
1458 \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.

```

1459 \protected\def\CharmFile#1{%
1460   \begingroup
1461   \M@fromCharmFiletrue
1462   \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.

```

1463 \def\@next{%
1464   \read\M@Charm to \@tempa
1465   \CharmLine\@tempa
1466   \ifeof\M@Charm\else % if file has more lines?
1467     \expandafter\@next
1468   \fi}

```

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

```

1469 \@next
1470 \immediate\closein\M@Charm
1471 \M@fromCharmFilefalse
1472 \endgroup}

```

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

```

1473 \directlua{
1474 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 .

```

1475 function mathfont:new_type_a(index, next, data)
1476   self[index] = {}
1477   self[index].type = "a"
1478   self[index].next = next
1479   self[index].left_stretch = data[1] / 1000
1480   self[index].right_stretch = data[2] / 1000
1481   self[index].top_accent_stretch = data[3] / 1000
1482   self[index].bot_accent_stretch = data[4] / 1000
1483 end

```

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

```
1484 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`.

```
1485   local v = \string# next
1486   self[index] = {}
1487   self[index].type = "e"
1488   self[index].smash = smash
1489   self[index].next = next
1490   self[index].total_variants = v
1491   self[index].data = {}
```

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

```
1492   for i = 1, v, 1 do
1493     self[index].data[i] = {}
1494     self[index].data[i].x = data[2*i-1] / 1000
1495     self[index].data[i].y = data[2*i] / 1000
1496   end
1497   self[index].top_accent_stretch = data[2*v+1] / 1000
1498   self[index].bot_accent_stretch = data[2*v+2] / 1000
1499 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.

```
1500 function mathfont:new_type_u(index, data)
1501   self[index] = {}
1502   self[index].type = "u"
1503   self[index].top_accent_stretch = data[1] / 1000
1504   self[index].bot_accent_stretch = data[2] / 1000
1505 end
```

Interim processing. We let the user 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`.

```

1506 function mathfont.parse_charm(charm_input)
1507   local split_string = {}
1508   local charm_string = charm_input
1509   local temp_string = ""
1510   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.

```

1511   while string.find(charm_string, " ") or
1512         string.find(charm_string, ",") do
1513     local length = string.len(charm_string)
1514     local first_space = string.find(charm_string, " ") or length
1515     local first_comma = string.find(charm_string, ",") or length

```

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

```

1516     local sep = first_space
1517     if first_comma < first_space then
1518         sep = first_comma
1519     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`.

```

1520     temp_string = string.sub(charm_string, 1, sep-1)
1521     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.

```

1522     if temp_string \noexpand~= "" then
1523         if tonumber(temp_string) then % if a number, append it
1524             split_string[i] = tonumber(temp_string)
1525             i = i+1
1526         elseif temp_string == "*" then % if asterisk, append it
1527             split_string[i] = temp_string
1528             i = i+1
1529         else % if neither, raise error
1530             error("\M@number@ssert")
1531         end
1532     end
1533 end

```

After iterating 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.

```

1534 temp_string = charm_string
1535 if temp_string \noexpand~="" then
1536     if tonumber(temp_string) then % if a number, append number
1537         split_string[i] = tonumber(temp_string)
1538     elseif temp_string == "*" then % if asterisk, append asterisk
1539         split_string[i] = temp_string
1540     else % if neither, raise error
1541         error(\M@number@assert)
1542     end
1543 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.

```

1544 local index = split_string[1]
1545 if index == "*" then
1546     error(\M@index@assert)
1547 end

```

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

```

1548 local rounded = index - (index \%percentchar 1) % round down
1549 local max = 1114111
1550 assert(index == rounded and
1551         index >= 0         and
1552         index <= max, \M@index@assert)
1553 return split_string
1554 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`.

```

1555 function mathfont:add_to_charm(charm_string)
1556     local charm_metrics = self.parse_charm(charm_string)

```

```

1557 local index = charm_metrics[1]
1558 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`.

```

1559 if not self[index] then
1560     self:new_type_u(index, {0, 0})
1561 end

```

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

```

1562 if self[index].type == "a" then
1563     local entries_needed = 5
1564     assert(number_of_entries >= entries_needed, \M@entries@assert)
1565     if charm_metrics[2] \noexpand~="*" then
1566         self[index].left_stretch = charm_metrics[2] / 1000
1567     end
1568     if charm_metrics[3] \noexpand~="*" then
1569         self[index].right_stretch = charm_metrics[3] / 1000
1570     end
1571     if charm_metrics[4] \noexpand~="*" then
1572         self[index].top_accent_stretch = charm_metrics[4] / 1000
1573     end
1574     if charm_metrics[5] \noexpand~="*" then
1575         self[index].bot_accent_stretch = charm_metrics[5] / 1000
1576     end

```

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

```

1577 elseif self[index].type == "e" then
1578     local tot_variants = self[index].total_variants
1579     local entries_needed = 2 * tot_variants + 3
1580     assert(number_of_entries >= entries_needed, \M@entries@assert)
1581     for i = 1, tot_variants, 1 do
1582         if charm_metrics[2*i] \noexpand~="*" then
1583             self[index].data[i].x = charm_metrics[2*i] / 1000

```

```

1584     end
1585     if charm_metrics[2*i+1] \noexpand~= "*" then
1586         self[index].data[i].y = charm_metrics[2*i+1] / 1000
1587     end
1588 end

```

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

```

1589     if charm_metrics[2 * tot_variants + 2] \noexpand~= "*" then
1590         self[index].top_accent_stretch =
1591             charm_metrics[2 * tot_variants + 2] / 1000
1592     end
1593     if charm_metrics[2 * tot_variants + 3] \noexpand~= "*" then
1594         self[index].bot_accent_stretch =
1595             charm_metrics[2 * tot_variants + 3] / 1000
1596     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.

```

1597 elseif self[index].type == "u" then
1598     local entries_needed = 3
1599     assert(number_of_entries >= entries_needed, \M@entries@assert)
1600     if charm_metrics[2] \noexpand~= "*" then
1601         self[index].top_accent_stretch = charm_metrics[2] / 1000
1602     end
1603     if charm_metrics[3] \noexpand~= "*" then
1604         self[index].bot_accent_stretch = charm_metrics[3] / 1000
1605     end
1606 end
1607 end

```

We end this section with three general-purpose Lua functions. The first function, `make_hex_value`, 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.

```

1608 function mathfont.make_hex_value(integer)
1609     if integer == 0 then
1610         return "0000"
1611     end
1612     if integer == 1 then
1613         return "0001"
1614     end
1615     local hex_digits = "0123456789ABCDEF" % for reference
1616     local hex_string = ""
1617     local curr_val = integer

```

```
1618 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`.

```
1619 local i = 0
1620 while 16^i <= curr_val do
1621     i = i+1
1622 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`.

```
1623 for j = i-1, 0, -1 do
1624     remainder = curr_val \%percentchar (16^j)
1625     curr_val = (curr_val - remainder) / (16^j)
1626     hex_string = hex_string ..
1627         string.sub(hex_digits, curr_val+1, curr_val+1)
1628     curr_val = remainder
1629 end
```

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

```
1630 if \string# hex_string < 4 then
1631     for i = \string# hex_string, 4, 1 do
1632         hex_string = "0" .. hex_string
1633     end
1634 end
1635 return hex_string
1636 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.

```
1637 function mathfont.glyph_info(char)
1638     local glyph_width = char.width or 0
1639     local glyph_height = char.height or 0
1640     local glyph_depth = char.depth or 0
1641     local glyph_italic = char.italic or 0
1642     return glyph_width, glyph_height, glyph_depth, glyph_italic
1643 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.

```

1644 function mathfont:smash_glyph(index, fontdata)
1645   local smash_table = {}
1646   smash_table.width = 0
1647   smash_table.height = 0
1648   smash_table.depth = 0
1649   smash_table.commands = {"char", index}}
1650   return smash_table
1651 end

```

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

```

1652 function mathfont.empty(arg)
1653 end

```

10 Adjust Fonts: Changes

This section contains the Lua functions that actually modify the font during loading. The three functions `set_nomath_false`, `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.

```
1654 function mathfont.make_a_commands(index, offset)
1655   local c_1 = {"right", offset}
1656   local c_2 = {"char", index}
1657   return {c_1, c_2}
1658 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.

```
1659 function mathfont:make_a_table(index, charm_data, fontdata)
1660   local a_table = {}
1661   local char = fontdata.characters[index] or {}
1662   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.

Table 4: Callbacks Created by `mathfont`

Callback Name	Default Behavior
<code>"mathfont.inspect_font"</code>	None
<code>"mathfont.pre_adjust"</code>	None
<code>"mathfont.disable_nomath"</code>	<code>mathfont.set_nomath_false</code>
<code>"mathfont.add_math_constants"</code>	<code>mathfont.math_constants</code>
<code>"mathfont.fix_character_metrics"</code>	<code>mathfont.apply_charm_info</code>
<code>"mathfont.post_adjust"</code>	None

Importantly, these values are additive.

```
1663 local left_stretch = charm_data.left_stretch
1664 local right_stretch = charm_data.right_stretch
1665 local width, height, depth, italic = self.glyph_info(char)
```

Incorporate the italic correction into the character width.

```
1666 width = width + italic
```

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

```
1667 local offset = width * left_stretch
1668 a_table.width = width * (1 + left_stretch + right_stretch)
1669 a_table.height = height
1670 a_table.depth = depth
1671 a_table.italic = italic
1672 a_table.unicode = index
```

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

```
1673 a_table.tounicode = self.make_hex_value(index)
```

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

$$\begin{aligned} \text{base measurement} &= \text{left_stretch} * \text{width} \\ &\quad + 0.5 * (\text{width} - \sigma_1 * \text{height}) + \sigma_1 * \text{height}, \end{aligned}$$

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

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

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

```

1674 local top_base = (0.5 + left_stretch) * width +
1675     0.5 * slant * height
1676 local bot_base = (0.5 + left_stretch) * width -
1677     0.5 * slant * height
1678 local top_accent_shift = charm_data.top_accent_stretch * width
1679 local bot_accent_shift = charm_data.bot_accent_stretch * width
1680 a_table.top_accent = top_base + top_accent_shift
1681 a_table.bot_accent = bot_base + bot_accent_shift

```

Add the commands to the table.

```

1682 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 Lua_T_E_X 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.

```

1683 a_table.mathkern = {}
1684 a_table.mathkern.top_right = {{height = 0, kern = -italic}}
1685 a_table.mathkern.bottom_right = {{height = 0, kern = -italic}}
1686 a_table.mathkern.top_left = {{height = 0, kern = 0}}
1687 a_table.mathkern.bottom_left = {{height = 0, kern = 0}}
1688 return a_table
1689 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.

```

1690 function mathfont:modify_e_base(index, fontdata)
1691 local char = fontdata.characters[index] or {}
1692 local width, height, depth, italic = self.glyph_info(char)
1693 char.width = width + italic

```

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

```
1694 if index == 8730 then
```

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

```
1695     local horizontal_scale =
1696         tex.getcount("M@SurdHorizontalFactor") / 1000
1697     local vertical_scale =
1698         tex.getcount("M@SurdVerticalFactor") / 1000
1699     char.width = horizontal_scale * char.width
1700     char.height = vertical_scale * height
1701 end
```

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

```
1702 if index == 8747 then
1703     local scale_factor =
1704         tex.getcount("M@IntegralItalicFactor") / 1000
1705     char.italic = scale_factor * width
1706 end
1707 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.

```
1708 function mathfont.make_e_commands(index, h_stretch, v_stretch)
1709     local c_1 = {"pdf", "origin",
1710         string.format("q \@percentchar s 0 0 \@percentchar s 0 0 cm",
1711             h_stretch, v_stretch)}
1712     local c_2 = {"char", index}
1713     local c_3 = {"pdf", "origin", "Q"}
1714     return {c_1, c_2, c_3}
1715 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.

```

1716 function mathfont:make_e_table(index, charm_data, fontdata)
1717   local e_table = {}
1718   local char = fontdata.characters[index] or {}
1719   local slant = fontdata.parameters.slant / 65536
1720   local tounicode = self.make_hex_value(index)
1721   local smash_index = charm_data.smash
1722   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.

```

1723   for i = 1, charm_data.total_variants, 1 do
1724     local h_stretch = charm_data.data[i].x
1725     local v_stretch = charm_data.data[i].y
1726     local new_width = width * h_stretch
1727     local new_height = height * v_stretch
1728     local new_depth = depth * v_stretch
1729     local new_italic = italic * h_stretch

```

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

```

1730     e_table[i] = {}
1731     e_table[i].width = new_width
1732     e_table[i].height = new_height
1733     e_table[i].depth = new_depth
1734     e_table[i].italic = new_italic

```

Add the unicode information.

```

1735     e_table[i].unicode = index
1736     e_table[i].tounicode = tounicode

```

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

```

1737     local base_top_accent = 0.5 * new_width +
1738       0.5 * slant * new_height

```

```

1739     local base_bot_accent = 0.5 * new_width -
1740         0.5 * slant * new_height
1741     local top_accent_shift =
1742         charm_data.top_accent_stretch * new_width
1743     local bot_accent_shift =
1744         charm_data.bot_accent_stretch * new_width
1745     e_table[i].top_accent = base_top_accent + top_accent_shift
1746     e_table[i].bot_accent = base_bot_accent + bot_accent_shift

```

Add the commands.

```

1747     e_table[i].commands =
1748         self.make_e_commands(smash_index, h_stretch, v_stretch)

```

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

```

1749     if i < charm_data.total_variants then
1750         e_table[i].next = charm_data.next[i+1]
1751     end
1752 end
1753 return e_table
1754 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.

```

1755 function mathfont:make_u_table(index, charm_data, fontdata)
1756     local u_table = fontdata.characters[index] or {}
1757     local slant = fontdata.parameters.slant / 65536 or 0
1758     local width, height, depth, italic = self.glyph_info(u_table)
1759     local new_width = width + italic
1760     u_table.width = new_width

```

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

```

1761     local base_top_accent = 0.5 * new_width +
1762         0.5 * slant * height
1763     local base_bot_accent = 0.5 * new_width -
1764         0.5 * slant * height
1765     local top_accent_shift =
1766         charm_data.top_accent_stretch * new_width
1767     local bot_accent_shift =
1768         charm_data.bot_accent_stretch * new_width

```

```

1769 u_table.top_accent = base_top_accent + top_accent_shift
1770 u_table.bot_accent = base_bot_accent + bot_accent_shift

```

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

```

1771 u_table.mathkern = {}
1772 u_table.mathkern.top_right = {{height = 0, kern = -italic}}
1773 u_table.mathkern.bottom_right = {{height = 0, kern = -italic}}
1774 u_table.mathkern.top_left = {{height = 0, kern = 0}}
1775 u_table.mathkern.bottom_left = {{height = 0, kern = 0}}
1776 return u_table
1777 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.

```

1778 function mathfont.make_fake_angle(index, smash, fontdata)
1779   local temp = {}
1780   local lparen = fontdata.characters[40] or {}
1781   local lparen_height = lparen.height or 0
1782   local lparen_depth = lparen.depth or 0
1783   local glyph = fontdata.characters[index] or {}
1784   local glyph_height = glyph.height or 0
1785   local base_height = 0.9 * glyph_height
1786   local factor = 0
1787   if glyph_height \noexpand~ = 0 then
1788     factor = (lparen_height + lparen_depth) / base_height
1789   end
1790   local shift = 0.1 * glyph_height * factor + lparen_depth
1791   temp.height = lparen_height
1792   temp.depth = lparen_depth
1793   temp.width = glyph.width or 0
1794   temp.italic = glyph.italic or 0
1795   temp.top_accent = glyph.top_accent or 0.5 * temp.width
1796   temp.bot_accent = glyph.bot_accent or 0.5 * temp.width
1797   temp.commands = {
1798     {"down", shift},
1799     {"pdf", "origin",
1800       string.format("q 1 0 0 \@percentchar s 0 0 cm", factor)},
1801     {"char", smash},
1802     {"pdf", "origin", "Q"},

```

```

1803     {"down", -shift}}
1804   return temp
1805 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.

```

1806 function mathfont.set_nomath_false(fontdata)
1807   fontdata.nomath = false
1808   fontdata.oldmath = false
1809 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.

```

1810 function mathfont.apply_charm_info(fontdata)
1811   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.

```

1812   chars[1044538] = mathfont.smash_glyph(8249, fontdata) % \lguil
1813   chars[1044539] = mathfont.smash_glyph(8250, fontdata) % \rguil
1814   chars[1044540] = mathfont.smash_glyph(171, fontdata) % \llguil
1815   chars[1044541] = mathfont.smash_glyph(187, fontdata) % \rrguil

```

Now add the characters to the font.

```

1816   chars[1044508] = mathfont.make_fake_angle(
1817     8249, 1044538, fontdata)
1818   chars[1044509] = mathfont.make_fake_angle(
1819     8250, 1044539, fontdata)
1820   chars[1044510] = mathfont.make_fake_angle(
1821     171, 1044540, fontdata)
1822   chars[1044511] = mathfont.make_fake_angle(
1823     187, 1044541, fontdata)

```

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

```

1824   if not chars[8711] then
1825     chars[8710] = chars[8710] or {}
1826     chars[1044508] = mathfont.smash_glyph(8710, fontdata)
1827     chars[8711] = {}
1828     chars[8711].width = chars[8710].width or 0

```

```

1829     chars[8711].height = chars[8710].height or 0
1830     chars[8711].depth = chars[8710].depth or 0
1831     chars[8711].italic = chars[8710].italic or 0
1832     chars[8711].top_accent = chars[8710].top_accent or
1833         0.5 * chars[8711].width
1834     chars[8711].bot_accent = chars[8710].bot_accent or
1835         0.5 * chars[8711].width
1836     chars[8711].unicode = 8711
1837     chars[8711].tounicode = mathfont.make_hex_value(8711)
1838     chars[8711].commands = {
1839         {"down", -chars[8711].height},
1840         {"pdf", "origin", "q 1 0 0 -1 0 0 cm"},
1841         {"char", 1044508},
1842         {"pdf", "origin", "Q"},
1843         {"down", chars[8711].height}}
1844     end

```

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

```

1845     for index, info in pairs(mathfont) do
1846         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.

```

1847             if info.type == "a" then
1848                 chars[info.next] =
1849                     mathfont:make_a_table(index, info, fontdata)

```

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

```

1850             elseif info.type == "e" then
1851                 local smash = info.smash
1852                 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.

```

1853                 chars[index].next = info.next[1]
1854                 mathfont:modify_e_base(index, fontdata)
1855                 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`.

```

1856     local variants_table =
1857         mathfont:make_e_table(index, info, fontdata)
1858     for i = 1, info.total_variants, 1 do
1859         chars[info.next[i]] = variants_table[i]
1860     end

```

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

```

1861     elseif info.type == "u" then
1862         chars[index] =
1863             mathfont:make_u_table(index, info, fontdata)
1864     end
1865 end
1866 end
1867 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.

```

1868 function mathfont.math_constants(fontdata)
1869     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.

```

1870     local size = fontdata.size or 0
1871     local ex = fontdata.parameters.x_height or 0
1872     local em = fontdata.parameters.quad or 0
1873     local A_height = 0
1874     local y_depth = 0
1875     if fontdata.characters[65] then
1876         A_height = fontdata.characters[65].height or 0 % A
1877     end
1878     if fontdata.characters[121] then
1879         y_depth = fontdata.characters[121].depth or 0 % y
1880     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.

```
1881 local axis = 0
1882 local rule_thickness = 0
```

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

```
1883 local dim = "FractionRuleThickness"
1884 if not fontdata.MathConstants[dim] then
1885     local scale_factor =
1886         tex.getcount("M@RuleThicknessFactor") / 1000
1887     rule_thickness = (size / 18) * scale_factor
1888     fontdata.MathConstants[dim] = rule_thickness
1889 else
1890     rule_thickness = fontdata.MathConstants[dim]
1891 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.

```
1892 local dim = "AxisHeight"
1893 if fontdata.MathConstants[dim] then
1894     axis = fontdata.MathConstants[dim]
1895 else
1896     if fontdata.characters[45] then
1897         axis = fontdata.characters[45].height - 0.5 * rule_thickness
1898     else
1899         axis = 0.8 * ex
1900     end
1901     fontdata.MathConstants[dim] = axis
1902 end
```

Apart from the axis height and rule thickness, we can group the traditional mathematics `\fontdimen` parameters into three categories: four for large operators, five for fractions, and six for superscripts and subscripts. (OpenType math does not use the fifth large-operator parameter ξ_{13} and the seventh script parameter σ_{14} .) We define variables with the same names as their traditional references from Appendix G in the *TEXBook*. I have taken the design approach

of using twice the rule height as a standard minimum clearance, and I am assuming that script styles are roughly 70% as large as text and display styles. We begin with the parameters for large operators.

The parameter ξ_9 is the minimum clearance between the top of a large operator and the limit above it, and we set it to be twice the rule thickness. Before ensuring that the bottom of the upper limit is at least ξ_9 away from the operator character, \TeX attempts to position the baseline of the limit at ξ_{10} distance above the operator character, and we set ξ_{10} to be slightly larger than ξ_9 . If the upper limit has no descender, \TeX will raise its baseline by ξ_{10} , and if it has a descender, \TeX will position the bottom of the descender to be ξ_9 above the operator, which in practice means it will be higher than limits without descenders. This approach balances the desire for consistency in whitespace with the desire for consistency in baseline height. Similarly, we set the minimum clearance ξ_{11} for the lower limit to be equal to the attempted clearance for the upper limit, and the attempted clearance ξ_{12} for the lower limit will be the minimum clearance plus the average of the `\scriptfont x-height` and `\scriptfont A-height`.

```

1903 local xi_9 = 2 * rule_thickness      % top limit min clearance
1904 local xi_10 = xi_9 + 0.35 * y_depth % bottom limit try placement
1905 local xi_11 = xi_10                  % top limit min clearance
1906 local xi_12 = xi_10 + 0.35 * (A_height + ex) % bottom attempt

```

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.

```

1907 local sigma_8 = axis + 1.5 * rule_thickness + y_depth +
1908     0.1 * A_height
1909 local sigma_9 = (axis + 1.35 * rule_thickness + 0.7 * y_depth +
1910     0.07 * A_height)
1911 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.

```

1912 local sigma_11 = (-axis + 1.5 * rule_thickness +
1913     0.5 * y_depth + 1.1 * A_height)
1914 local sigma_12 = (-axis + 1.35 * rule_thickness +
1915     0.35 * y_depth + 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.

```

1916 local sigma_13 = 0.6 * A_height    % attempted superscript height
1917 local sigma_15 = 0.5 * A_height    % attempt for cramped scripts
1918 local sigma_16 = 1.1 * y_depth     % attempted subscript lower
1919 if sigma_16 < 0.2 * A_height then
1920     sigma_16 = 0.2 * A_height
1921 end
1922 local sigma_17 = sigma_16           % sigma_16 when superscript
1923 local sigma_18 = 0.5 * A_height    % superscript lower for boxed
1924 local sigma_19 = 0.1 * A_height    % subscript lower for boxed

```

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.

```

1925 local dim = "DisplayOperatorMinHeight"
1926 if not fontdata.MathConstants[dim] then
1927     fontdata.MathConstants[dim] = 1.8 * A_height
1928 end
1929 local dim = "FractionDelimiterDisplayStyleSize"
1930 if not fontdata.MathConstants[dim] then
1931     fontdata.MathConstants[dim] = 2 * size
1932 end
1933 local dim = "FractionDelimiterSize"
1934 if not fontdata.MathConstants[dim] then

```

```

1935     fontdata.MathConstants[dim] = 1.3 * size
1936 end
1937 local dim = "FractionDenominatorDisplayStyleShiftDown"
1938 if not fontdata.MathConstants[dim] then
1939     fontdata.MathConstants[dim] = sigma_11
1940 end
1941 local dim = "FractionDenominatorShiftDown"
1942 if not fontdata.MathConstants[dim] then
1943     fontdata.MathConstants[dim] = sigma_12
1944 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 .

```

1945 local dim = "FractionDenominatorDisplayStyleGapMin"
1946 if not fontdata.MathConstants[dim] then
1947     fontdata.MathConstants[dim] = rule_thickness + 0.5 * y_depth
1948 end
1949 local dim = "FractionDenominatorGapMin"
1950 if not fontdata.MathConstants[dim] then
1951     fontdata.MathConstants[dim] = rule_thickness + 0.35 * y_depth
1952 end
1953 local dim = "FractionNumeratorDisplayStyleShiftUp"
1954 if not fontdata.MathConstants[dim] then
1955     fontdata.MathConstants[dim] = sigma_8
1956 end
1957 local dim = "FractionNumeratorShiftUp"
1958 if not fontdata.MathConstants[dim] then
1959     fontdata.MathConstants[dim] = sigma_9
1960 end
1961 local dim = "FractionNumeratorDisplayStyleGapMin"
1962 if not fontdata.MathConstants[dim] then
1963     fontdata.MathConstants[dim] = rule_thickness
1964 end
1965 local dim = "FractionNumeratorGapMin"
1966 if not fontdata.MathConstants[dim] then

```

```

1967     fontdata.MathConstants[dim] = rule_thickness
1968 end

```

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

```

1969 local dim = "SkewedFractionHorizontalGap"
1970 if not fontdata.MathConstants[dim] then
1971     fontdata.MathConstants[dim] = 0.5 * em
1972 end
1973 local dim = "SkewedFractionVerticalGap"
1974 if not fontdata.MathConstants[dim] then
1975     fontdata.MathConstants[dim] = ex
1976 end

```

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

```

1977 local dim = "UpperLimitBaselineRiseMin"
1978 if not fontdata.MathConstants[dim] then
1979     fontdata.MathConstants[dim] = xi_11
1980 end
1981 local dim = "UpperLimitGapMin"
1982 if not fontdata.MathConstants[dim] then
1983     fontdata.MathConstants[dim] = xi_9
1984 end
1985 local dim = "LowerLimitBaselineDropMin"
1986 if not fontdata.MathConstants[dim] then
1987     fontdata.MathConstants[dim] = xi_12
1988 end
1989 local dim = "LowerLimitGapMin"
1990 if not fontdata.MathConstants[dim] then
1991     fontdata.MathConstants[dim] = xi_10
1992 end

```

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

```

1993 local dim = "StretchStackGapBelowMin"
1994 if not fontdata.MathConstants[dim] then
1995     fontdata.MathConstants[dim] = xi_10
1996 end
1997 local dim = "StretchStackTopShiftUp"
1998 if not fontdata.MathConstants[dim] then
1999     fontdata.MathConstants[dim] = xi_11
2000 end
2001 local dim = "StretchStackGapAboveMin"

```

```

2002 if not fontdata.MathConstants[dim] then
2003     fontdata.MathConstants[dim] = xi_9
2004 end
2005 local dim = "StretchStackBottomShiftDown"
2006 if not fontdata.MathConstants[dim] then
2007     fontdata.MathConstants[dim] = xi_12
2008 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.

```

2009 local dim = "OverbarExtraAscender"
2010 if not fontdata.MathConstants[dim] then
2011     fontdata.MathConstants[dim] = rule_thickness
2012 end
2013 local dim = "OverbarRuleThickness"
2014 if not fontdata.MathConstants[dim] then
2015     fontdata.MathConstants[dim] = rule_thickness
2016 end
2017 local dim = "OverbarVerticalGap"
2018 if not fontdata.MathConstants[dim] then
2019     fontdata.MathConstants[dim] = 2 * rule_thickness
2020 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.

```

2021 local dim = "RadicalExtraAscender"
2022 if not fontdata.MathConstants[dim] then
2023     fontdata.MathConstants[dim] = rule_thickness
2024 end
2025 local dim = "RadicalRuleThickness"
2026 if not fontdata.MathConstants[dim] then
2027     fontdata.MathConstants[dim] = rule_thickness
2028 end
2029 local dim = "RadicalDisplayStyleVerticalGap"
2030 if not fontdata.MathConstants[dim] then
2031     fontdata.MathConstants[dim] = 2 * rule_thickness
2032 end
2033 local dim = "RadicalVerticalGap"
2034 if not fontdata.MathConstants[dim] then
2035     fontdata.MathConstants[dim] = rule_thickness

```

2036 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.

```
2037   local dim = "RadicalKernBeforeDegree"
2038   if not fontdata.MathConstants[dim] then
2039     fontdata.MathConstants[dim] = (5/18) * em
2040   end
2041   local dim = "RadicalKernAfterDegree"
2042   if not fontdata.MathConstants[dim] then
2043     fontdata.MathConstants[dim] = (10/18) * em
2044   end
2045   local dim = "RadicalDegreeBottomRaisePercent"
2046   if not fontdata.MathConstants[dim] then
2047     fontdata.MathConstants[dim] = 60
2048   end
```

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

```
2049   local dim = "SpaceAfterScript"
2050   if not fontdata.MathConstants[dim] then
2051     fontdata.MathConstants[dim] = 0.1 * em
2052   end
```

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

```
2053   local dim = "StackBottomDisplayStyleShiftDown"
2054   if not fontdata.MathConstants[dim] then
2055     fontdata.MathConstants[dim] = sigma_11
2056   end
2057   local dim = "StackBottomShiftDown"
2058   if not fontdata.MathConstants[dim] then
2059     fontdata.MathConstants[dim] = sigma_12
2060   end
2061   local dim = "StackTopDisplayStyleShiftUp"
2062   if not fontdata.MathConstants[dim] then
2063     fontdata.MathConstants[dim] = sigma_8
2064   end
2065   local dim = "StackTopShiftUp"
2066   if not fontdata.MathConstants[dim] then
2067     fontdata.MathConstants[dim] = sigma_10
2068   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 clear-

ance for numerators and denominators in fractions. For `\displaystyle`, that gives us

$$\text{rule_thickness} + (2 * \text{rule_thickness}) + (2 * \text{rule_thickness} + 0.5 * y_depth)$$

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

```

2069 local dim = "StackDisplayStyleGapMin"
2070 if not fontdata.MathConstants[dim] then
2071     fontdata.MathConstants[dim] = 5 * rule_thickness +
2072     0.5 * y_depth
2073 end
2074 local dim = "StackGapMin"
2075 if not fontdata.MathConstants[dim] then
2076     fontdata.MathConstants[dim] = 3 * rule_thickness +
2077     0.35 * y_depth
2078 end

```

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

```

2079 local dim = "SubscriptShiftDown"
2080 if not fontdata.MathConstants[dim] then
2081     fontdata.MathConstants[dim] = sigma_16
2082 end
2083 local dim = "SubscriptBaselineDropMin"
2084 if not fontdata.MathConstants[dim] then
2085     fontdata.MathConstants[dim] = sigma_19
2086 end
2087 local dim = "SubscriptShiftDownWithSuperscript"
2088 if not fontdata.MathConstants[dim] then
2089     fontdata.MathConstants[dim] = sigma_17
2090 end

```

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

```

2091 local dim = "SubscriptTopMax"
2092 if not fontdata.MathConstants[dim] then
2093     fontdata.MathConstants[dim] = 0.5 * A_height
2094 end

```

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

```

2095 local dim = "SubSuperscriptGapMin"
2096 if not fontdata.MathConstants[dim] then

```



```

2097     fontdata.MathConstants[dim] = rule_thickness
2098 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.

```

2099 local dim = "SuperscriptBottomMin"
2100 if not fontdata.MathConstants[dim] then
2101     fontdata.MathConstants[dim] = sigma_15 - 0.7 * y_depth
2102 end
2103 local dim = "SuperscriptBaselineDropMax"
2104 if not fontdata.MathConstants[dim] then
2105     fontdata.MathConstants[dim] = sigma_18
2106 end
2107 local dim = "SuperscriptShiftUp"
2108 if not fontdata.MathConstants[dim] then
2109     fontdata.MathConstants[dim] = sigma_13
2110 end
2111 local dim = "SuperscriptShiftUpCramped"
2112 if not fontdata.MathConstants[dim] then
2113     fontdata.MathConstants[dim] = sigma_15
2114 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.

```

2115 local dim = "SuperscriptBottomMaxWithSubscript"
2116 if not fontdata.MathConstants[dim] then
2117     fontdata.MathConstants[dim] = -sigma_16 +
2118         0.7 * A_height + rule_thickness
2119 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.

```

2120 local dim = "UnderbarExtraDescender"
2121 if not fontdata.MathConstants[dim] then

```

```

2122     fontdata.MathConstants[dim] = rule_thickness
2123 end
2124 local dim = "UnderbarRuleThickness"
2125 if not fontdata.MathConstants[dim] then
2126     fontdata.MathConstants[dim] = rule_thickness
2127 end
2128 local dim = "UnderbarVerticalGap"
2129 if not fontdata.MathConstants[dim] then
2130     fontdata.MathConstants[dim] = 2 * rule_thickness
2131 end

```

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

```

2132 local dim = "MinConnectorOverlap"
2133 if not fontdata.MathConstants[dim] then
2134     fontdata.MathConstants[dim] = 0
2135 end
2136 end

```

Time for callbacks! We create six of them.

```

2137 luatexbase.create_callback("mathfont.inspect_font",
2138     "simple", mathfont.empty)
2139 luatexbase.create_callback("mathfont.pre_adjust",
2140     "simple", mathfont.empty)
2141 luatexbase.create_callback("mathfont.disable_nomath",
2142     "simple", mathfont.set_nomath_false)
2143 luatexbase.create_callback("mathfont.add_math_constants",
2144     "simple", mathfont.math_constants)
2145 luatexbase.create_callback("mathfont.fix_character_metrics",
2146     "simple", mathfont.apply_charm_info)
2147 luatexbase.create_callback("mathfont.post_adjust",
2148     "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.

```

2149 function mathfont.info(msg)
2150     texio.write_nl("log", "Package mathfont Info: " .. msg)
2151 end
2152 function mathfont.get_font_name(fontdata)
2153     return fontdata.fullname or
2154         fontdata.psname or
2155         fontdata.name or "<??>"
2156 end

```

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

```

2157 function mathfont.adjust_font(fontdata)
2158   luatexbase.call_callback("mathfont.inspect_font", fontdata)
2159   if fontdata.nomath then
2160     local the_font = mathfont.get_font_name(fontdata)
2161     mathfont.info("Adjusting font " .. the_font .. ".")
2162     luatexbase.call_callback("mathfont.pre_adjust",
2163       fontdata)
2164     luatexbase.call_callback("mathfont.disable_nomath",
2165       fontdata)
2166     luatexbase.call_callback("mathfont.add_math_constants",
2167       fontdata)
2168     luatexbase.call_callback("mathfont.fix_character_metrics",
2169       fontdata)
2170     luatexbase.call_callback("mathfont.post_adjust",
2171       fontdata)
2172   else
2173     mathfont.info("No changes made to " .. the_font .. ".")
2174   end
2175 end

```

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

```

2176 luatexbase.add_to_callback("luaotfload.patch_font",
2177   mathfont.adjust_font, "mathfont.adjust_font")

```

11 Adjust Fonts: Metrics

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

```

2178 mathfont:new_type_a(97, 1044480, {50, 50, -50, 0})      % a
2179 mathfont:new_type_a(98, 1044481, {50, 50, -50, 0})      % b
2180 mathfont:new_type_a(99, 1044482, {50, 50, 0, 0})        % c
2181 mathfont:new_type_a(100, 1044483, {50, -50, -50, 0})     % d
2182 mathfont:new_type_a(101, 1044484, {50, 50, 0, 0})       % e
2183 mathfont:new_type_a(102, 1044485, {200, 0, 0, 0})        % f
2184 mathfont:new_type_a(103, 1044486, {100, 50, -50, 0})     % g

```

```

2185 mathfont:new_type_a(104, 1044487, {50, 0, -50, 0}) % h
2186 mathfont:new_type_a(105, 1044488, {50, 100, -100, 0}) % i
2187 mathfont:new_type_a(106, 1044489, {400, 50, -50, 0}) % j
2188 mathfont:new_type_a(107, 1044490, {50, 50, -100, 0}) % k
2189 mathfont:new_type_a(108, 1044491, {100, 150, -100, 0}) % l
2190 mathfont:new_type_a(109, 1044492, {50, 0, 0, 0}) % m
2191 mathfont:new_type_a(110, 1044493, {50, 0, 0, 0}) % n
2192 mathfont:new_type_a(111, 1044494, {50, 0, 0, 0}) % o
2193 mathfont:new_type_a(112, 1044495, {200, 50, -50, 0}) % p
2194 mathfont:new_type_a(113, 1044496, {50, 0, -50, 0}) % q
2195 mathfont:new_type_a(114, 1044497, {100, 100, -50, 0}) % r
2196 mathfont:new_type_a(115, 1044498, {50, 50, -50, 0}) % s
2197 mathfont:new_type_a(116, 1044499, {50, 50, -50, 0}) % t
2198 mathfont:new_type_a(117, 1044500, {0, 50, 0, 0}) % u
2199 mathfont:new_type_a(118, 1044501, {0, 50, -50, 0}) % v
2200 mathfont:new_type_a(119, 1044502, {0, 50, 0, 0}) % w
2201 mathfont:new_type_a(120, 1044503, {50, 0, -50, 0}) % x
2202 mathfont:new_type_a(121, 1044504, {150, 50, -50, 0}) % y
2203 mathfont:new_type_a(122, 1044505, {100, 50, -100, 0}) % z
2204 mathfont:new_type_a(305, 1044506, {100, 100, -150, 0}) % \imath
2205 mathfont:new_type_a(567, 1044507, {700, 50, -150, 0}) % \jmath

```

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

```

2206 mathfont:new_type_a(65, 1044512, {50, 0, 150, 0}) % A
2207 mathfont:new_type_a(66, 1044513, {50, 0, 0, 0}) % B
2208 mathfont:new_type_a(67, 1044514, {0, 0, 0, 0}) % C
2209 mathfont:new_type_a(68, 1044515, {50, 0, -50, 0}) % D
2210 mathfont:new_type_a(69, 1044516, {50, 0, 0, 0}) % E
2211 mathfont:new_type_a(70, 1044517, {50, 0, 0, 0}) % F
2212 mathfont:new_type_a(71, 1044518, {0, 0, 0, 0}) % G
2213 mathfont:new_type_a(72, 1044519, {50, 0, -50, 0}) % H
2214 mathfont:new_type_a(73, 1044520, {100, 0, 0, 0}) % I
2215 mathfont:new_type_a(74, 1044521, {50, 0, 100, 0}) % J
2216 mathfont:new_type_a(75, 1044522, {50, 0, 0, 0}) % K
2217 mathfont:new_type_a(76, 1044523, {50, 0, -180, 0}) % L
2218 mathfont:new_type_a(77, 1044524, {50, 0, -50, 0}) % M
2219 mathfont:new_type_a(78, 1044525, {50, 0, -50, 0}) % N
2220 mathfont:new_type_a(79, 1044526, {0, 0, 0, 0}) % O
2221 mathfont:new_type_a(80, 1044527, {0, 0, -50, 0}) % P
2222 mathfont:new_type_a(81, 1044528, {0, 50, 0, 0}) % Q
2223 mathfont:new_type_a(82, 1044529, {50, 0, -50, 0}) % R
2224 mathfont:new_type_a(83, 1044530, {0, 0, -50, 0}) % S
2225 mathfont:new_type_a(84, 1044531, {0, 0, -50, 0}) % T

```

```

2226 mathfont:new_type_a(85, 1044532, {0, 0, -50, 0})      % U
2227 mathfont:new_type_a(86, 1044533, {0, 50, 0, 0})      % V
2228 mathfont:new_type_a(87, 1044534, {0, 50, -50, 0})     % W
2229 mathfont:new_type_a(88, 1044535, {50, 0, 0, 0})       % X
2230 mathfont:new_type_a(89, 1044536, {0, 0, -50, 0})       % Y
2231 mathfont:new_type_a(90, 1044537, {50, 0, -50, 0})     % Z

```

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

```

2232 mathfont:new_type_u(945, {0, 0})      % \alpha
2233 mathfont:new_type_u(946, {0, 0})      % \beta
2234 mathfont:new_type_u(947, {-50, 0})    % \gamma
2235 mathfont:new_type_u(948, {0, 0})      % \delta
2236 mathfont:new_type_u(1013, {50, 0})    % \epsilon
2237 mathfont:new_type_u(950, {0, 0})      % \zeta
2238 mathfont:new_type_u(951, {-50, 0})    % \eta
2239 mathfont:new_type_u(952, {0, 0})      % \theta
2240 mathfont:new_type_u(953, {-50, 0})    % \iota
2241 mathfont:new_type_u(954, {0, 0})      % \kappa
2242 mathfont:new_type_u(955, {-150, 0})   % \lambda
2243 mathfont:new_type_u(956, {0, 0})      % \mu
2244 mathfont:new_type_u(957, {-50, 0})    % \nu
2245 mathfont:new_type_u(958, {0, 0})      % \xi
2246 mathfont:new_type_u(959, {0, 0})      % \omicron
2247 mathfont:new_type_u(960, {-100, 0})   % \pi
2248 mathfont:new_type_u(961, {-50, 0})    % \rho
2249 mathfont:new_type_u(963, {-100, 0})   % \sigma
2250 mathfont:new_type_u(964, {-100, 0})   % \tau
2251 mathfont:new_type_u(965, {-50, 0})    % \upsilon
2252 mathfont:new_type_u(981, {0, 0})      % \phi
2253 mathfont:new_type_u(967, {-50, 0})    % \chi
2254 mathfont:new_type_u(968, {-50, 0})    % \psi
2255 mathfont:new_type_u(969, {0, 0})      % \omega
2256 mathfont:new_type_u(976, {0, 0})      % \varbeta
2257 mathfont:new_type_u(949, {-50, 0})    % \varepsilon
2258 mathfont:new_type_u(977, {50, 0})     % \vartheta
2259 mathfont:new_type_u(1009, {-50, 0})   % \varrho
2260 mathfont:new_type_u(962, {-50, 0})    % \varsigma
2261 mathfont:new_type_u(966, {0, 0})      % \varphi

```

Upper-case Greek characters. Same as previously.

```

2262 mathfont:new_type_u(913, {0, 0})      % \Alpha

```

```

2263 mathfont:new_type_u(914, {0, 0})    % \Beta
2264 mathfont:new_type_u(915, {0, 0})    % \Gamma
2265 mathfont:new_type_u(916, {0, 0})    % \Delta
2266 mathfont:new_type_u(917, {0, 0})    % \Epsilon
2267 mathfont:new_type_u(918, {0, 0})    % \Zeta
2268 mathfont:new_type_u(919, {0, 0})    % \Eta
2269 mathfont:new_type_u(920, {0, 0})    % \Theta
2270 mathfont:new_type_u(921, {0, 0})    % \Iota
2271 mathfont:new_type_u(922, {0, 0})    % \Kappa
2272 mathfont:new_type_u(923, {0, 0})    % \Lambda
2273 mathfont:new_type_u(924, {0, 0})    % \Mu
2274 mathfont:new_type_u(925, {0, 0})    % \Nu
2275 mathfont:new_type_u(926, {0, 0})    % \Xi
2276 mathfont:new_type_u(927, {0, 0})    % \Omicron
2277 mathfont:new_type_u(928, {0, 0})    % \Pi
2278 mathfont:new_type_u(929, {0, 0})    % \Rho
2279 mathfont:new_type_u(931, {0, 0})    % \Sigma
2280 mathfont:new_type_u(932, {0, 0})    % \Tau
2281 mathfont:new_type_u(933, {0, 0})    % \Upsilon
2282 mathfont:new_type_u(934, {0, 0})    % \Phi
2283 mathfont:new_type_u(935, {0, 0})    % \Chi
2284 mathfont:new_type_u(936, {0, 0})    % \Psi
2285 mathfont:new_type_u(937, {0, 0})    % \Omega
2286 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`.

```

2287 local delim_glyphs = {40, % (
2288   41,      % )
2289   47,      % /
2290   91,      % [
2291   92,      % backslash
2292   93,      % ]
2293  123,      % {
2294  125,      % }
2295  8249,     % \lguil
2296  8250,     % \rguil
2297   171,     % \llguil
2298   187,     % \rrguil

```

```

2299 1044508, % \fakelangle
2300 1044509, % \fakerrangle
2301 1044510, % \fakellangle
2302 1044511} % \fakerrangle
2303 local big_op_glyphs = {33, % !
2304 35, % #
2305 36, % $
2306 37, % %
2307 38, % &
2308 43, % +
2309 63, % ?
2310 64, % @
2311 167, % \S
2312 215, % \times
2313 247, % \div
2314 8719, % \prod
2315 8721, % \sum
2316 8720, % \coprod
2317 8897, % \bigvee
2318 8896, % \bigwedge
2319 8899, % \bigcup
2320 8898, % \bigcap
2321 10753, % \bigoplus
2322 10754, % \bigotimes
2323 10752, % \bigodot
2324 10757, % \bigsqcap
2325 10758} % \bigsqcup
2326 local vert_glyphs = {124, 8730} % | and \surd
2327 local int_glyphs = {8747, % \intop
2328 8748, % \iint
2329 8749, % \iiint
2330 8750, % \oint
2331 8751, % \oiint
2332 8752} % \oiiint

```

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

```
2333 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.

```

2334 local delim_scale = {}
2335 local big_op_scale = {}

```

```
2336 local vert_scale = {}
```

```
2337 local int_scale = {}
```

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

```
2338 for i = 1, 15, 1 do
```

```
2339   delim_scale[2*i-1] = 1000 + 100*i % delimiters - horizontal
```

```
2340   delim_scale[2*i] = 1000 + 500*i   % delimiters - vertical
```

```
2341   vert_scale[2*i-1] = 1000
```

```
2342   vert_scale[2*i] = 1000 + 500*i     % vertically scaled chars
```

```
2343   big_op_scale[2*i-1] = 1000 + 100*i % big operators - horizontal
```

```
2344   big_op_scale[2*i] = 1000 + 100*i   % big operators - vertical
```

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

```
2345   int_scale[2*i-1] = 1000 + 500*i     % integral sign - horizontal
```

```
2346   int_scale[2*i] = 1000 + 1500*i     % integral sign - vertical
```

```
2347 end
```

We do not modify accent placement.

```
2348 delim_scale[31] = 0
```

```
2349 delim_scale[32] = 0
```

```
2350 big_op_scale[31] = 0
```

```
2351 big_op_scale[32] = 0
```

```
2352 vert_scale[31] = 0
```

```
2353 vert_scale[32] = 0
```

```
2354 int_scale[31] = 0
```

```
2355 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.

```
2356 function mathfont:add_extensible_variants(first_smash, glyph_list,
```



```

2357     scale_list)
2358   local variants = (\string# scale_list - 2) / 2
2359   local curr_smash = first_smash
2360   for i = 1, \string# glyph_list, 1 do
2361     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`.

```

2362     local curr_slots = {}
2363     for j = 1, variants, 1 do
2364       curr_slots[j] = curr_smash + j
2365     end

```

Add the charm information and increment `smash`.

```

2366     self:new_type_e(curr_char, curr_smash, curr_slots, scale_list)
2367     smash = smash + variants + 1
2368     curr_smash = smash
2369   end
2370 end

```

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

```

2371 mathfont:add_extensible_variants(smash, delim_glyphs, delim_scale)
2372 mathfont:add_extensible_variants(smash, big_op_glyphs, big_op_scale)
2373 mathfont:add_extensible_variants(smash, vert_glyphs, vert_scale)
2374 mathfont:add_extensible_variants(smash, int_glyphs, int_scale)

```

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

```

2375 }
2376 \fi % matches previous \ifM@adjust@font

```

12 Unicode Hex Values

We make aliases for `\DeclareMathAccent` and `\DeclareMathSymbol` to save space.

```

2377 \let\@DMA\DeclareMathAccent
2378 \let\@DMS\DeclareMathSymbol
2379 \@onlypreamble\@DMA
2380 \@onlypreamble\@DMS

```

Set upper-case Latin characters. We use an `\edef` for `\M@upper@font` because every expansion now will save L^AT_EX twenty-six expansions later when it evaluates each `\DeclareMathSymbol`. If the user has enabled Lua font adjustments,

we set the math codes to be the large values from the Supplemental Private Use Area-A.

```

2381 \ifM@adjust@font
\M@upper@set 2382 \def\M@upper@set{%
2383 \edef\M@upper@font{M\M@uppershape\@tempa}
2384 \@DMS{A}{\mathalpha}{\M@upper@font}{1044512}
2385 \@DMS{B}{\mathalpha}{\M@upper@font}{1044513}
2386 \@DMS{C}{\mathalpha}{\M@upper@font}{1044514}
2387 \@DMS{D}{\mathalpha}{\M@upper@font}{1044515}
2388 \@DMS{E}{\mathalpha}{\M@upper@font}{1044516}
2389 \@DMS{F}{\mathalpha}{\M@upper@font}{1044517}
2390 \@DMS{G}{\mathalpha}{\M@upper@font}{1044518}
2391 \@DMS{H}{\mathalpha}{\M@upper@font}{1044519}
2392 \@DMS{I}{\mathalpha}{\M@upper@font}{1044520}
2393 \@DMS{J}{\mathalpha}{\M@upper@font}{1044521}
2394 \@DMS{K}{\mathalpha}{\M@upper@font}{1044522}
2395 \@DMS{L}{\mathalpha}{\M@upper@font}{1044523}
2396 \@DMS{M}{\mathalpha}{\M@upper@font}{1044524}
2397 \@DMS{N}{\mathalpha}{\M@upper@font}{1044525}
2398 \@DMS{O}{\mathalpha}{\M@upper@font}{1044526}
2399 \@DMS{P}{\mathalpha}{\M@upper@font}{1044527}
2400 \@DMS{Q}{\mathalpha}{\M@upper@font}{1044528}
2401 \@DMS{R}{\mathalpha}{\M@upper@font}{1044529}
2402 \@DMS{S}{\mathalpha}{\M@upper@font}{1044530}
2403 \@DMS{T}{\mathalpha}{\M@upper@font}{1044531}
2404 \@DMS{U}{\mathalpha}{\M@upper@font}{1044532}
2405 \@DMS{V}{\mathalpha}{\M@upper@font}{1044533}
2406 \@DMS{W}{\mathalpha}{\M@upper@font}{1044534}
2407 \@DMS{X}{\mathalpha}{\M@upper@font}{1044535}
2408 \@DMS{Y}{\mathalpha}{\M@upper@font}{1044536}
2409 \@DMS{Z}{\mathalpha}{\M@upper@font}{1044537}}
2410 \else
\M@upper@set 2411 \def\M@upper@set{%
2412 \edef\M@upper@font{M\M@uppershape\@tempa}
2413 \@DMS{A}{\mathalpha}{\M@upper@font}{`A}
2414 \@DMS{B}{\mathalpha}{\M@upper@font}{`B}
2415 \@DMS{C}{\mathalpha}{\M@upper@font}{`C}
2416 \@DMS{D}{\mathalpha}{\M@upper@font}{`D}
2417 \@DMS{E}{\mathalpha}{\M@upper@font}{`E}
2418 \@DMS{F}{\mathalpha}{\M@upper@font}{`F}
2419 \@DMS{G}{\mathalpha}{\M@upper@font}{`G}
2420 \@DMS{H}{\mathalpha}{\M@upper@font}{`H}

```

```

2421 \@DMS{I}{\mathalpha}{\M@upper@font}{`I}
2422 \@DMS{J}{\mathalpha}{\M@upper@font}{`J}
2423 \@DMS{K}{\mathalpha}{\M@upper@font}{`K}
2424 \@DMS{L}{\mathalpha}{\M@upper@font}{`L}
2425 \@DMS{M}{\mathalpha}{\M@upper@font}{`M}
2426 \@DMS{N}{\mathalpha}{\M@upper@font}{`N}
2427 \@DMS{O}{\mathalpha}{\M@upper@font}{`O}
2428 \@DMS{P}{\mathalpha}{\M@upper@font}{`P}
2429 \@DMS{Q}{\mathalpha}{\M@upper@font}{`Q}
2430 \@DMS{R}{\mathalpha}{\M@upper@font}{`R}
2431 \@DMS{S}{\mathalpha}{\M@upper@font}{`S}
2432 \@DMS{T}{\mathalpha}{\M@upper@font}{`T}
2433 \@DMS{U}{\mathalpha}{\M@upper@font}{`U}
2434 \@DMS{V}{\mathalpha}{\M@upper@font}{`V}
2435 \@DMS{W}{\mathalpha}{\M@upper@font}{`W}
2436 \@DMS{X}{\mathalpha}{\M@upper@font}{`X}
2437 \@DMS{Y}{\mathalpha}{\M@upper@font}{`Y}
2438 \@DMS{Z}{\mathalpha}{\M@upper@font}{`Z}}
2439 \fi

```

Set lower-case Latin characters.

```

2440 \ifM@adjust@font
\M@lower@set 2441 \def\M@lower@set{%
2442 \edef\M@lower@font{M\M@lowershape\@tempa}
2443 \@DMS{a}{\mathalpha}{\M@lower@font}{1044480}
2444 \@DMS{b}{\mathalpha}{\M@lower@font}{1044481}
2445 \@DMS{c}{\mathalpha}{\M@lower@font}{1044482}
2446 \@DMS{d}{\mathalpha}{\M@lower@font}{1044483}
2447 \@DMS{e}{\mathalpha}{\M@lower@font}{1044484}
2448 \@DMS{f}{\mathalpha}{\M@lower@font}{1044485}
2449 \@DMS{g}{\mathalpha}{\M@lower@font}{1044486}
2450 \@DMS{h}{\mathalpha}{\M@lower@font}{1044487}
2451 \@DMS{i}{\mathalpha}{\M@lower@font}{1044488}
2452 \@DMS{j}{\mathalpha}{\M@lower@font}{1044489}
2453 \@DMS{k}{\mathalpha}{\M@lower@font}{1044490}
2454 \@DMS{l}{\mathalpha}{\M@lower@font}{1044491}
2455 \@DMS{m}{\mathalpha}{\M@lower@font}{1044492}
2456 \@DMS{n}{\mathalpha}{\M@lower@font}{1044493}
2457 \@DMS{o}{\mathalpha}{\M@lower@font}{1044494}
2458 \@DMS{p}{\mathalpha}{\M@lower@font}{1044495}
2459 \@DMS{q}{\mathalpha}{\M@lower@font}{1044496}
2460 \@DMS{r}{\mathalpha}{\M@lower@font}{1044497}
2461 \@DMS{s}{\mathalpha}{\M@lower@font}{1044498}

```

```

2462 \DMS{t}{\mathalpha}{\M@lower@font}{1044499}
2463 \DMS{u}{\mathalpha}{\M@lower@font}{1044500}
2464 \DMS{v}{\mathalpha}{\M@lower@font}{1044501}
2465 \DMS{w}{\mathalpha}{\M@lower@font}{1044502}
2466 \DMS{x}{\mathalpha}{\M@lower@font}{1044503}
2467 \DMS{y}{\mathalpha}{\M@lower@font}{1044504}
2468 \DMS{z}{\mathalpha}{\M@lower@font}{1044505}
2469 \DMS{\imath}{\mathalpha}{\M@lower@font}{1044506}
2470 \DMS{\jmath}{\mathalpha}{\M@lower@font}{1044507}
2471 \DMS{\hbar}{\mathord}{\M@lower@font}{127}}
2472 \else
\M@lower@set 2473 \def\M@lower@set{%
2474 \edef\M@lower@font{M\M@lower@shape\@tempa}
2475 \DMS{a}{\mathalpha}{\M@lower@font}{`a}
2476 \DMS{b}{\mathalpha}{\M@lower@font}{`b}
2477 \DMS{c}{\mathalpha}{\M@lower@font}{`c}
2478 \DMS{d}{\mathalpha}{\M@lower@font}{`d}
2479 \DMS{e}{\mathalpha}{\M@lower@font}{`e}
2480 \DMS{f}{\mathalpha}{\M@lower@font}{`f}
2481 \DMS{g}{\mathalpha}{\M@lower@font}{`g}
2482 \DMS{h}{\mathalpha}{\M@lower@font}{`h}
2483 \DMS{i}{\mathalpha}{\M@lower@font}{`i}
2484 \DMS{j}{\mathalpha}{\M@lower@font}{`j}
2485 \DMS{k}{\mathalpha}{\M@lower@font}{`k}
2486 \DMS{l}{\mathalpha}{\M@lower@font}{`l}
2487 \DMS{m}{\mathalpha}{\M@lower@font}{`m}
2488 \DMS{n}{\mathalpha}{\M@lower@font}{`n}
2489 \DMS{o}{\mathalpha}{\M@lower@font}{`o}
2490 \DMS{p}{\mathalpha}{\M@lower@font}{`p}
2491 \DMS{q}{\mathalpha}{\M@lower@font}{`q}
2492 \DMS{r}{\mathalpha}{\M@lower@font}{`r}
2493 \DMS{s}{\mathalpha}{\M@lower@font}{`s}
2494 \DMS{t}{\mathalpha}{\M@lower@font}{`t}
2495 \DMS{u}{\mathalpha}{\M@lower@font}{`u}
2496 \DMS{v}{\mathalpha}{\M@lower@font}{`v}
2497 \DMS{w}{\mathalpha}{\M@lower@font}{`w}
2498 \DMS{x}{\mathalpha}{\M@lower@font}{`x}
2499 \DMS{y}{\mathalpha}{\M@lower@font}{`y}
2500 \DMS{z}{\mathalpha}{\M@lower@font}{`z}
2501 \DMS{\imath}{\mathalpha}{\M@lower@font}{131}
2502 \DMS{\jmath}{\mathalpha}{\M@lower@font}{237}
2503 \DMS{\hbar}{\mathord}{\M@lower@font}{127}}

```

2504 \fi

Set diacritics.

```
\M@diacritics@set 2505 \def\M@diacritics@set{%
2506   \edef\M@diacritics@font{M\M@diacriticssshape\@tempa}
2507   \@DMA{\acute}    {\mathalpha}{\M@diacritics@font}{B4}
2508   \@DMA{\aacute}   {\mathalpha}{\M@diacritics@font}{2DD}
2509   \@DMA{\dot}      {\mathalpha}{\M@diacritics@font}{2D9}
2510   \@DMA{\ddot}     {\mathalpha}{\M@diacritics@font}{A8}
2511   \@DMA{\grave}    {\mathalpha}{\M@diacritics@font}{60}
2512   \@DMA{\breve}    {\mathalpha}{\M@diacritics@font}{2D8}
2513   \@DMA{\hat}      {\mathalpha}{\M@diacritics@font}{2C6}
2514   \@DMA{\check}    {\mathalpha}{\M@diacritics@font}{2C7}
2515   \@DMA{\bar}      {\mathalpha}{\M@diacritics@font}{2C9}
2516   \@DMA{\mathring}{\mathalpha}{\M@diacritics@font}{2DA}
2517   \@DMA{\tilde}    {\mathalpha}{\M@diacritics@font}{2DC}}
```

Set capital Greek characters. We undefine \Chi because math-operator defines it to be the hyperbolic cosine integral function.

```
\M@greekupper@set 2518 \def\M@greekupper@set{%
2519   \edef\M@greekupper@font{M\M@greekuppersshape\@tempa}
\Chi 2520   \let\Chi\@undefined
2521   \@DMS{\Alpha}    {\mathalpha}{\M@greekupper@font}{391}
2522   \@DMS{\Beta}     {\mathalpha}{\M@greekupper@font}{392}
2523   \@DMS{\Gamma}    {\mathalpha}{\M@greekupper@font}{393}
2524   \@DMS{\Delta}    {\mathalpha}{\M@greekupper@font}{394}
2525   \@DMS{\Epsilon}  {\mathalpha}{\M@greekupper@font}{395}
2526   \@DMS{\Zeta}     {\mathalpha}{\M@greekupper@font}{396}
2527   \@DMS{\Eta}      {\mathalpha}{\M@greekupper@font}{397}
2528   \@DMS{\Theta}    {\mathalpha}{\M@greekupper@font}{398}
2529   \@DMS{\Iota}     {\mathalpha}{\M@greekupper@font}{399}
2530   \@DMS{\Kappa}    {\mathalpha}{\M@greekupper@font}{39A}
2531   \@DMS{\Lambda}   {\mathalpha}{\M@greekupper@font}{39B}
2532   \@DMS{\Mu}       {\mathalpha}{\M@greekupper@font}{39C}
2533   \@DMS{\Nu}       {\mathalpha}{\M@greekupper@font}{39D}
2534   \@DMS{\Xi}       {\mathalpha}{\M@greekupper@font}{39E}
2535   \@DMS{\Omicron}  {\mathalpha}{\M@greekupper@font}{39F}
2536   \@DMS{\Pi}       {\mathalpha}{\M@greekupper@font}{3A0}
2537   \@DMS{\Rho}      {\mathalpha}{\M@greekupper@font}{3A1}
2538   \@DMS{\Sigma}    {\mathalpha}{\M@greekupper@font}{3A3}
2539   \@DMS{\Tau}      {\mathalpha}{\M@greekupper@font}{3A4}
2540   \@DMS{\Upsilon}  {\mathalpha}{\M@greekupper@font}{3A5}
2541   \@DMS{\Phi}      {\mathalpha}{\M@greekupper@font}{3A6}}
```

```

2542 \@DMS{\Chi}      {\mathalpha}{\M@greekupper@font}{\ "3A7}
2543 \@DMS{\Psi}      {\mathalpha}{\M@greekupper@font}{\ "3A8}
2544 \@DMS{\Omega}     {\mathalpha}{\M@greekupper@font}{\ "3A9}
2545 \@DMS{\varTheta} {\mathalpha}{\M@greekupper@font}{\ "3F4}

```

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

```

2546 \ifM@adjust@font
2547   \ifM@symbols\else
2548     \@DMS{\increment}
2549     {\mathord}{\M@greekupper@font}{\ "2206}
2550   \@DMS{\nabla}
2551   {\mathord}{\M@greekupper@font}{\ "2207}
2552 \fi
2553 \else
2554   \ifM@symbols\else
2555     \@DMS{\increment}
2556     {\mathord}{\M@greekupper@font}{\ "2206}
2557 \fi
2558   \ifM@extsymbols\else
2559     \@DMS{\nabla}
2560     {\mathord}{\M@greekupper@font}{\ "2207}
2561 \fi
2562 \fi}

```

Set minuscule Greek characters.

```

\M@greeklower@set 2563 \def\M@greeklower@set{%
2564   \edef\M@greeklower@font{M\M@greeklower@shape\@tempa}
2565   \@DMS{\alpha}      {\mathalpha}{\M@greeklower@font}{\ "3B1}
2566   \@DMS{\beta}       {\mathalpha}{\M@greeklower@font}{\ "3B2}
2567   \@DMS{\gamma}      {\mathalpha}{\M@greeklower@font}{\ "3B3}
2568   \@DMS{\delta}      {\mathalpha}{\M@greeklower@font}{\ "3B4}
2569   \@DMS{\epsilon}    {\mathalpha}{\M@greeklower@font}{\ "3B5}
2570   \@DMS{\zeta}       {\mathalpha}{\M@greeklower@font}{\ "3B6}
2571   \@DMS{\eta}        {\mathalpha}{\M@greeklower@font}{\ "3B7}
2572   \@DMS{\theta}      {\mathalpha}{\M@greeklower@font}{\ "3B8}
2573   \@DMS{\iota}       {\mathalpha}{\M@greeklower@font}{\ "3B9}
2574   \@DMS{\kappa}      {\mathalpha}{\M@greeklower@font}{\ "3BA}
2575   \@DMS{\lambda}     {\mathalpha}{\M@greeklower@font}{\ "3BB}
2576   \@DMS{\mu}         {\mathalpha}{\M@greeklower@font}{\ "3BC}
2577   \@DMS{\nu}         {\mathalpha}{\M@greeklower@font}{\ "3BD}
2578   \@DMS{\xi}         {\mathalpha}{\M@greeklower@font}{\ "3BE}
2579   \@DMS{\omicron}   {\mathalpha}{\M@greeklower@font}{\ "3BF}

```

```

2580 \DMS{\pi}          {\mathalpha}{\M@greeklower@font}{\3C0}
2581 \DMS{\rho}        {\mathalpha}{\M@greeklower@font}{\3C1}
2582 \DMS{\sigma}       {\mathalpha}{\M@greeklower@font}{\3C3}
2583 \DMS{\tau}         {\mathalpha}{\M@greeklower@font}{\3C4}
2584 \DMS{\upsilon}     {\mathalpha}{\M@greeklower@font}{\3C5}
2585 \DMS{\phi}         {\mathalpha}{\M@greeklower@font}{\3C6}
2586 \DMS{\chi}         {\mathalpha}{\M@greeklower@font}{\3C7}
2587 \DMS{\psi}         {\mathalpha}{\M@greeklower@font}{\3C8}
2588 \DMS{\omega}       {\mathalpha}{\M@greeklower@font}{\3C9}
2589 \DMS{\varbeta}     {\mathalpha}{\M@greeklower@font}{\3D0}
2590 \DMS{\varepsilon}  {\mathalpha}{\M@greeklower@font}{\3F5}
2591 \DMS{\varkappa}    {\mathalpha}{\M@greeklower@font}{\3F0}
2592 \DMS{\vartheta}    {\mathalpha}{\M@greeklower@font}{\3D1}
2593 \DMS{\varrho}      {\mathalpha}{\M@greeklower@font}{\3F1}
2594 \DMS{\varsigma}    {\mathalpha}{\M@greeklower@font}{\3C2}
2595 \DMS{\varphi}       {\mathalpha}{\M@greeklower@font}{\3D5}

```

Set capital ancient Greek characters.

```

\M@greekupper@set 2596 \def\M@greekupper@set{%
2597   \edef\M@greekupper@font{\M\M@greekuppershape\@tempa}
2598   \DMS{\Heta}       {\mathalpha}{\M@greekupper@font}{\370}
2599   \DMS{\Sampi}      {\mathalpha}{\M@greekupper@font}{\3E0}
2600   \DMS{\Digamma}    {\mathalpha}{\M@greekupper@font}{\3DC}
2601   \DMS{\Koppa}      {\mathalpha}{\M@greekupper@font}{\3D8}
2602   \DMS{\Stigma}     {\mathalpha}{\M@greekupper@font}{\3DA}
2603   \DMS{\Sho}        {\mathalpha}{\M@greekupper@font}{\3F7}
2604   \DMS{\San}        {\mathalpha}{\M@greekupper@font}{\3FA}
2605   \DMS{\varSampi}   {\mathalpha}{\M@greekupper@font}{\372}
2606   \DMS{\varDigamma} {\mathalpha}{\M@greekupper@font}{\376}
2607   \DMS{\varKoppa}   {\mathalpha}{\M@greekupper@font}{\3DE}}

```

Set minuscule ancient Greek characters.

```

\M@greeklower@set 2608 \def\M@greeklower@set{%
2609   \edef\M@greeklower@font{\M\M@greeklowershape\@tempa}
2610   \DMS{\heta}       {\mathalpha}{\M@greeklower@font}{\371}
2611   \DMS{\sampi}      {\mathalpha}{\M@greeklower@font}{\3E1}
2612   \DMS{\digamma}    {\mathalpha}{\M@greeklower@font}{\3DD}
2613   \DMS{\koppa}      {\mathalpha}{\M@greeklower@font}{\3D9}
2614   \DMS{\stigma}     {\mathalpha}{\M@greeklower@font}{\3DB}
2615   \DMS{\sho}        {\mathalpha}{\M@greeklower@font}{\3F8}
2616   \DMS{\san}        {\mathalpha}{\M@greeklower@font}{\3FB}
2617   \DMS{\varsampi}   {\mathalpha}{\M@greeklower@font}{\373}
2618   \DMS{\vardigamma} {\mathalpha}{\M@greeklower@font}{\377}
2619   \DMS{\varkoppa}   {\mathalpha}{\M@greeklower@font}{\3DF}}

```

Set capital Cyrillic characters.

```
\M@cyrillicupper@s 2620 \def\M@cyrillicupper@set{%
2621   \edef\M@cyrillicupper@font{M\M@cyrillicuppershape\@tempa}
2622   \@DMS{\cyrA}      {\mathalpha}{\M@cyrillicupper@font}{\ "410}
2623   \@DMS{\cyrBe}     {\mathalpha}{\M@cyrillicupper@font}{\ "411}
2624   \@DMS{\cyrVe}     {\mathalpha}{\M@cyrillicupper@font}{\ "412}
2625   \@DMS{\cyrGhe}    {\mathalpha}{\M@cyrillicupper@font}{\ "413}
2626   \@DMS{\cyrDe}     {\mathalpha}{\M@cyrillicupper@font}{\ "414}
2627   \@DMS{\cyrIe}     {\mathalpha}{\M@cyrillicupper@font}{\ "415}
2628   \@DMS{\cyrZhe}    {\mathalpha}{\M@cyrillicupper@font}{\ "416}
2629   \@DMS{\cyrZe}     {\mathalpha}{\M@cyrillicupper@font}{\ "417}
2630   \@DMS{\cyrI}      {\mathalpha}{\M@cyrillicupper@font}{\ "418}
2631   \@DMS{\cyrKa}     {\mathalpha}{\M@cyrillicupper@font}{\ "41A}
2632   \@DMS{\cyrEl}     {\mathalpha}{\M@cyrillicupper@font}{\ "41B}
2633   \@DMS{\cyrEm}     {\mathalpha}{\M@cyrillicupper@font}{\ "41C}
2634   \@DMS{\cyrEn}     {\mathalpha}{\M@cyrillicupper@font}{\ "41D}
2635   \@DMS{\cyrO}      {\mathalpha}{\M@cyrillicupper@font}{\ "41E}
2636   \@DMS{\cyrPe}     {\mathalpha}{\M@cyrillicupper@font}{\ "41F}
2637   \@DMS{\cyrEr}     {\mathalpha}{\M@cyrillicupper@font}{\ "420}
2638   \@DMS{\cyrEs}     {\mathalpha}{\M@cyrillicupper@font}{\ "421}
2639   \@DMS{\cyrTe}     {\mathalpha}{\M@cyrillicupper@font}{\ "422}
2640   \@DMS{\cyrU}      {\mathalpha}{\M@cyrillicupper@font}{\ "423}
2641   \@DMS{\cyrEf}     {\mathalpha}{\M@cyrillicupper@font}{\ "424}
2642   \@DMS{\cyrHa}     {\mathalpha}{\M@cyrillicupper@font}{\ "425}
2643   \@DMS{\cyrTse}    {\mathalpha}{\M@cyrillicupper@font}{\ "426}
2644   \@DMS{\cyrChe}    {\mathalpha}{\M@cyrillicupper@font}{\ "427}
2645   \@DMS{\cyrSha}    {\mathalpha}{\M@cyrillicupper@font}{\ "428}
2646   \@DMS{\cyrShcha}  {\mathalpha}{\M@cyrillicupper@font}{\ "429}
2647   \@DMS{\cyrHard}   {\mathalpha}{\M@cyrillicupper@font}{\ "42A}
2648   \@DMS{\cyrYeru}   {\mathalpha}{\M@cyrillicupper@font}{\ "42B}
2649   \@DMS{\cyrSoft}   {\mathalpha}{\M@cyrillicupper@font}{\ "42C}
2650   \@DMS{\cyrE}      {\mathalpha}{\M@cyrillicupper@font}{\ "42D}
2651   \@DMS{\cyrYu}     {\mathalpha}{\M@cyrillicupper@font}{\ "42E}
2652   \@DMS{\cyrYa}     {\mathalpha}{\M@cyrillicupper@font}{\ "42F}
2653   \@DMS{\cyrvarI}   {\mathalpha}{\M@cyrillicupper@font}{\ "419}}
```

Set minuscule Cyrillic characters.

```
\M@cyrilliclower@s 2654 \def\M@cyrilliclower@set{%
2655   \edef\M@cyrilliclower@font{M\M@cyrilliclowershape\@tempa}
2656   \@DMS{\cyra}      {\mathalpha}{\M@cyrilliclower@font}{\ "430}
2657   \@DMS{\cyrbe}     {\mathalpha}{\M@cyrilliclower@font}{\ "431}
2658   \@DMS{\cyrve}     {\mathalpha}{\M@cyrilliclower@font}{\ "432}
2659   \@DMS{\cyrghhe}   {\mathalpha}{\M@cyrilliclower@font}{\ "433}}
```



```

2660 \@DMS{\cyrde} {\mathalpha}{\M@cyrilliclower@font}{\{"434}
2661 \@DMS{\cyrrie} {\mathalpha}{\M@cyrilliclower@font}{\{"435}
2662 \@DMS{\cyrzhe} {\mathalpha}{\M@cyrilliclower@font}{\{"436}
2663 \@DMS{\cyrze} {\mathalpha}{\M@cyrilliclower@font}{\{"437}
2664 \@DMS{\cyrri} {\mathalpha}{\M@cyrilliclower@font}{\{"438}
2665 \@DMS{\cyrka} {\mathalpha}{\M@cyrilliclower@font}{\{"43A}
2666 \@DMS{\cyrel} {\mathalpha}{\M@cyrilliclower@font}{\{"43B}
2667 \@DMS{\cyrem} {\mathalpha}{\M@cyrilliclower@font}{\{"43C}
2668 \@DMS{\cyren} {\mathalpha}{\M@cyrilliclower@font}{\{"43D}
2669 \@DMS{\cyro} {\mathalpha}{\M@cyrilliclower@font}{\{"43E}
2670 \@DMS{\cyrpe} {\mathalpha}{\M@cyrilliclower@font}{\{"43F}
2671 \@DMS{\cyrer} {\mathalpha}{\M@cyrilliclower@font}{\{"440}
2672 \@DMS{\cyres} {\mathalpha}{\M@cyrilliclower@font}{\{"441}
2673 \@DMS{\cyrte} {\mathalpha}{\M@cyrilliclower@font}{\{"442}
2674 \@DMS{\cyru} {\mathalpha}{\M@cyrilliclower@font}{\{"443}
2675 \@DMS{\cyref} {\mathalpha}{\M@cyrilliclower@font}{\{"444}
2676 \@DMS{\cyrha} {\mathalpha}{\M@cyrilliclower@font}{\{"445}
2677 \@DMS{\cyrts} {\mathalpha}{\M@cyrilliclower@font}{\{"446}
2678 \@DMS{\cyrche} {\mathalpha}{\M@cyrilliclower@font}{\{"447}
2679 \@DMS{\cyrsha} {\mathalpha}{\M@cyrilliclower@font}{\{"448}
2680 \@DMS{\cyrshcha} {\mathalpha}{\M@cyrilliclower@font}{\{"449}
2681 \@DMS{\cyrhard} {\mathalpha}{\M@cyrilliclower@font}{\{"44A}
2682 \@DMS{\cyrgeru} {\mathalpha}{\M@cyrilliclower@font}{\{"44B}
2683 \@DMS{\cyrsoft} {\mathalpha}{\M@cyrilliclower@font}{\{"44C}
2684 \@DMS{\cyre} {\mathalpha}{\M@cyrilliclower@font}{\{"44D}
2685 \@DMS{\cyrju} {\mathalpha}{\M@cyrilliclower@font}{\{"44E}
2686 \@DMS{\cyrja} {\mathalpha}{\M@cyrilliclower@font}{\{"44F}
2687 \@DMS{\cyrvari} {\mathalpha}{\M@cyrilliclower@font}{\{"439}}

```

Set Hebrew characters.

```

\M@hebrew@set 2688 \def\M@hebrew@set{%
2689 \edef\M@hebrew@font{M\M@hebrewshape\@tempa}
2690 \@DMS{\aleph} {\mathalpha}{\M@hebrew@font}{\{"5D0}
2691 \@DMS{\beth} {\mathalpha}{\M@hebrew@font}{\{"5D1}
2692 \@DMS{\gimel} {\mathalpha}{\M@hebrew@font}{\{"5D2}
2693 \@DMS{\daleth} {\mathalpha}{\M@hebrew@font}{\{"5D3}
2694 \@DMS{\he} {\mathalpha}{\M@hebrew@font}{\{"5D4}
2695 \@DMS{\vav} {\mathalpha}{\M@hebrew@font}{\{"5D5}
2696 \@DMS{\zayin} {\mathalpha}{\M@hebrew@font}{\{"5D6}
2697 \@DMS{\het} {\mathalpha}{\M@hebrew@font}{\{"5D7}
2698 \@DMS{\tet} {\mathalpha}{\M@hebrew@font}{\{"5D8}
2699 \@DMS{\yod} {\mathalpha}{\M@hebrew@font}{\{"5D9}
2700 \@DMS{\kaf} {\mathalpha}{\M@hebrew@font}{\{"5DB}

```

```

2701 \@DMS{\lamed}    {\mathalpha}{\M@hebrew@font}{\5DC}
2702 \@DMS{\mem}     {\mathalpha}{\M@hebrew@font}{\5DE}
2703 \@DMS{\nun}     {\mathalpha}{\M@hebrew@font}{\5E0}
2704 \@DMS{\samekh}  {\mathalpha}{\M@hebrew@font}{\5E1}
2705 \@DMS{\ayin}    {\mathalpha}{\M@hebrew@font}{\5E2}
2706 \@DMS{\pe}      {\mathalpha}{\M@hebrew@font}{\5E4}
2707 \@DMS{\tsadi}   {\mathalpha}{\M@hebrew@font}{\5E6}
2708 \@DMS{\qof}    {\mathalpha}{\M@hebrew@font}{\5E7}
2709 \@DMS{\resh}   {\mathalpha}{\M@hebrew@font}{\5E8}
2710 \@DMS{\shin}    {\mathalpha}{\M@hebrew@font}{\5E9}
2711 \@DMS{\tav}     {\mathalpha}{\M@hebrew@font}{\5EA}
2712 \@DMS{\varkaf}  {\mathalpha}{\M@hebrew@font}{\5DA}
2713 \@DMS{\varmem}  {\mathalpha}{\M@hebrew@font}{\5DD}
2714 \@DMS{\varnun}  {\mathalpha}{\M@hebrew@font}{\5DF}
2715 \@DMS{\varpe}   {\mathalpha}{\M@hebrew@font}{\5E3}
2716 \@DMS{\vartsadi}{\mathalpha}{\M@hebrew@font}{\5E5}

```

Set digits.

```

\M@digits@set 2717 \def\M@digits@set{%
2718   \edef\M@digits@font{M\M@digitsshape\@tempa}
2719   \@DMS{0}{\mathalpha}{\M@digits@font}{\`0}
2720   \@DMS{1}{\mathalpha}{\M@digits@font}{\`1}
2721   \@DMS{2}{\mathalpha}{\M@digits@font}{\`2}
2722   \@DMS{3}{\mathalpha}{\M@digits@font}{\`3}
2723   \@DMS{4}{\mathalpha}{\M@digits@font}{\`4}
2724   \@DMS{5}{\mathalpha}{\M@digits@font}{\`5}
2725   \@DMS{6}{\mathalpha}{\M@digits@font}{\`6}
2726   \@DMS{7}{\mathalpha}{\M@digits@font}{\`7}
2727   \@DMS{8}{\mathalpha}{\M@digits@font}{\`8}
2728   \@DMS{9}{\mathalpha}{\M@digits@font}{\`9}}

```

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

```

\M@operator@set 2729 \def\M@operator@set{%
2730   \ifM@adjust@font

```

```

\M@operator@num 2731 \edef\M@operator@num{\number
2732 \csname symM\M@operator@shape\@tempa\endcsname}
\M@operator@mathco 2733 \protected\edef\M@operator@mathcodes{%
2734 \Umathcode`A=7+\M@operator@num+`A\relax
2735 \Umathcode`B=7+\M@operator@num+`B\relax
2736 \Umathcode`C=7+\M@operator@num+`C\relax
2737 \Umathcode`D=7+\M@operator@num+`D\relax
2738 \Umathcode`E=7+\M@operator@num+`E\relax
2739 \Umathcode`F=7+\M@operator@num+`F\relax
2740 \Umathcode`G=7+\M@operator@num+`G\relax
2741 \Umathcode`H=7+\M@operator@num+`H\relax
2742 \Umathcode`I=7+\M@operator@num+`I\relax
2743 \Umathcode`J=7+\M@operator@num+`J\relax
2744 \Umathcode`K=7+\M@operator@num+`K\relax
2745 \Umathcode`L=7+\M@operator@num+`L\relax
2746 \Umathcode`M=7+\M@operator@num+`M\relax
2747 \Umathcode`N=7+\M@operator@num+`N\relax
2748 \Umathcode`O=7+\M@operator@num+`O\relax
2749 \Umathcode`P=7+\M@operator@num+`P\relax
2750 \Umathcode`Q=7+\M@operator@num+`Q\relax
2751 \Umathcode`R=7+\M@operator@num+`R\relax
2752 \Umathcode`S=7+\M@operator@num+`S\relax
2753 \Umathcode`T=7+\M@operator@num+`T\relax
2754 \Umathcode`U=7+\M@operator@num+`U\relax
2755 \Umathcode`V=7+\M@operator@num+`V\relax
2756 \Umathcode`W=7+\M@operator@num+`W\relax
2757 \Umathcode`X=7+\M@operator@num+`X\relax
2758 \Umathcode`Y=7+\M@operator@num+`Y\relax
2759 \Umathcode`Z=7+\M@operator@num+`Z\relax
2760 \Umathcode`a=7+\M@operator@num+`a\relax
2761 \Umathcode`b=7+\M@operator@num+`b\relax
2762 \Umathcode`c=7+\M@operator@num+`c\relax
2763 \Umathcode`d=7+\M@operator@num+`d\relax
2764 \Umathcode`e=7+\M@operator@num+`e\relax
2765 \Umathcode`f=7+\M@operator@num+`f\relax
2766 \Umathcode`g=7+\M@operator@num+`g\relax
2767 \Umathcode`h=7+\M@operator@num+`h\relax
2768 \Umathcode`i=7+\M@operator@num+`i\relax
2769 \Umathcode`j=7+\M@operator@num+`j\relax
2770 \Umathcode`k=7+\M@operator@num+`k\relax
2771 \Umathcode`l=7+\M@operator@num+`l\relax
2772 \Umathcode`m=7+\M@operator@num+`m\relax

```

```

2773 \Umathcode`n=7+\M@operator@num+`n\relax
2774 \Umathcode`o=7+\M@operator@num+`o\relax
2775 \Umathcode`p=7+\M@operator@num+`p\relax
2776 \Umathcode`q=7+\M@operator@num+`q\relax
2777 \Umathcode`r=7+\M@operator@num+`r\relax
2778 \Umathcode`s=7+\M@operator@num+`s\relax
2779 \Umathcode`t=7+\M@operator@num+`t\relax
2780 \Umathcode`u=7+\M@operator@num+`u\relax
2781 \Umathcode`v=7+\M@operator@num+`v\relax
2782 \Umathcode`w=7+\M@operator@num+`w\relax
2783 \Umathcode`x=7+\M@operator@num+`x\relax
2784 \Umathcode`y=7+\M@operator@num+`y\relax
2785 \Umathcode`z=7+\M@operator@num+`z\relax
2786 \Umathchardef\imath=7+\M@operator@num+1044506\relax
2787 \Umathchardef\jmath=7+\M@operator@num+1044500\relax}
2788 \else
\M@operator@mathco 2789 \let\M@operator@mathcodes\@empty
2790 \fi

```

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

```

\operator@font 2791 \xdef\operator@font{\mathgroup
2792 \csname symM\M@operatorsshape\@tempa\endcsname
2793 \M@operator@mathcodes}}

```

Set delimiters.

```

2794 \ifM@adjust@font
\M@delimiters@set 2795 \def\M@delimiters@set{%
2796 \edef\M@delimiters@font{M\M@delimitersshape\@tempa}
\M@delimiters@num 2797 \edef\M@delimiters@num{%
2798 \csname sym\M@delimiters@font\endcsname}
2799 \@DMS{(} {\mathopen} {\M@delimiters@font}{"28}
2800 \@DMS{)} {\mathclose} {\M@delimiters@font}{"29}
2801 \@DMS{[} {\mathopen} {\M@delimiters@font}{"5B}
2802 \@DMS{]} {\mathclose} {\M@delimiters@font}{"5D}
2803 \@DMS{\leftbrace} {\mathopen} {\M@delimiters@font}{"7B}
2804 \@DMS{\rightbrace} {\mathclose} {\M@delimiters@font}{"7D}

```

Set `\Udelcodes` for delimiters that come from individual characters.

```

2805 \global\Udelcode40 +\M@delimiters@num+40\relax % (
2806 \global\Udelcode41 +\M@delimiters@num+41\relax % )
2807 \global\Udelcode47 +\M@delimiters@num+47\relax % /
2808 \global\Udelcode91 +\M@delimiters@num+91\relax % [
2809 \global\Udelcode93 +\M@delimiters@num+93\relax % ]

```

```

2810 \global\Udelcode124+\M@delimiters@num+124\relax % |
2811 \ifM@symbols\else
2812 \DMS{|}{\mathord}{\M@delimiters@font}{"7C}
2813 \fi
2814 \global\let\vert=|

```

For the delimiters that come from control sequences, we use `\xdef` and `\Udelimiter`.

```

2815 \protected\xdef\backslash{%
2816 \ifmmode\mathbackslash\else\textbackslash\fi}
2817 \protected\xdef\mathbackslash{%
2818 \Udelimiter+2+\M@delimiters@num+92\relax}
2819 \protected\xdef\lbrace{%
2820 \Udelimiter+4+\M@delimiters@num+123\relax}
2821 \protected\xdef\rbrace{%
2822 \Udelimiter+5+\M@delimiters@num+125\relax}
2823 \protected\xdef\lguil{%
2824 \Udelimiter+4+\M@delimiters@num+8249\relax}
2825 \protected\xdef\rguil{%
2826 \Udelimiter+5+\M@delimiters@num+8250\relax}
2827 \protected\xdef\llguil{%
2828 \Udelimiter+4+\M@delimiters@num+171\relax}
2829 \protected\xdef\rrguil{%
2830 \Udelimiter+5+\M@delimiters@num+187\relax}
2831 \protected\xdef\fakelangle{%
2832 \Udelimiter+4+\M@delimiters@num+1044508\relax}
2833 \protected\xdef\fakerangle{%
2834 \Udelimiter+5+\M@delimiters@num+1044509\relax}
2835 \protected\xdef\fakellangle{%
2836 \Udelimiter+4+\M@delimiters@num+1044510\relax}
2837 \protected\xdef\fakerrangle{%
2838 \Udelimiter+5+\M@delimiters@num+1044511\relax}}
2839 \else
\M@delimiters@set 2840 \def\M@delimiters@set{%
2841 \edef\M@delimiters@font{M\M@delimiters@shape\@tempa}
2842 \DMS{(} {\mathopen} {\M@delimiters@font}{"28}
2843 \DMS{)} {\mathclose}{\M@delimiters@font}{"29}
2844 \DMS{[} {\mathopen} {\M@delimiters@font}{"5B}
2845 \DMS{]} {\mathclose}{\M@delimiters@font}{"5D}
2846 \DMS{\lguil} {\mathopen} {\M@delimiters@font}{"2039}
2847 \DMS{\rguil} {\mathclose}{\M@delimiters@font}{"203A}
2848 \DMS{\llguil} {\mathopen} {\M@delimiters@font}{"AB}
2849 \DMS{\rrguil} {\mathclose}{\M@delimiters@font}{"BB}

```

```

2850 \DMS{\leftbrace} {\mathopen} {\M@delimiters@font}{\7B}
2851 \DMS{\rightbrace}{\mathclose}{\M@delimiters@font}{\7D}}
2852 \fi

```

Radicals.

```

2853 \ifM@adjust@font
\M@radical@set 2854 \def\M@radical@set{%
2855 \edef\M@radical@font{M\M@radicalshape\@tempa}
2856 \DMS{\surd}{\mathord}{\M@radical@font}{\221A}
\sqrts@gn 2857 \xdef\sqrts@gn##1{\Uradical+\number
2858 \csname sym\M@radical@font\endcsname+8730\relax{##1}}

```

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

```

\r@ot 2859 \gdef\r@ot##1##2{%
2860 \setbox\z@\hbox{\$ \m@th##1\sqrtsign{##2}$}%
2861 \setbox\surdbox\hbox{\$ \m@th##1\sqrts@gn{%
2862 \hbox{\vphantom{\$ \m@th##1##2$}}}%}
2863 \dimen@ \ht\surdbox
2864 \advance\dimen@ \dp\surdbox
2865 \dimen@ = 0.6\dimen@
2866 \advance\dimen@ - \dp\surdbox
2867 \ifdim\wd\rootbox < 0.6\wd\surdbox
2868 \kern 0.6\wd\surdbox
2869 \else
2870 \kern\wd\rootbox
2871 \fi
2872 \raise\dimen@\hbox{\llap{\copy\rootbox}}
2873 \kern -0.6\wd\surdbox
2874 \box\z@}
\sqrtsign 2875 \gdef\sqrtsign##1{\sqrts@gn{\mkern\radicandoffset##1}}
2876 \else
\M@radical@set 2877 \def\M@radical@set{%
2878 \edef\M@radical@font{M\M@radicalshape\@tempa}
2879 \DMS{\surd}{\mathord}{\M@radical@font}{\221A}}
2880 \fi

```

Big operators.

```

\M@bigops@set 2881 \def\M@bigops@set{%
2882 \edef\M@bigops@font{M\M@bigopssshape\@tempa}
2883 \let\sum\@undefined
2884 \let\prod\@undefined

```

2885 \@DMS{\sum}{\mathop}{\M@bigops@font}{"2211}
2886 \@DMS{\prod}{\mathop}{\M@bigops@font}{"220F}
2887 \@DMS{\intop}{\mathop}{\M@bigops@font}{"222B}}

Extended big operators.

```

\M@extbigops@set 2888 \def\M@extbigops@set{%
2889   \edef\M@extbigops@font{M\M@extbigopssshape\@tempa}
2890   \let\coprod\@undefined
2891   \let\bigvee\@undefined
2892   \let\bigwedge\@undefined
2893   \let\bigcup\@undefined
2894   \let\bigcap\@undefined
2895   \let\bigoplus\@undefined
2896   \let\bigotimes\@undefined
2897   \let\bigodot\@undefined
2898   \let\bigsqcup\@undefined
2899   \@DMS{\coprod}    {\mathop}{\M@extbigops@font}{ "2210}
2900   \@DMS{\bigvee}    {\mathop}{\M@extbigops@font}{ "22C1}
2901   \@DMS{\bigwedge}  {\mathop}{\M@extbigops@font}{ "22C0}
2902   \@DMS{\bigcup}    {\mathop}{\M@extbigops@font}{ "22C3}
2903   \@DMS{\bigcap}    {\mathop}{\M@extbigops@font}{ "22C2}
2904   \@DMS{\iintop}    {\mathop}{\M@extbigops@font}{ "222C}
2905   \@DMS{\iiintop}   {\mathop}{\M@extbigops@font}{ "222D}
2906   \@DMS{\ointop}    {\mathop}{\M@extbigops@font}{ "222E}
2907   \@DMS{\oiintop}   {\mathop}{\M@extbigops@font}{ "222F}
2908   \@DMS{\oiintop}   {\mathop}{\M@extbigops@font}{ "2230}
2909   \@DMS{\bigoplus}  {\mathop}{\M@extbigops@font}{ "2A01}
2910   \@DMS{\bigotimes} {\mathop}{\M@extbigops@font}{ "2A02}
2911   \@DMS{\bigodot}   {\mathop}{\M@extbigops@font}{ "2A00}
2912   \@DMS{\bigsqcap}  {\mathop}{\M@extbigops@font}{ "2A05}
2913   \@DMS{\bigsqcup}  {\mathop}{\M@extbigops@font}{ "2A06}
2914   \protected\gdef\iint{\iintop\nolimits}
2915   \protected\gdef\iiint{\iiintop\nolimits}
2916   \protected\gdef\oint{\ointop\nolimits}
2917   \protected\gdef\oiint{\oiintop\nolimits}
2918   \protected\gdef\oiint{\oiintop\nolimits}}

```

Set symbols.

```

\M@symbols@set 2919 \def\M@symbols@set{%
                2920 \edef\M@symbols@font{M\M@symbolsshape\@tempa}
                2921 \let\colon\@undefined
                2922 \let\mathellipsis\@undefined

```

Before we start declaring symbols, specifically minus or equals signs, we have to address a minor clash with `amsmath`. That package defines `\relbar` and `\Relbar` as essentially a minus and equals sign respectively. However, those two control sequences are for making arrows, so they should come from the `arrows` font, not the `symbols` font. If the user already called `\mathfont` with the `arrows` keyword, we do nothing because `\M@arrows@set` defines `\relbar` and `\Relbar` correctly. If not, we make these two control sequences be the current minus and equals sign (before the font changes in `\M@symbols@set`) because that's as good a choice as any, and we prevent `amsmath` from changing them to the `symbols` font. Users or package authors who want to modify `\relbar` or `\Relbar` should change `\@relbar` or `\@Relbar` respectively.

```

2923 \ifM@arrows\else
2924   \global\Umathcharnumdef\@relbar=\Umathcodenum`\-
2925   \global\Umathcharnumdef\@Relbar=\Umathcodenum`\=
2926   \protected\gdef\relbar{\mathrel
2927     {\mathpalette\mathsm@sh\@relbar}}
2928   \protected\gdef\Relbar{\@Relbar}

```

We redefine stuff if `amsmath` gets loaded after `mathfont`.

```

2929 \ifpackageloaded{amsmath}
2930   {\relax}{%
2931     \global\let\@relbar\relbar
2932     \global\let\@Relbar\Relbar
2933     \AtBeginDocument{\ifM@arrows\else
2934       \ifpackageloaded{amsmath}{%
2935         \global\let\relbar\@relbar
2936         \global\let\Relbar\@Relbar}
2937       {\relax}
2938     \fi}}
2939 \fi

```

The rest of the symbols.

```

2940 \@DMS{.}          {\mathord} {\M@symbols@font}{"2E}
2941 \@DMS{@}          {\mathord} {\M@symbols@font}{"40}
2942 \@DMS{' }         {\mathord} {\M@symbols@font}{"2032}
2943 \@DMS{\prime}     {\mathord} {\M@symbols@font}{"2032}
2944 \@DMS{" }         {\mathord} {\M@symbols@font}{"2033}
2945 \@DMS{\mathhash}  {\mathord} {\M@symbols@font}{"23}
2946 \@DMS{\mathdollar} {\mathord} {\M@symbols@font}{"24}
2947 \@DMS{\mathpercent} {\mathord} {\M@symbols@font}{"25}
2948 \@DMS{\mathand}    {\mathord} {\M@symbols@font}{"26}
2949 \@DMS{\mathparagraph} {\mathord} {\M@symbols@font}{"B6}
2950 \@DMS{\mathsection} {\mathord} {\M@symbols@font}{"A7}

```


2951	<code>\@DMS{\mathsterling}</code>	<code>{\mathord}</code>	<code>{\M@symbols@font}{\A3}</code>
2952	<code>\@DMS{\neg}</code>	<code>{\mathord}</code>	<code>{\M@symbols@font}{\AC}</code>
2953	<code>\@DMS{ \}</code>	<code>{\mathord}</code>	<code>{\M@symbols@font}{\7C}</code>
2954	<code>\@DMS{\infty}</code>	<code>{\mathord}</code>	<code>{\M@symbols@font}{\221E}</code>
2955	<code>\@DMS{\partial}</code>	<code>{\mathord}</code>	<code>{\M@symbols@font}{\2202}</code>
2956	<code>\@DMS{\degree}</code>	<code>{\mathord}</code>	<code>{\M@symbols@font}{\B0}</code>
2957	<code>\@DMS{\increment}</code>	<code>{\mathord}</code>	<code>{\M@symbols@font}{\2206}</code>
2958	<code>\@DMS{\comma}</code>	<code>{\mathord}</code>	<code>{\M@symbols@font}{\2C}</code>
2959	<code>\@DMS{+}</code>	<code>{\mathbin}</code>	<code>{\M@symbols@font}{\2B}</code>
2960	<code>\@DMS{-}</code>	<code>{\mathbin}</code>	<code>{\M@symbols@font}{\2212}</code>
2961	<code>\@DMS{*}</code>	<code>{\mathbin}</code>	<code>{\M@symbols@font}{\2A}</code>
2962	<code>\@DMS{\times}</code>	<code>{\mathbin}</code>	<code>{\M@symbols@font}{\D7}</code>
2963	<code>\@DMS{/}</code>	<code>{\mathord}</code>	<code>{\M@symbols@font}{\2F}</code>
2964	<code>\@DMS{\fractionslash}</code>	<code>{\mathord}</code>	<code>{\M@symbols@font}{\2215}</code>
2965	<code>\@DMS{\div}</code>	<code>{\mathbin}</code>	<code>{\M@symbols@font}{\F7}</code>
2966	<code>\@DMS{\pm}</code>	<code>{\mathbin}</code>	<code>{\M@symbols@font}{\B1}</code>
2967	<code>\@DMS{\bullet}</code>	<code>{\mathbin}</code>	<code>{\M@symbols@font}{\2022}</code>
2968	<code>\@DMS{\dagger}</code>	<code>{\mathbin}</code>	<code>{\M@symbols@font}{\2020}</code>
2969	<code>\@DMS{\ddagger}</code>	<code>{\mathbin}</code>	<code>{\M@symbols@font}{\2021}</code>
2970	<code>\@DMS{\cdot}</code>	<code>{\mathbin}</code>	<code>{\M@symbols@font}{\2219}</code>
2971	<code>\@DMS{\setminus}</code>	<code>{\mathbin}</code>	<code>{\M@symbols@font}{\5C}</code>
2972	<code>\@DMS{=}</code>	<code>{\mathrel}</code>	<code>{\M@symbols@font}{\3D}</code>
2973	<code>\@DMS{<}</code>	<code>{\mathrel}</code>	<code>{\M@symbols@font}{\3C}</code>
2974	<code>\@DMS{>}</code>	<code>{\mathrel}</code>	<code>{\M@symbols@font}{\3E}</code>
2975	<code>\@DMS{\leq}</code>	<code>{\mathrel}</code>	<code>{\M@symbols@font}{\2264}</code>
2976	<code>\@DMS{\geq}</code>	<code>{\mathrel}</code>	<code>{\M@symbols@font}{\2265}</code>
2977	<code>\@DMS{\sim}</code>	<code>{\mathrel}</code>	<code>{\M@symbols@font}{\7E}</code>
2978	<code>\@DMS{\approx}</code>	<code>{\mathrel}</code>	<code>{\M@symbols@font}{\2248}</code>
2979	<code>\@DMS{\equiv}</code>	<code>{\mathrel}</code>	<code>{\M@symbols@font}{\2261}</code>
2980	<code>\@DMS{\mid}</code>	<code>{\mathrel}</code>	<code>{\M@symbols@font}{\7C}</code>
2981	<code>\@DMS{\parallel}</code>	<code>{\mathrel}</code>	<code>{\M@symbols@font}{\2016}</code>
2982	<code>\@DMS{:}</code>	<code>{\mathrel}</code>	<code>{\M@symbols@font}{\3A}</code>
2983	<code>\@DMS{?}</code>	<code>{\mathclose}</code>	<code>{\M@symbols@font}{\3F}</code>
2984	<code>\@DMS{!}</code>	<code>{\mathclose}</code>	<code>{\M@symbols@font}{\21}</code>
2985	<code>\@DMS{,}</code>	<code>{\mathpunct}</code>	<code>{\M@symbols@font}{\2C}</code>
2986	<code>\@DMS{;}</code>	<code>{\mathpunct}</code>	<code>{\M@symbols@font}{\3B}</code>
2987	<code>\@DMS{\colon}</code>	<code>{\mathpunct}</code>	<code>{\M@symbols@font}{\3A}</code>
2988	<code>\@DMS{\mathellipsis}</code>	<code>{\mathinner}</code>	<code>{\M@symbols@font}{\2026}</code>

Now a bit of housekeeping. We redefine `\#`, `\%`, and `\&` as `\protected` macros 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` if the user hasn't called `\mathfont` with `extsymbols`.

```

2989 \protected\gdef\#{\ifmmode\mathhash\else\char"23\relax\fi}
2990 \protected\gdef\%{\ifmmode\mathpercent\else\char"25\relax\fi}
2991 \protected\gdef\&{\ifmmode\mathand\else\char"26\relax\fi}
2992 \ifM@extsymbols\else
2993   \protected\gdef\simeq{%
2994     \mathrel{\mathpalette\stack@flatrel{\{-}\{\sim}\}}
2995   \protected\gdef\cong{%
2996     \mathrel{\mathpalette\stack@flatrel{\{=\}\{\sim}\}}}
2997 \fi
2998 \protected\gdef\models{\mathrel{|\}\joinrel\mathrel{=}}
```

If the user enabled Lua-based font adjustments, we declare a few more big operators for fun. For brevity, we put the `adjust@font` conditional here rather than redefining `M@symbols@set`. Apparently, `newtx` defines `\bigtimes`, so if that package gets loaded ahead of `mathfont`, we should make sure to clear that definition.

```

2999 \ifM@adjust@font
3000   \let\bigtimes\@undefined
3001   \@DMS{\bigat}\mathop{\M@symbols@font}{40}
3002   \@DMS{\bighash}\mathop{\M@symbols@font}{23}
3003   \@DMS{\bigdollar}\mathop{\M@symbols@font}{24}
3004   \@DMS{\bigpercent}\mathop{\M@symbols@font}{25}
3005   \@DMS{\bigand}\mathop{\M@symbols@font}{26}
3006   \@DMS{\bigplus}\mathop{\M@symbols@font}{2B}
3007   \@DMS{\bigp}\mathop{\M@symbols@font}{21}
3008   \@DMS{\bigq}\mathop{\M@symbols@font}{3F}
3009   \@DMS{\bigS}\mathop{\M@symbols@font}{A7}
3010   \@DMS{\bigtimes}\mathop{\M@symbols@font}{D7}
3011   \@DMS{\bigdiv}\mathop{\M@symbols@font}{F7}
```

Define `\nabla` here if we're adjusting the font. If we are not doing that, this declaration goes in `extsymbols`.

```

3012   \@DMS{\nabla}\mathord{\M@symbols@font}{2207}
3013 \fi
```

Set extended symbols.

```

\M@extsymbols@set 3014 \def\M@extsymbols@set{%
3015   \edef\M@extsymbols@font{M\M@extsymbolsshape\@tempa}
3016   \let\angle\@undefined
3017   \let\simeq\@undefined
3018   \let\sqsubset\@undefined
3019   \let\sqsupset\@undefined
```

3020	<code>\let\bowtie\@undefined</code>	
3021	<code>\let\doteq\@undefined</code>	
3022	<code>\let\neq\@undefined</code>	
3023	<code>\@DMS{\wp}</code>	<code>{\mathord}{\M@extsymbols@font}{ "2118}</code>
3024	<code>\@DMS{\ell}</code>	<code>{\mathord}{\M@extsymbols@font}{ "2113}</code>
3025	<code>\@DMS{\forall}</code>	<code>{\mathord}{\M@extsymbols@font}{ "2200}</code>
3026	<code>\@DMS{\exists}</code>	<code>{\mathord}{\M@extsymbols@font}{ "2203}</code>
3027	<code>\@DMS{\emptyset}</code>	<code>{\mathord}{\M@extsymbols@font}{ "2205}</code>
3028	<code>\@DMS{\in}</code>	<code>{\mathord}{\M@extsymbols@font}{ "2208}</code>
3029	<code>\@DMS{\ni}</code>	<code>{\mathord}{\M@extsymbols@font}{ "220B}</code>
3030	<code>\@DMS{\mp}</code>	<code>{\mathord}{\M@extsymbols@font}{ "2213}</code>
3031	<code>\@DMS{\angle}</code>	<code>{\mathord}{\M@extsymbols@font}{ "2220}</code>
3032	<code>\@DMS{\top}</code>	<code>{\mathord}{\M@extsymbols@font}{ "22A4}</code>
3033	<code>\@DMS{\bot}</code>	<code>{\mathord}{\M@extsymbols@font}{ "22A5}</code>
3034	<code>\@DMS{\vdash}</code>	<code>{\mathord}{\M@extsymbols@font}{ "22A2}</code>
3035	<code>\@DMS{\dashv}</code>	<code>{\mathord}{\M@extsymbols@font}{ "22A3}</code>
3036	<code>\@DMS{\flat}</code>	<code>{\mathord}{\M@extsymbols@font}{ "266D}</code>
3037	<code>\@DMS{\natural}</code>	<code>{\mathord}{\M@extsymbols@font}{ "266E}</code>
3038	<code>\@DMS{\sharp}</code>	<code>{\mathord}{\M@extsymbols@font}{ "266F}</code>
3039	<code>\@DMS{\fflat}</code>	<code>{\mathord}{\M@extsymbols@font}{ "1D12B}</code>
3040	<code>\@DMS{\ssharp}</code>	<code>{\mathord}{\M@extsymbols@font}{ "1D12A}</code>
3041	<code>\@DMS{\clubsuit}</code>	<code>{\mathord}{\M@extsymbols@font}{ "2663}</code>
3042	<code>\@DMS{\diamondsuit}</code>	<code>{\mathord}{\M@extsymbols@font}{ "2666}</code>
3043	<code>\@DMS{\heartsuit}</code>	<code>{\mathord}{\M@extsymbols@font}{ "2665}</code>
3044	<code>\@DMS{\spadesuit}</code>	<code>{\mathord}{\M@extsymbols@font}{ "2660}</code>
3045	<code>\@DMS{\wclubsuit}</code>	<code>{\mathord}{\M@extsymbols@font}{ "2667}</code>
3046	<code>\@DMS{\wdiamondsuit}</code>	<code>{\mathord}{\M@extsymbols@font}{ "2662}</code>
3047	<code>\@DMS{\wheartsuit}</code>	<code>{\mathord}{\M@extsymbols@font}{ "2661}</code>
3048	<code>\@DMS{\wspadesuit}</code>	<code>{\mathord}{\M@extsymbols@font}{ "2664}</code>
3049	<code>\global\let\spadesuit\bspadesuit</code>	
3050	<code>\global\let\heartsuit\wheartsuit</code>	
3051	<code>\global\let\diamondsuit\wdiamondsuit</code>	
3052	<code>\global\let\clubsuit\bclubsuit</code>	
3053	<code>\@DMS{\wedge}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "2227}</code>
3054	<code>\@DMS{\vee}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "2228}</code>
3055	<code>\@DMS{\cap}</code>	<code>{\mathord}{\M@extsymbols@font}{ "2229}</code>
3056	<code>\@DMS{\cup}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "222A}</code>
3057	<code>\@DMS{\sqcap}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "2293}</code>
3058	<code>\@DMS{\sqcup}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "2294}</code>
3059	<code>\@DMS{\amalg}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "2A3F}</code>
3060	<code>\@DMS{\wr}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "2240}</code>
3061	<code>\@DMS{\ast}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "2217}</code>

3062	<code>\@DMS{\star}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "22C6}</code>
3063	<code>\@DMS{\diamond}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "22C4}</code>
3064	<code>\@DMS{\varcdot}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "22C5}</code>
3065	<code>\@DMS{\varsetminus}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "2216}</code>
3066	<code>\@DMS{\oplus}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "2295}</code>
3067	<code>\@DMS{\otimes}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "2297}</code>
3068	<code>\@DMS{\ominus}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "2296}</code>
3069	<code>\@DMS{\odiv}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "2A38}</code>
3070	<code>\@DMS{\oslash}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "2298}</code>
3071	<code>\@DMS{\odot}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "2299}</code>
3072	<code>\@DMS{\sqplus}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "229E}</code>
3073	<code>\@DMS{\sqtimes}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "22A0}</code>
3074	<code>\@DMS{\sqminus}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "229F}</code>
3075	<code>\@DMS{\sqdot}</code>	<code>{\mathbin}{\M@extsymbols@font}{ "22A1}</code>
3076	<code>\@DMS{\in}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2208}</code>
3077	<code>\@DMS{\ni}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "220B}</code>
3078	<code>\@DMS{\subset}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2282}</code>
3079	<code>\@DMS{\supset}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2283}</code>
3080	<code>\@DMS{\subteq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2286}</code>
3081	<code>\@DMS{\supseteq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2287}</code>
3082	<code>\@DMS{\sqsubset}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "228F}</code>
3083	<code>\@DMS{\sqsupset}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2290}</code>
3084	<code>\@DMS{\sqsubteq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2291}</code>
3085	<code>\@DMS{\sqsupseteq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2292}</code>
3086	<code>\@DMS{\triangleleft}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "22B2}</code>
3087	<code>\@DMS{\triangleright}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "22B3}</code>
3088	<code>\@DMS{\trianglelefteq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "22B4}</code>
3089	<code>\@DMS{\trianglerighteq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "22B5}</code>
3090	<code>\@DMS{\propto}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "221D}</code>
3091	<code>\@DMS{\bowtie}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "22C8}</code>
3092	<code>\@DMS{\hourglass}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "29D6}</code>
3093	<code>\@DMS{\therefore}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2234}</code>
3094	<code>\@DMS{\because}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2235}</code>
3095	<code>\@DMS{\ratio}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2236}</code>
3096	<code>\@DMS{\proportion}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2237}</code>
3097	<code>\@DMS{\ll}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "226A}</code>
3098	<code>\@DMS{\gg}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "226B}</code>
3099	<code>\@DMS{\lll}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "22D8}</code>
3100	<code>\@DMS{\ggg}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "22D9}</code>
3101	<code>\@DMS{\leqq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2266}</code>
3102	<code>\@DMS{\geqq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2267}</code>
3103	<code>\@DMS{\lapprox}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2A85}</code>

3104	<code>\@DMS{\gapprox}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2A86}</code>
3105	<code>\@DMS{\simeq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2243}</code>
3106	<code>\@DMS{\eqsim}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2242}</code>
3107	<code>\@DMS{\simeqq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2245}</code>
3108	<code>\global\let\cong\simeqq</code>	
3109	<code>\@DMS{\approxeq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "224A}</code>
3110	<code>\@DMS{\sssim}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "224B}</code>
3111	<code>\@DMS{\seq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "224C}</code>
3112	<code>\@DMS{\doteq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2250}</code>
3113	<code>\@DMS{\coloneq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2254}</code>
3114	<code>\@DMS{\eqcolon}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2255}</code>
3115	<code>\@DMS{\ringeq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2257}</code>
3116	<code>\@DMS{\arceq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2258}</code>
3117	<code>\@DMS{\wedgieq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2259}</code>
3118	<code>\@DMS{\veeeq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "225A}</code>
3119	<code>\@DMS{\stareq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "225B}</code>
3120	<code>\@DMS{\triangleeq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "225C}</code>
3121	<code>\@DMS{\defeq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "225D}</code>
3122	<code>\@DMS{\qeq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "225F}</code>
3123	<code>\@DMS{\lsim}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2272}</code>
3124	<code>\@DMS{\gsim}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2273}</code>
3125	<code>\@DMS{\prec}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "227A}</code>
3126	<code>\@DMS{\succ}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "227B}</code>
3127	<code>\@DMS{\preceq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "227C}</code>
3128	<code>\@DMS{\succeq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "227D}</code>
3129	<code>\@DMS{\preceqq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2AB3}</code>
3130	<code>\@DMS{\succeqq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2AB4}</code>
3131	<code>\@DMS{\precsim}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "227E}</code>
3132	<code>\@DMS{\succsim}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "227F}</code>
3133	<code>\@DMS{\precapprox}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2AB7}</code>
3134	<code>\@DMS{\succapprox}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2AB8}</code>
3135	<code>\@DMS{\precprec}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2ABB}</code>
3136	<code>\@DMS{\succsucc}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2ABC}</code>
3137	<code>\@DMS{\asymp}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "224D}</code>
3138	<code>\@DMS{\nin}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2209}</code>
3139	<code>\@DMS{\nni}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "220C}</code>
3140	<code>\@DMS{\nsubset}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2284}</code>
3141	<code>\@DMS{\nsupset}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2285}</code>
3142	<code>\@DMS{\nsubsetq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2288}</code>
3143	<code>\@DMS{\nsupseteq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "2289}</code>
3144	<code>\@DMS{\subsetneq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "228A}</code>
3145	<code>\@DMS{\supsetneq}</code>	<code>{\mathrel}{\M@extsymbols@font}{ "228B}</code>

3146	<code>\@DMS{\nsqsubseteq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "22E2}</code>
3147	<code>\@DMS{\nsqsupseteq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "22E3}</code>
3148	<code>\@DMS{\sqsubsubseteq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "22E4}</code>
3149	<code>\@DMS{\sqsupseteq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "22E5}</code>
3150	<code>\@DMS{\neq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2260}</code>
3151	<code>\@DMS{\nleq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "226E}</code>
3152	<code>\@DMS{\nleqq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2270}</code>
3153	<code>\@DMS{\ngeq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2271}</code>
3154	<code>\@DMS{\lneq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2A87}</code>
3155	<code>\@DMS{\gneq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2A88}</code>
3156	<code>\@DMS{\lneqq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2268}</code>
3157	<code>\@DMS{\gneqq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2269}</code>
3158	<code>\@DMS{\ntriangleleft}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "22EA}</code>
3159	<code>\@DMS{\ntriangleright}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "22EB}</code>
3160	<code>\@DMS{\ntrianglelefteq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "22EC}</code>
3161	<code>\@DMS{\ntrianglerighteq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "22ED}</code>
3162	<code>\@DMS{\nsim}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2241}</code>
3163	<code>\@DMS{\napprox}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2249}</code>
3164	<code>\@DMS{\nsimeq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2244}</code>
3165	<code>\@DMS{\nsimeqq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2247}</code>
3166	<code>\@DMS{\simneqq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2246}</code>
3167	<code>\@DMS{\nlsim}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2274}</code>
3168	<code>\@DMS{\ngsim}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2275}</code>
3169	<code>\@DMS{\lnsim}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "22E6}</code>
3170	<code>\@DMS{\gnsim}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "22E7}</code>
3171	<code>\@DMS{\lnapprox}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2A89}</code>
3172	<code>\@DMS{\gnapprox}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2A8A}</code>
3173	<code>\@DMS{\nprec}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2280}</code>
3174	<code>\@DMS{\nsucc}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2281}</code>
3175	<code>\@DMS{\npreceq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "22E0}</code>
3176	<code>\@DMS{\nsucceq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "22E1}</code>
3177	<code>\@DMS{\precneq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2AB1}</code>
3178	<code>\@DMS{\succneq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2AB2}</code>
3179	<code>\@DMS{\precneqq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2AB5}</code>
3180	<code>\@DMS{\succneqq}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2AB6}</code>
3181	<code>\@DMS{\precnsim}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "22E8}</code>
3182	<code>\@DMS{\succnsim}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "22E9}</code>
3183	<code>\@DMS{\precnapprox}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2AB9}</code>
3184	<code>\@DMS{\succnapprox}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2ABA}</code>
3185	<code>\@DMS{\nequiv}</code>	<code>{\mathrel}{\M@extsymbols@font}{\ "2262}</code>

The `math-operator` package renames `\Re` and `\Im` to `\varRe` and `\varIm`. To make `mathfont` compatible with that package, we test whether these commands

have been redefined.

```

3186 \ifpackageloaded{math-operator}{\if@operator@s
3187     \@DMS{\varRe}{\mathord}{\M@extsymbols@font}{ "211C}
3188     \@DMS{\varIm}{\mathord}{\M@extsymbols@font}{ "2111}
3189 \else
3190     \@DMS{\Re}{\mathord}{\M@extsymbols@font}{ "211C}
3191     \@DMS{\Im}{\mathord}{\M@extsymbols@font}{ "2111}
3192 \fi}{%
3193     \@DMS{\Re}{\mathord}{\M@extsymbols@font}{ "211C}
3194     \@DMS{\Im}{\mathord}{\M@extsymbols@font}{ "2111}}

```

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

```

3195 \global\let\textng\ng
3196 \@DMS{\mathng}{\mathrel}{\M@extsymbols@font}{ "226F}
3197 \protected\gdef\ng{\ifmmode\mathng\else\textng\fi}

```

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

```

3198 \ifM@adjust@font\else
3199     \@DMS{\nabla}{\mathord}{\M@extsymbols@font}{ "2207}
3200 \fi}

```

Set arrows.

```

\M@arrows@set 3201 \def\M@arrows@set{%
3202     \edef\M@arrows@font{M\M@arrowsshape\@tempa}
3203     \let\uparrow\@undefined
3204     \let\Uparrow\@undefined
3205     \let\downarrow\@undefined
3206     \let\Downarrow\@undefined
3207     \let\updownarrow\@undefined
3208     \let\Updownarrow\@undefined
3209     \let\longrightarrow\@undefined
3210     \let\longleftarrow\@undefined
3211     \let\longleftarrow\@undefined
3212     \let\hookrightarrow\@undefined
3213     \let\hookleftarrow\@undefined
3214     \let\Longrightarrow\@undefined
3215     \let\Longleftarrow\@undefined
3216     \let\Longleftarrow\@undefined
3217     \let\rightleftharpoons\@undefined
3218     \@DMS{\rightarrow}{\mathrel}{\M@arrows@font}{ "2192}
3219     \global\let\to\rightarrow
3220     \@DMS{\nrightarrow}{\mathrel}{\M@arrows@font}{ "219B}
3221     \@DMS{\Rrightarrow}{\mathrel}{\M@arrows@font}{ "21D2}

```

3222	<code>\@DMS{\nRightarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21CF}</code>
3223	<code>\@DMS{\Rrightarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21DB}</code>
3224	<code>\@DMS{\longrightarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "27F6}</code>
3225	<code>\@DMS{\Longrightarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "27F9}</code>
3226	<code>\@DMS{\rightbararrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21A6}</code>
3227	<code>\global\let\mapsto\rightbararrow</code>	
3228	<code>\@DMS{\Rightbararrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "2907}</code>
3229	<code>\@DMS{\longrightbararrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "27FC}</code>
3230	<code>\global\let\longmapsto\longrightbararrow</code>	
3231	<code>\@DMS{\Longrightbararrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "27FE}</code>
3232	<code>\@DMS{\hookrightarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21AA}</code>
3233	<code>\@DMS{\righdasharrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21E2}</code>
3234	<code>\@DMS{\rightharpoonup}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21C0}</code>
3235	<code>\@DMS{\rightharpoondown}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21C1}</code>
3236	<code>\@DMS{\rightarrowtail}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21A3}</code>
3237	<code>\@DMS{\rightrightarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "27F4}</code>
3238	<code>\@DMS{\rightwvearrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "219D}</code>
3239	<code>\@DMS{\rightsquigarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21DD}</code>
3240	<code>\@DMS{\longrightsquigarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "27FF}</code>
3241	<code>\@DMS{\looparrowright}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21AC}</code>
3242	<code>\@DMS{\curvearrowright}</code>	<code>{\mathrel}{\M@arrows@font}{\ "293B}</code>
3243	<code>\@DMS{\circlearrowright}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21BB}</code>
3244	<code>\@DMS{\twoheadrightarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21A0}</code>
3245	<code>\@DMS{\rightarrowtail}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21E5}</code>
3246	<code>\@DMS{\rightwhitearrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21E8}</code>
3247	<code>\@DMS{\rightrightarrows}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21C9}</code>
3248	<code>\@DMS{\rightrightarrows}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21F6}</code>
3249	<code>\@DMS{\leftarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "2190}</code>
3250	<code>\global\let\from\leftarrow</code>	
3251	<code>\@DMS{\nleftarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "219A}</code>
3252	<code>\@DMS{\Leftarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21D0}</code>
3253	<code>\@DMS{\nLeftarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21CD}</code>
3254	<code>\@DMS{\Lleftarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21DA}</code>
3255	<code>\@DMS{\longleftarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "27F5}</code>
3256	<code>\@DMS{\Longleftarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "27F8}</code>
3257	<code>\@DMS{\leftbararrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21A4}</code>
3258	<code>\global\let\mapsfrom\leftbararrow</code>	
3259	<code>\@DMS{\Leftbararrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "2906}</code>
3260	<code>\@DMS{\longleftbararrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "27FB}</code>
3261	<code>\global\let\longmapsfrom\longleftbararrow</code>	
3262	<code>\@DMS{\Longleftbararrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "27FD}</code>
3263	<code>\@DMS{\hookleftarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21A9}</code>

3264	<code>\@DMS{\leftdasharrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21E0}</code>
3265	<code>\@DMS{\leftharpoonup}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21BC}</code>
3266	<code>\@DMS{\leftharpoondown}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21BD}</code>
3267	<code>\@DMS{\leftarrowtail}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21A2}</code>
3268	<code>\@DMS{\leftoplusarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "2B32}</code>
3269	<code>\@DMS{\leftwvearrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "219C}</code>
3270	<code>\@DMS{\leftsquigarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21DC}</code>
3271	<code>\@DMS{\longleftsquigarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "2B33}</code>
3272	<code>\@DMS{\looparrowleft}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21AB}</code>
3273	<code>\@DMS{\curvearrowleft}</code>	<code>{\mathrel}{\M@arrows@font}{\ "293A}</code>
3274	<code>\@DMS{\circlearrowleft}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21BA}</code>
3275	<code>\@DMS{\twoheadleftarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "219E}</code>
3276	<code>\@DMS{\leftarrowto bar}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21E4}</code>
3277	<code>\@DMS{\leftwhitearrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21E6}</code>
3278	<code>\@DMS{\leftleftarrows}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21C7}</code>
3279	<code>\@DMS{\leftleftleftarrows}</code>	<code>{\mathrel}{\M@arrows@font}{\ "2B31}</code>
3280	<code>\@DMS{\leftrightarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "2194}</code>
3281	<code>\@DMS{\Leftrightarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21D4}</code>
3282	<code>\@DMS{\nLeftrightarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21CE}</code>
3283	<code>\@DMS{\longleftrightarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "27F7}</code>
3284	<code>\@DMS{\Longleftrightarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "27FA}</code>
3285	<code>\@DMS{\leftrightwvearrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21AD}</code>
3286	<code>\@DMS{\leftrightarrows}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21C6}</code>
3287	<code>\@DMS{\leftrightharpoons}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21CB}</code>
3288	<code>\@DMS{\leftrightarrowstobar}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21B9}</code>
3289	<code>\@DMS{\rightleftarrows}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21C4}</code>
3290	<code>\@DMS{\rightleftharpoons}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21CC}</code>
3291	<code>\@DMS{\uparrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "2191}</code>
3292	<code>\@DMS{\Uparrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21D1}</code>
3293	<code>\@DMS{\Uparrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "290A}</code>
3294	<code>\@DMS{\upbararrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21A5}</code>
3295	<code>\@DMS{\updasharrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21E1}</code>
3296	<code>\@DMS{\upharpoonleft}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21BF}</code>
3297	<code>\@DMS{\upharpoonright}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21BE}</code>
3298	<code>\@DMS{\twoheaduparrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "219F}</code>
3299	<code>\@DMS{\uparrowto bar}</code>	<code>{\mathrel}{\M@arrows@font}{\ "2912}</code>
3300	<code>\@DMS{\upwhitearrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21E7}</code>
3301	<code>\@DMS{\upwhitebararrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21EA}</code>
3302	<code>\@DMS{\upuparrows}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21C8}</code>
3303	<code>\@DMS{\downarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "2193}</code>
3304	<code>\@DMS{\Downarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "21D3}</code>
3305	<code>\@DMS{\Ddownarrow}</code>	<code>{\mathrel}{\M@arrows@font}{\ "290B}</code>

```

3306 \DMS{\downbararrow}      {\mathrel}{\M@arrows@font}{21A7}
3307 \DMS{\downdasharrow}    {\mathrel}{\M@arrows@font}{21E3}
3308 \DMS{\zigzagarrow}      {\mathrel}{\M@arrows@font}{21AF}
3309 \global\let\lightningboltarrow\zigzagarrow
3310 \DMS{\downharpoonleft}    {\mathrel}{\M@arrows@font}{21C3}
3311 \DMS{\downharpoonright}   {\mathrel}{\M@arrows@font}{21C2}
3312 \DMS{\twoheaddownarrow}   {\mathrel}{\M@arrows@font}{21A1}
3313 \DMS{\downarrowto bar}    {\mathrel}{\M@arrows@font}{2913}
3314 \DMS{\downwhitearrow}     {\mathrel}{\M@arrows@font}{21E9}
3315 \DMS{\downdownarrows}     {\mathrel}{\M@arrows@font}{21CA}
3316 \DMS{\updownarrow}        {\mathrel}{\M@arrows@font}{2195}
3317 \DMS{\Updownarrow}        {\mathrel}{\M@arrows@font}{21D5}
3318 \DMS{\updownarrows}       {\mathrel}{\M@arrows@font}{21C5}
3319 \DMS{\downuparrows}       {\mathrel}{\M@arrows@font}{21F5}
3320 \DMS{\updownharpoons}     {\mathrel}{\M@arrows@font}{296E}
3321 \DMS{\downupharpoons}     {\mathrel}{\M@arrows@font}{296F}
3322 \DMS{\nearrow}             {\mathrel}{\M@arrows@font}{2197}
3323 \DMS{\Nearrow}            {\mathrel}{\M@arrows@font}{21D7}
3324 \DMS{\nrightarrow}        {\mathrel}{\M@arrows@font}{2196}
3325 \DMS{\Nrightarrow}        {\mathrel}{\M@arrows@font}{21D6}
3326 \DMS{\searrow}            {\mathrel}{\M@arrows@font}{2198}
3327 \DMS{\Searrow}            {\mathrel}{\M@arrows@font}{21D8}
3328 \DMS{\swarrow}            {\mathrel}{\M@arrows@font}{2199}
3329 \DMS{\Swarrow}            {\mathrel}{\M@arrows@font}{21D9}
3330 \DMS{\nwsearrow}          {\mathrel}{\M@arrows@font}{2921}
3331 \DMS{\neswarrow}          {\mathrel}{\M@arrows@font}{2922}
3332 \DMS{\lrcirclearrow}      {\mathrel}{\M@arrows@font}{27F2}
3333 \DMS{\rcirclearrow}      {\mathrel}{\M@arrows@font}{27F3}

```

The commands `\relbar` and `\Relbar` produce a smashed minus and an equals sign respectively. They are helper control sequences that L^AT_EX uses to create other arrows. We have a small issue with `amsmath` because in X_YL^AT_EX and Lua^AT_EX, `amsmath` defines `\relbar` and `\Relbar` in terms of the `\Umathcodes` of the minus and equals signs respectively. That is a good approach in general, but it doesn't work when a package like `mathfont` allows users to pick different fonts for symbols and arrows. We really want `\relbar` and `\Relbar` to come from the `arrows` font, so our approach is to define the control sequences now and then redefine `\AtBeginDocument` if needed.

```

3334 \let\@relbar\@undefined
3335 \let\@Relbar\@undefined
3336 \DMS{\@relbar}{\mathbin}{\M@arrows@font}{2212}
3337 \DMS{\@Relbar}{\mathrel}{\M@arrows@font}{3D}

```

```

3338 \protected\gdef\relbar{\mathrel{\mathpalette\mathsm@sh\@relbar}}
3339 \protected\gdef\Relbar{\@Relbar}

```

We redefine stuff if `amsmath` gets loaded after `mathfont`.

```

3340 \@ifpackageloaded{amsmath}
3341   {\relax}{%
3342     \global\let\@relbar\relbar
3343     \global\let\@Relbar\Relbar
3344     \AtBeginDocument{\@ifpackageloaded{amsmath}{%
3345       \global\let\relbar\@relbar
3346       \global\let\Relbar\@Relbar}
3347     {\relax}}}}

```

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

```

\M@bb@set 3348 \def\M@bb@set{%
  \mathbb 3349 \protected\def\mathbb##1{\relax
    3350   \ifmmode\else
    3351     \M@HModeError\mathbb
    3352     $%
    3353     \fi
    3354     \begingroup
    3355       \M@bb@mathcodes
    3356       ##1%
    3357     \endgroup}
\M@bb@num 3358 \edef\M@bb@num{\number
    3359   \csname symM\M@bbshape\@tempa\endcsname}
\M@bb@mathcodes 3360 \protected\edef\M@bb@mathcodes{%
    3361   \Umathcode`A=0+\M@bb@num"1D538\relax
    3362   \Umathcode`B=0+\M@bb@num"1D539\relax
    3363   \Umathcode`C=0+\M@bb@num"2102\relax
    3364   \Umathcode`D=0+\M@bb@num"1D53B\relax
    3365   \Umathcode`E=0+\M@bb@num"1D53C\relax
    3366   \Umathcode`F=0+\M@bb@num"1D53D\relax
    3367   \Umathcode`G=0+\M@bb@num"1D53E\relax
    3368   \Umathcode`H=0+\M@bb@num"210D\relax

```

```

3369 \Umathcode`I=0+\M@bb@num"1D540\relax
3370 \Umathcode`J=0+\M@bb@num"1D541\relax
3371 \Umathcode`K=0+\M@bb@num"1D542\relax
3372 \Umathcode`L=0+\M@bb@num"1D543\relax
3373 \Umathcode`M=0+\M@bb@num"1D544\relax
3374 \Umathcode`N=0+\M@bb@num"2115\relax
3375 \Umathcode`O=0+\M@bb@num"1D546\relax
3376 \Umathcode`P=0+\M@bb@num"2119\relax
3377 \Umathcode`Q=0+\M@bb@num"211A\relax
3378 \Umathcode`R=0+\M@bb@num"211D\relax
3379 \Umathcode`S=0+\M@bb@num"1D54A\relax
3380 \Umathcode`T=0+\M@bb@num"1D54B\relax
3381 \Umathcode`U=0+\M@bb@num"1D54C\relax
3382 \Umathcode`V=0+\M@bb@num"1D54D\relax
3383 \Umathcode`W=0+\M@bb@num"1D54E\relax
3384 \Umathcode`X=0+\M@bb@num"1D54F\relax
3385 \Umathcode`Y=0+\M@bb@num"1D550\relax
3386 \Umathcode`Z=0+\M@bb@num"2124\relax
3387 \Umathcode`a=0+\M@bb@num"1D552\relax
3388 \Umathcode`b=0+\M@bb@num"1D553\relax
3389 \Umathcode`c=0+\M@bb@num"1D554\relax
3390 \Umathcode`d=0+\M@bb@num"1D555\relax
3391 \Umathcode`e=0+\M@bb@num"1D556\relax
3392 \Umathcode`f=0+\M@bb@num"1D557\relax
3393 \Umathcode`g=0+\M@bb@num"1D558\relax
3394 \Umathcode`h=0+\M@bb@num"1D559\relax
3395 \Umathcode`i=0+\M@bb@num"1D55A\relax
3396 \Umathcode`j=0+\M@bb@num"1D55B\relax
3397 \Umathcode`k=0+\M@bb@num"1D55C\relax
3398 \Umathcode`l=0+\M@bb@num"1D55D\relax
3399 \Umathcode`m=0+\M@bb@num"1D55E\relax
3400 \Umathcode`n=0+\M@bb@num"1D55F\relax
3401 \Umathcode`o=0+\M@bb@num"1D560\relax
3402 \Umathcode`p=0+\M@bb@num"1D561\relax
3403 \Umathcode`q=0+\M@bb@num"1D562\relax
3404 \Umathcode`r=0+\M@bb@num"1D563\relax
3405 \Umathcode`s=0+\M@bb@num"1D564\relax
3406 \Umathcode`t=0+\M@bb@num"1D565\relax
3407 \Umathcode`u=0+\M@bb@num"1D566\relax
3408 \Umathcode`v=0+\M@bb@num"1D567\relax
3409 \Umathcode`w=0+\M@bb@num"1D568\relax
3410 \Umathcode`x=0+\M@bb@num"1D569\relax

```

```

3411 \Umathcode`y=0+\M@bb@num"1D56A\relax
3412 \Umathcode`z=0+\M@bb@num"1D56B\relax
3413 \Umathcode`0=0+\M@bb@num"1D7D8\relax
3414 \Umathcode`1=0+\M@bb@num"1D7D9\relax
3415 \Umathcode`2=0+\M@bb@num"1D7DA\relax
3416 \Umathcode`3=0+\M@bb@num"1D7DB\relax
3417 \Umathcode`4=0+\M@bb@num"1D7DC\relax
3418 \Umathcode`5=0+\M@bb@num"1D7DD\relax
3419 \Umathcode`6=0+\M@bb@num"1D7DE\relax
3420 \Umathcode`7=0+\M@bb@num"1D7DF\relax
3421 \Umathcode`8=0+\M@bb@num"1D7E0\relax
3422 \Umathcode`9=0+\M@bb@num"1D7E1\relax}}

```

Set caligraphic letters.

```

\M@cal@set 3423 \def\M@cal@set{%
  \mathcal 3424 \protected\def\mathcal##1{\relax
    3425 \ifmmode\else
    3426 \M@HModeError\mathcal
    3427 $%
    3428 \fi
    3429 \begingroup
    3430 \M@cal@mathcodes
    3431 ##1%
    3432 \endgroup}
\M@cal@num 3433 \edef\M@cal@num{\number
  3434 \csname symM\M@cal@shape\@tempa\endcsname}
\M@cal@mathcodes 3435 \protected\edef\M@cal@mathcodes{%
  3436 \Umathcode`A=0+\M@cal@num"1D49C\relax
  3437 \Umathcode`B=0+\M@cal@num"212C\relax
  3438 \Umathcode`C=0+\M@cal@num"1D49E\relax
  3439 \Umathcode`D=0+\M@cal@num"1D49F\relax
  3440 \Umathcode`E=0+\M@cal@num"2130\relax
  3441 \Umathcode`F=0+\M@cal@num"2131\relax
  3442 \Umathcode`G=0+\M@cal@num"1D4A2\relax
  3443 \Umathcode`H=0+\M@cal@num"210B\relax
  3444 \Umathcode`I=0+\M@cal@num"2110\relax
  3445 \Umathcode`J=0+\M@cal@num"1D4A5\relax
  3446 \Umathcode`K=0+\M@cal@num"1D4A6\relax
  3447 \Umathcode`L=0+\M@cal@num"2112\relax
  3448 \Umathcode`M=0+\M@cal@num"2133\relax
  3449 \Umathcode`N=0+\M@cal@num"1D4A9\relax
  3450 \Umathcode`O=0+\M@cal@num"1D4AA\relax
  3451 \Umathcode`P=0+\M@cal@num"1D4AB\relax

```

```

3452 \Umathcode`Q=0+\M@cal@num"1D4AC\relax
3453 \Umathcode`R=0+\M@cal@num"211B\relax
3454 \Umathcode`S=0+\M@cal@num"1D4AE\relax
3455 \Umathcode`T=0+\M@cal@num"1D4AF\relax
3456 \Umathcode`U=0+\M@cal@num"1D4B0\relax
3457 \Umathcode`V=0+\M@cal@num"1D4B1\relax
3458 \Umathcode`W=0+\M@cal@num"1D4B2\relax
3459 \Umathcode`X=0+\M@cal@num"1D4B3\relax
3460 \Umathcode`Y=0+\M@cal@num"1D4B4\relax
3461 \Umathcode`Z=0+\M@cal@num"1D4B5\relax
3462 \Umathcode`a=0+\M@cal@num"1D4B6\relax
3463 \Umathcode`b=0+\M@cal@num"1D4B7\relax
3464 \Umathcode`c=0+\M@cal@num"1D4B8\relax
3465 \Umathcode`d=0+\M@cal@num"1D4B9\relax
3466 \Umathcode`e=0+\M@cal@num"212F\relax
3467 \Umathcode`f=0+\M@cal@num"1D4BB\relax
3468 \Umathcode`g=0+\M@cal@num"210A\relax
3469 \Umathcode`h=0+\M@cal@num"1D4BD\relax
3470 \Umathcode`i=0+\M@cal@num"1D4BE\relax
3471 \Umathcode`j=0+\M@cal@num"1D4BF\relax
3472 \Umathcode`k=0+\M@cal@num"1D4C0\relax
3473 \Umathcode`l=0+\M@cal@num"1D4C1\relax
3474 \Umathcode`m=0+\M@cal@num"1D4C2\relax
3475 \Umathcode`n=0+\M@cal@num"1D4C3\relax
3476 \Umathcode`o=0+\M@cal@num"2134\relax
3477 \Umathcode`p=0+\M@cal@num"1D4C5\relax
3478 \Umathcode`q=0+\M@cal@num"1D4C6\relax
3479 \Umathcode`r=0+\M@cal@num"1D4C7\relax
3480 \Umathcode`s=0+\M@cal@num"1D4C8\relax
3481 \Umathcode`t=0+\M@cal@num"1D4C9\relax
3482 \Umathcode`u=0+\M@cal@num"1D4CA\relax
3483 \Umathcode`v=0+\M@cal@num"1D4CB\relax
3484 \Umathcode`w=0+\M@cal@num"1D4CC\relax
3485 \Umathcode`x=0+\M@cal@num"1D4CD\relax
3486 \Umathcode`y=0+\M@cal@num"1D4CE\relax
3487 \Umathcode`z=0+\M@cal@num"1D4CF\relax}}

```

Set fraktur letters.

```

\M@frak@set 3488 \def\M@frak@set{%
\mathfrak 3489 \protected\def\mathfrak##1{\relax
3490 \ifmmode\else
3491 \M@HModeError\mathfrak
3492 $%

```

```

3493     \fi
3494     \begingroup
3495         \M@frak@mathcodes
3496         ##1%
3497     \endgroup}
\M@frak@num 3498 \edef\M@frak@num{\number
3499     \csname symM\M@frakshape\@tempa\endcsname}
\M@frak@mathcodes 3500 \protected\edef\M@frak@mathcodes{%
3501     \Umathcode`A=0+\M@frak@num"1D504\relax
3502     \Umathcode`B=0+\M@frak@num"1D505\relax
3503     \Umathcode`C=0+\M@frak@num"212D\relax
3504     \Umathcode`D=0+\M@frak@num"1D507\relax
3505     \Umathcode`E=0+\M@frak@num"1D508\relax
3506     \Umathcode`F=0+\M@frak@num"1D509\relax
3507     \Umathcode`G=0+\M@frak@num"1D50A\relax
3508     \Umathcode`H=0+\M@frak@num"210C\relax
3509     \Umathcode`I=0+\M@frak@num"2111\relax
3510     \Umathcode`J=0+\M@frak@num"1D50D\relax
3511     \Umathcode`K=0+\M@frak@num"1D50E\relax
3512     \Umathcode`L=0+\M@frak@num"1D50F\relax
3513     \Umathcode`M=0+\M@frak@num"1D510\relax
3514     \Umathcode`N=0+\M@frak@num"1D511\relax
3515     \Umathcode`O=0+\M@frak@num"1D512\relax
3516     \Umathcode`P=0+\M@frak@num"1D513\relax
3517     \Umathcode`Q=0+\M@frak@num"1D514\relax
3518     \Umathcode`R=0+\M@frak@num"211C\relax
3519     \Umathcode`S=0+\M@frak@num"1D516\relax
3520     \Umathcode`T=0+\M@frak@num"1D517\relax
3521     \Umathcode`U=0+\M@frak@num"1D518\relax
3522     \Umathcode`V=0+\M@frak@num"1D519\relax
3523     \Umathcode`W=0+\M@frak@num"1D51A\relax
3524     \Umathcode`X=0+\M@frak@num"1D51B\relax
3525     \Umathcode`Y=0+\M@frak@num"1D51C\relax
3526     \Umathcode`Z=0+\M@frak@num"2128\relax
3527     \Umathcode`a=0+\M@frak@num"1D51E\relax
3528     \Umathcode`b=0+\M@frak@num"1D51F\relax
3529     \Umathcode`c=0+\M@frak@num"1D520\relax
3530     \Umathcode`d=0+\M@frak@num"1D521\relax
3531     \Umathcode`e=0+\M@frak@num"1D522\relax
3532     \Umathcode`f=0+\M@frak@num"1D523\relax
3533     \Umathcode`g=0+\M@frak@num"1D524\relax
3534     \Umathcode`h=0+\M@frak@num"1D525\relax

```

```

3535 \Umathcode`i=0+\M@frak@num"1D526\relax
3536 \Umathcode`j=0+\M@frak@num"1D527\relax
3537 \Umathcode`k=0+\M@frak@num"1D528\relax
3538 \Umathcode`l=0+\M@frak@num"1D529\relax
3539 \Umathcode`m=0+\M@frak@num"1D52A\relax
3540 \Umathcode`n=0+\M@frak@num"1D52B\relax
3541 \Umathcode`o=0+\M@frak@num"1D52C\relax
3542 \Umathcode`p=0+\M@frak@num"1D52D\relax
3543 \Umathcode`q=0+\M@frak@num"1D52E\relax
3544 \Umathcode`r=0+\M@frak@num"1D52F\relax
3545 \Umathcode`s=0+\M@frak@num"1D530\relax
3546 \Umathcode`t=0+\M@frak@num"1D531\relax
3547 \Umathcode`u=0+\M@frak@num"1D532\relax
3548 \Umathcode`v=0+\M@frak@num"1D533\relax
3549 \Umathcode`w=0+\M@frak@num"1D534\relax
3550 \Umathcode`x=0+\M@frak@num"1D535\relax
3551 \Umathcode`y=0+\M@frak@num"1D536\relax
3552 \Umathcode`z=0+\M@frak@num"1D537\relax}}

```

Set bold caligraphic letters.

```

\M@bcal@set 3553 \def\M@bcal@set{%
  \mathbcal 3554 \protected\def\mathbcal##1{\relax
    3555 \ifmmode\else
    3556 \M@HModeError\mathbcal
    3557 $%
    3558 \fi
    3559 \begingroup
    3560 \M@bcal@mathcodes
    3561 ##1%
    3562 \endgroup}
\M@bcal@num 3563 \edef\M@bcal@num{\number
  3564 \csname symM\M@bcalshape\@tempa\endcsname}
\M@bcal@mathcodes 3565 \protected\edef\M@bcal@mathcodes{%
  3566 \Umathcode`A=0+\M@bcal@num"1D4D0\relax
  3567 \Umathcode`B=0+\M@bcal@num"1D4D1\relax
  3568 \Umathcode`C=0+\M@bcal@num"1D4D2\relax
  3569 \Umathcode`D=0+\M@bcal@num"1D4D3\relax
  3570 \Umathcode`E=0+\M@bcal@num"1D4D4\relax
  3571 \Umathcode`F=0+\M@bcal@num"1D4D5\relax
  3572 \Umathcode`G=0+\M@bcal@num"1D4D6\relax
  3573 \Umathcode`H=0+\M@bcal@num"1D4D7\relax
  3574 \Umathcode`I=0+\M@bcal@num"1D4D8\relax
  3575 \Umathcode`J=0+\M@bcal@num"1D4D9\relax

```



```
3576    \Umathcode`K=0+\M@bcal@num"1D4DA\relax
3577    \Umathcode`L=0+\M@bcal@num"1D4DB\relax
3578    \Umathcode`M=0+\M@bcal@num"1D4DC\relax
3579    \Umathcode`N=0+\M@bcal@num"1D4DD\relax
3580    \Umathcode`O=0+\M@bcal@num"1D4DE\relax
3581    \Umathcode`P=0+\M@bcal@num"1D4DF\relax
3582    \Umathcode`Q=0+\M@bcal@num"1D4E0\relax
3583    \Umathcode`R=0+\M@bcal@num"1D4E1\relax
3584    \Umathcode`S=0+\M@bcal@num"1D4E2\relax
3585    \Umathcode`T=0+\M@bcal@num"1D4E3\relax
3586    \Umathcode`U=0+\M@bcal@num"1D4E4\relax
3587    \Umathcode`V=0+\M@bcal@num"1D4E5\relax
3588    \Umathcode`W=0+\M@bcal@num"1D4E6\relax
3589    \Umathcode`X=0+\M@bcal@num"1D4E7\relax
3590    \Umathcode`Y=0+\M@bcal@num"1D4E8\relax
3591    \Umathcode`Z=0+\M@bcal@num"1D4E9\relax
3592    \Umathcode`a=0+\M@bcal@num"1D4EA\relax
3593    \Umathcode`b=0+\M@bcal@num"1D4EB\relax
3594    \Umathcode`c=0+\M@bcal@num"1D4EC\relax
3595    \Umathcode`d=0+\M@bcal@num"1D4ED\relax
3596    \Umathcode`e=0+\M@bcal@num"1D4EE\relax
3597    \Umathcode`f=0+\M@bcal@num"1D4EF\relax
3598    \Umathcode`g=0+\M@bcal@num"1D4F0\relax
3599    \Umathcode`h=0+\M@bcal@num"1D4F1\relax
3600    \Umathcode`i=0+\M@bcal@num"1D4F2\relax
3601    \Umathcode`j=0+\M@bcal@num"1D4F3\relax
3602    \Umathcode`k=0+\M@bcal@num"1D4F4\relax
3603    \Umathcode`l=0+\M@bcal@num"1D4F5\relax
3604    \Umathcode`m=0+\M@bcal@num"1D4F6\relax
3605    \Umathcode`n=0+\M@bcal@num"1D4F7\relax
3606    \Umathcode`o=0+\M@bcal@num"1D4F8\relax
3607    \Umathcode`p=0+\M@bcal@num"1D4F9\relax
3608    \Umathcode`q=0+\M@bcal@num"1D4FA\relax
3609    \Umathcode`r=0+\M@bcal@num"1D4FB\relax
3610    \Umathcode`s=0+\M@bcal@num"1D4FC\relax
3611    \Umathcode`t=0+\M@bcal@num"1D4FD\relax
3612    \Umathcode`u=0+\M@bcal@num"1D4FE\relax
3613    \Umathcode`v=0+\M@bcal@num"1D4FF\relax
3614    \Umathcode`w=0+\M@bcal@num"1D500\relax
3615    \Umathcode`x=0+\M@bcal@num"1D501\relax
3616    \Umathcode`y=0+\M@bcal@num"1D502\relax
3617    \Umathcode`z=0+\M@bcal@num"1D503\relax}}}
```

Set bold fraktur letters.

```

\M@bfrak@set 3618 \def\M@bfrak@set{%
  \mathbfrak 3619 \protected\def\mathbfrak##1{\relax
    3620 \ifmmode\else
    3621 \M@HModeError\mathbfrak
    3622 $%
    3623 \fi
    3624 \begingroup
    3625 \M@bfrak@mathcodes
    3626 ##1%
    3627 \endgroup}
\M@bfrak@num 3628 \edef\M@bfrak@num{\number
    3629 \csname symM\M@bfrakshape\@tempa\endcsname}
\M@bfrak@mathcodes 3630 \protected\edef\M@bfrak@mathcodes{%
    3631 \Umathcode`A=0+\M@bfrak@num"1D56C\relax
    3632 \Umathcode`B=0+\M@bfrak@num"1D56D\relax
    3633 \Umathcode`C=0+\M@bfrak@num"1D56E\relax
    3634 \Umathcode`D=0+\M@bfrak@num"1D56F\relax
    3635 \Umathcode`E=0+\M@bfrak@num"1D570\relax
    3636 \Umathcode`F=0+\M@bfrak@num"1D571\relax
    3637 \Umathcode`G=0+\M@bfrak@num"1D572\relax
    3638 \Umathcode`H=0+\M@bfrak@num"1D573\relax
    3639 \Umathcode`I=0+\M@bfrak@num"1D574\relax
    3640 \Umathcode`J=0+\M@bfrak@num"1D575\relax
    3641 \Umathcode`K=0+\M@bfrak@num"1D576\relax
    3642 \Umathcode`L=0+\M@bfrak@num"1D577\relax
    3643 \Umathcode`M=0+\M@bfrak@num"1D578\relax
    3644 \Umathcode`N=0+\M@bfrak@num"1D579\relax
    3645 \Umathcode`O=0+\M@bfrak@num"1D57A\relax
    3646 \Umathcode`P=0+\M@bfrak@num"1D57B\relax
    3647 \Umathcode`Q=0+\M@bfrak@num"1D57C\relax
    3648 \Umathcode`R=0+\M@bfrak@num"1D57D\relax
    3649 \Umathcode`S=0+\M@bfrak@num"1D57E\relax
    3650 \Umathcode`T=0+\M@bfrak@num"1D57F\relax
    3651 \Umathcode`U=0+\M@bfrak@num"1D580\relax
    3652 \Umathcode`V=0+\M@bfrak@num"1D581\relax
    3653 \Umathcode`W=0+\M@bfrak@num"1D582\relax
    3654 \Umathcode`X=0+\M@bfrak@num"1D583\relax
    3655 \Umathcode`Y=0+\M@bfrak@num"1D584\relax
    3656 \Umathcode`Z=0+\M@bfrak@num"1D585\relax
    3657 \Umathcode`a=0+\M@bfrak@num"1D586\relax
    3658 \Umathcode`b=0+\M@bfrak@num"1D587\relax

```

```

3659    \Umathcode`c=0+\M@bfrak@num"1D588\relax
3660    \Umathcode`d=0+\M@bfrak@num"1D589\relax
3661    \Umathcode`e=0+\M@bfrak@num"1D58A\relax
3662    \Umathcode`f=0+\M@bfrak@num"1D58B\relax
3663    \Umathcode`g=0+\M@bfrak@num"1D58C\relax
3664    \Umathcode`h=0+\M@bfrak@num"1D58D\relax
3665    \Umathcode`i=0+\M@bfrak@num"1D58E\relax
3666    \Umathcode`j=0+\M@bfrak@num"1D58F\relax
3667    \Umathcode`k=0+\M@bfrak@num"1D590\relax
3668    \Umathcode`l=0+\M@bfrak@num"1D591\relax
3669    \Umathcode`m=0+\M@bfrak@num"1D592\relax
3670    \Umathcode`n=0+\M@bfrak@num"1D593\relax
3671    \Umathcode`o=0+\M@bfrak@num"1D594\relax
3672    \Umathcode`p=0+\M@bfrak@num"1D595\relax
3673    \Umathcode`q=0+\M@bfrak@num"1D596\relax
3674    \Umathcode`r=0+\M@bfrak@num"1D597\relax
3675    \Umathcode`s=0+\M@bfrak@num"1D598\relax
3676    \Umathcode`t=0+\M@bfrak@num"1D599\relax
3677    \Umathcode`u=0+\M@bfrak@num"1D59A\relax
3678    \Umathcode`v=0+\M@bfrak@num"1D59B\relax
3679    \Umathcode`w=0+\M@bfrak@num"1D59C\relax
3680    \Umathcode`x=0+\M@bfrak@num"1D59D\relax
3681    \Umathcode`y=0+\M@bfrak@num"1D59E\relax
3682    \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.

- | | |
|--|--|
| <p>1.1b July 2018</p> <ul style="list-style-type: none"> —initial release <p>1.2 August 2018</p> <ul style="list-style-type: none"> —minor bug fix for <code>\mathfrak</code> —eliminated redundant batchfile <p>1.3 January 2019</p> <ul style="list-style-type: none"> —added <code>symbols</code> keyword —created <code>mathfont_example.pdf</code> —corrected the description of the <code>mathastext</code> package —font-change <code>\message</code> added to <code>\mathfont</code> <p>1.4 April 2019</p> <ul style="list-style-type: none"> —<code>\setfont</code> command added —<code>\mathfont</code> optional argument can parse spaces —<code>no-operators</code> now default package optional argument —added <code>\comma</code> command —new fancy fatal error message —improved messaging for <code>\mathfont</code> —internal command <code>\mathpound</code> changed to <code>\mathhash</code> —added a missing <code>#1</code> after <code>\char`\"</code> in the example code —redefining <code>"</code> in the user guide <p>1.5 April 2019</p> <ul style="list-style-type: none"> —separated <code>\increment</code> and <code>\Delta</code> —version history added —initial off-the-shelf use insert added | <p>1.6 December 2019</p> <ul style="list-style-type: none"> —separated implementation and user documentation —created <code>mathfont_heading.tex</code> —created <code>mathfont_doc_patch.tex</code> for use with the index —changed <code>mathfont_greek.pdf</code> to <code>mathfont_symbol_list.pdf</code> —eliminated <code>mathfont_example.pdf</code> —eliminated <code>operators</code> package option —eliminated <code>packages</code> package option —font name can be package option —added Hebrew and Cyrillic characters —separated ancient Greek from modern Greek characters —created new keywords: <code>extsymbols</code>, <code>delimiters</code>, <code>arrows</code>, <code>diacritics</code>, <code>bigops</code>, <code>extbigops</code> —improved messaging —improved internal code for local font-change commands —improved space parsing for the optional argument of <code>\mathfont</code> —bug fix for <code>\#</code>, etc. commands —bad input for <code>\mathbb</code>, etc. now gives a warning —improved error checking for <code>\newmathrm</code>, etc. commands —<code>\mathfont</code> now ignores bad options (on top of issuing an error) —internal commands now begin with <code>\M@...</code> |
|--|--|

- added Easter Egg!
- improved indexing
- `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 LuaTeX: new glyph boundaries for Latin letters in math mode, resizable delimiters, 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 `\Umathcodes` of letters instead of `\M@<bb,etc.>@<letter>` commands
- removed warnings about non-letter contents of `\mathbb`, etc.

- fonts loaded twice, once with default settings (for text) and once in base mode (for math)
 - `\mathconstantsfont` accepts “upright” or “italic” as optional argument
- 2.2** December 2022
- changed the easter egg text
 - updated patch for `\DeclareSymbolFont` to work with changes to the kernel (eliminated `\M@p@tch@decl@re` error message)
 - calling Plain \TeX on `mathfont_code.dtx` produces sty file and no pdf file
- 2.2a** December 2022
- bug fix for `\mathconstantsfont`
 - bug fix for `\M@check@int`
 - added `doc2` option to `ltxdoc` in `mathfont_code.dtx`
- 2.2b** August 2023
- minor changes to code and documentation
 - `\ng` now works in math (as not greater than symbol) and text (as pronunciation symbol)
- 2.3** September 2023
- solidus and `\fractionslash` are `\mathord` instead of `\mathbin`
 - removed `\mathfont{fontspec}` functionality
 - redesigned font-loader
 - added package options `default-loader` and `fontspec-loader`
- 2.4** April 2025
- `\colon` is `\mathpunct` instead of `\mathord`
 - moved `\relbar` and `\Relbar` to **arrows**
 - reformatted `mathfont_code.pdf`
 - made compatible with `\mathbb`, etc. commands from other packages
 - renamed `set_nomath_true` to `set_nomath_false`
 - improved messaging `\AtBeginDocument`
 - removed deprecated package options, `\newmathbold`, `\restoremathinternals`
 - more Easter egg messages

Index

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

Symbols			
<code>\#</code>	2989	<code>\approxeq</code>	3109
<code>\%</code>	2990	<code>\arceq</code>	3116
<code>\&</code>	2991	<code>\asymp</code>	3137
<code>*</code>	295, 310, 314, 318	<code>\ayin</code>	2705
<code>\-</code>	2924	B	
<code>\/</code>	296, 311, 315, 319	Bad argument for	<i>p. 18</i>
<code>\=</code>	297, 324–326, 1408, 2925	<code>\bar</code>	2515
<code>\@@Relbar</code>	2932, 2936, 3343, 3346	<code>\bclubsuit</code>	3041, 3052
<code>\@relbar</code>	2931, 2935, 3342, 3345	<code>\bdiamondsuit</code>	3042
<code>\@Relbar</code>	2925, 2928, 3335, 3337, 3339	<code>\because</code>	3094
<code>\@mathconstantsfont</code>	1010, 1011	<code>\Beta</code>	2263, 2522
<code>\@optionpresentfalse</code>	886, 899	<code>\beta</code>	2233, 2566
<code>\@optionpresenttrue</code>	881	<code>\beth</code>	2691
<code>\@percentchar</code>	1548, 1624, 1710, 1800	<code>\bheartsuit</code>	3043
<code>\@relbar</code>	2924, 2927, 3334, 3336, 3338	<code>\bigand</code>	3005
<code>\@secondoftwo</code>	1103	<code>\bigat</code>	3001
<code>\@sqrts@gn</code>	2857 , 2861, 2875	<code>\bigcap</code>	2320, 2894, 2903
<code>\@tempbase</code>	764, 789 , 798, 799, 818, 821, 833, 841, 851, 854, 856, 860, 968, 974, 975, 981, 995, 1090, 1092, 1093, 1104, 1105	<code>\bigcup</code>	2319, 2893, 2902
<code>\@tempfeatures</code>	790 , 795, 796, 796 , 838, 846, 854, 859	<code>\bigdiv</code>	3011
<code>\@whilenum</code>	1319	<code>\bigdollar</code>	3003
<code>\~</code>	1407	<code>\bighash</code>	3002
 		<code>\bigodot</code>	2323, 2897, 2911
<code>\sqcup</code>	119, 162	<code>\bigoplus</code>	2321, 2895, 2909
A		<code>\bigotimes</code>	2322, 2896, 2910
<code>\aacute</code>	2508	<code>\bigp</code>	3007
<code>\acute</code>	2507	<code>\bigpercent</code>	3004
<code>\aleph</code>	2690	<code>\bigplus</code>	3006
<code>\Alpha</code>	2262, 2521	<code>\bigq</code>	3008
<code>\amalg</code>	3059	<code>\bigS</code>	3009
<code>\angle</code>	3016, 3031	<code>\bigsqcap</code>	2324, 2912
<code>\approx</code>	2978	<code>\bigsqcup</code>	2325, 2898, 2913
		<code>\bigtimes</code>	3000, 3010
		<code>\bigvee</code>	2317, 2891, 2900
		<code>\bigwedge</code>	2318, 2892, 2901
		<code>\bot</code>	3033
		<code>\bowtie</code>	3020, 3091
		<code>\breve</code>	2512

`\bspadesuit` 3044, 3049
`\bullet` 2967

C

cannot find the file `luaotfload` . *p. 5*
 catcode changes *p. 4*
`\cdot` 2970
`\CharmLine` 83,
 580, 586, 1176, 1186, **1435**, 1465
`\check` 2514
`\Chi` 2283, **2520**, 2542
`\chi` 2253, 2586
`\circlearrowleft` 3274
`\circlearrowright` 3243
`\clubsuit` 3052
`\colon` 2921, 2987
`\coloneq` 3113
`\comma` 2958
`\cong` 2995, 3108
`\coprod` 2316, 2890, 2899
`\curvearrowleft` 3273
`\curvearrowright` 3242

D

`\dagger` 2968
`\daleth` 2693
`\dashv` 3035
`\ddagger` 2969
`\ddot` 2510
`\Ddownarrow` 3305
`\DeclareFontFamily` 836, 844
`\DeclareFontShape` 764
`\DeclareMathAlphabet` 1091
 default font changes *p. 32*
`\defeq` 3121
`\degree` 2956
`\Delta` 2265, 2524
`\delta` 2235, 2568
`\diamond` 3063
`\diamondsuit` 3051
`\Digamma` 2600
`\digamma` 2612
`\div` 2313, 2965

`\dot` 2509
`\doteq` 3021, 3112
`\Downarrow` 3206, 3304
`\downarrow` 3205, 3303
`\downarrowto` 3313
`\downbararrow` 3306
`\downdasharrow` 3307
`\downdownarrows` 3315
`\downharpoonleft` 3310
`\downharpoonright` 3311
`\downuparrows` 3319
`\downupharpoons` 3321
`\downwhitearrow` 3314

E

`\ell` 3024
`\emptyset` 3027
`\Epsilon` 2266, 2525
`\epsilon` 2236, 2569
`\eqcolon` 3114
`\eqsim` 3106
`\equiv` 2979
`\Eta` 2268, 2527
`\eta` 2238, 2571
`\exists` 3026

F

`\fakelangle` 2299, 2831
`\fakellangle` 2301, 2835
`\fakerrangle` 2300, 2833
`\fakerrangle` 2302, 2837
`\fflat` 3039
`\flat` 3036
`\forall` 3025
 Forbidden charm info *p. 18*
`\fractionslash` 2964

G

`\Gamma` 2264, 2523
`\gamma` 2234, 2567
`\gapprox` 3104
`\geq` 2976
`\geqq` 3102

- | | | | |
|--|------------------------|--|--------------------------------|
| <code>\getanddefine@fonts</code> | 1038 | <code>\increment</code> | 2548, 2555, 2957 |
| <code>\ggg</code> | 3100 | <code>\infty</code> | 2954 |
| <code>\gimel</code> | 2692 | <code>\IntegralItalicFactor</code> | |
| <code>\gnapprox</code> | 3172 | | 80, 1160, 1175, 1183 |
| <code>\gneq</code> | 3155 | <code>\intop</code> | 2327, 2887 |
| <code>\gneqq</code> | 3157 | invalid command | <i>p. 2</i> |
| <code>\gnsim</code> | 3170 | Invalid font specifier | <i>p. 16</i> |
| <code>\grave</code> | 2511 | Invalid option for <code>\mathfont</code> . . | <i>p. 15</i> |
| <code>\gsim</code> | 3124 | Invalid suboption for <code>\mathfont</code> . | <i>p. 15</i> |
| H | | | |
| <code>\hat</code> | 2513 | <code>\Iota</code> | 2270, 2529 |
| <code>\hbar</code> | 2471, 2503 | <code>\iota</code> | 2240, 2573 |
| <code>\heartsuit</code> | 3050 | J | |
| <code>\het</code> | 2697 | <code>\jmath</code> | 2205, 2470, 2502, 2787 |
| <code>\Heta</code> | 2598 | K | |
| <code>\heta</code> | 2610 | <code>\kaf</code> | 2700 |
| <code>\hookleftarrow</code> | 3213, 3263 | <code>\Kappa</code> | 2271, 2530 |
| <code>\hookrightarrow</code> | 3212, 3232 | <code>\kappa</code> | 2241, 2574 |
| <code>\hourglass</code> | 3092 | <code>\kern</code> | 2868, 2870, 2873 |
| I | | | |
| I already set the font | <i>p. 13, p. 33</i> | keyword options for <code>\mathfont</code> . . | |
| I couldn't find a base-mode . . . | <i>p. 13</i> | | <i>p. 23, p. 30, p. 32, 33</i> |
| if-parser | <i>p. 33</i> | keyword <code>agreeklower</code> | <i>p. 95</i> |
| <code>\if@tempwa</code> | 1107 | keyword <code>agreekupper</code> | <i>p. 95</i> |
| <code>\ifdim</code> | 2867 | keyword <code>arrows</code> | <i>p. 111</i> |
| <code>\ifM@adjust@font</code> | | keyword <code>bb</code> | <i>p. 115</i> |
| 67, 408, 420, 758, 1051, 1147, | | keyword <code>bcal</code> | <i>p. 120</i> |
| 1208, 1404, 2376, 2381, 2440, | | keyword <code>bfrac</code> | <i>p. 122</i> |
| 2546, 2730, 2794, 2853, 2999, 3198 | | keyword <code>bigops</code> | <i>p. 102</i> |
| <code>\ifM@arg@good</code> | 724, 1088, 1164 | keyword <code>cal</code> | <i>p. 117</i> |
| <code>\ifM@Decl@reF@mily</code> | 725 | keyword <code>cyrilliclower</code> | <i>p. 96</i> |
| <code>\ifM@font@loaded</code> | 68, 1365, 1377 | keyword <code>cyrillicupper</code> | <i>p. 96</i> |
| <code>\ifM@fromCharmFile</code> | 726, 1441, 1449 | keyword <code>delimiters</code> | <i>p. 100</i> |
| <code>\ifM@NoLuaotfload</code> | 66, 160 | keyword <code>diacritics</code> | <i>p. 93</i> |
| <code>\ifM@XeTeXLuaTeX</code> | 65, 117 | keyword <code>digits</code> | <i>p. 98</i> |
| <code>\iiint</code> | 2329, 2915 | keyword <code>extbigops</code> | <i>p. 103</i> |
| <code>\iiintop</code> | 2905, 2915 | keyword <code>extsymbols</code> | <i>p. 106</i> |
| <code>\iint</code> | 2328, 2914 | keyword <code>frak</code> | <i>p. 118</i> |
| <code>\iintop</code> | 2904, 2914 | keyword <code>greeklower</code> | <i>p. 94</i> |
| <code>\Im</code> | 3191, 3194 | keyword <code>greekupper</code> | <i>p. 93</i> |
| <code>\imath</code> | 2204, 2469, 2501, 2786 | keyword <code>hebrew</code> | <i>p. 97</i> |

keyword radical	<i>p. 102</i>
keyword symbols	<i>p. 103</i>
keyword upper	<i>p. 90</i>
\Koppa	2601
\koppa	2613

L

<code>\Lambda</code>	2272, 2531
<code>\lambda</code>	2575
<code>\lamed</code>	2701
<code>\lapprox</code>	3103
<code>L^AT_EX kernel</code>	<i>p. 19, p. 37, p. 102</i>
<code>\lbrace</code>	2819
<code>\lcirclearrow</code>	3332
<code>\Leftarrow</code>	3252
<code>\leftarrow</code>	3249, 3250
<code>\leftarrowtail</code>	3267
<code>\leftarrowto</code>	3276
<code>\Leftbararrow</code>	3259
<code>\leftbararrow</code>	3257, 3258
<code>\leftbrace</code>	2803, 2850
<code>\leftdasharrow</code>	3264
<code>\leftharpoondown</code>	3266
<code>\leftharpoonup</code>	3265
<code>\leftleftarrows</code>	3278
<code>\leftleftleftarrows</code>	3279
<code>\leftoplusarrow</code>	3268
<code>\Leftrightarrow</code>	3281
<code>\leftrightarrow</code>	3280
<code>\leftrightarrows</code>	3286
<code>\leftrightarrowsto</code>	3288
<code>\leftrightharpoons</code>	3287
<code>\leftrighthvewarrow</code>	3285
<code>\leftsquigarrow</code>	3270
<code>\leftwvearrow</code>	3269
<code>\leftwhitearrow</code>	3277
<code>\leq</code>	2975
<code>\leqq</code>	3101
<code>\lguil</code>	1812, 2295, 2823, 2846
<code>\lightningboltarrow</code>	3309
<code>\Lleftarrow</code>	3254
<code>\llguil</code>	1814, 2297, 2827, 2848

<code>\lil</code>	3099
<code>\lnapprox</code>	3171
<code>\lneq</code>	3154
<code>\lneqq</code>	3156
<code>\lnsim</code>	3169
local font changes	<i>p. 37</i>
log file	<i>p. 28, p. 34, 35, p. 43, p. 46</i>
<code>\long</code>	71, 927
<code>\Longleftarrow</code>	3215, 3256
<code>\longleftarrow</code>	3210, 3255
<code>\Longleftbararrow</code>	3262
<code>\longleftbararrow</code>	3260, 3261
<code>\Longleftrightarrow</code>	3216, 3284
<code>\longleftrightarrow</code>	3211, 3283
<code>\longleftsquigarrow</code>	3271
<code>\longmapsfrom</code>	3261
<code>\longmapsto</code>	3230
<code>\Longrightarrow</code>	3214, 3225
<code>\longrightarrow</code>	3209, 3224
<code>\Longrightbararrow</code>	3231
<code>\longrightbararrow</code>	3229, 3230
<code>\longrightsquigarrow</code>	3240
<code>\looparrowleft</code>	3272
<code>\looparrowright</code>	3241
<code>\lsim</code>	3123

M

```

\M@addto@localfonts 978, 1093, 1096
\M@agreeklower@set 984, 1386, 2608
\M@agreeklowershape .. 733, 2609
\M@agreekupper@set 984, 1385, 2596
\M@agreekuppershape ... 732, 2597
\M@arrows@set .... 984, 1398, 3201
\M@arrowsshape ..... 745, 3202
\M@BadIntegerError ... 620, 1170
\M@BadMathConstantsFontError .
..... 511, 1015
\M@BadMathConstantsFontTypeError
..... 523, 1026
\M@bb@mathcodes ..... 3355, 3360
\M@bb@num ..... 3358, 3361–3422
\M@bb@set ..... 984, 1399, 3348

```

- \M@bbshape **746**, 3359
- \M@bcal@mathcodes 3560, **3565**
- \M@bcal@num **3563**, 3566–3617
- \M@bcal@set 984, 1402, **3553**
- \M@bcalshape **749**, 3564
- \M@bfrak@mathcodes . . . 3625, **3630**
- \M@bfrak@num **3628**, 3631–3682
- \M@bfrak@set 984, 1403, **3618**
- \M@bfrakshape **750**, 3629
- \M@bigops@set 984, 1394, **2881**
- \M@bigopsshape **741**, 2882
- \M@cal@mathcodes 3430, **3435**
- \M@cal@num **3433**, 3436–3487
- \M@cal@set 984, 1400, **3423**
- \M@calshape **747**, 3434
- \M@Charm 689, 1462, 1464, 1466, 1470
- \M@CharsSetWarning **443**, 949
- \M@check@csarg . . **1056**, 1087, **1114**
- \M@check@int **1148**, 1163
- \M@check@nfss@shapes **776**, 800, 809
- \M@check@option@valid . . **876**, 906
- \M@check@suboption@valid **889**, 916
- \M@checkspecials **1068**, 1092
- \M@cyrilliclower@set
. 984, 1388, **2654**
- \M@cyrilliclowershape . **735**, 2655
- \M@cyrillicupper@set
. 984, 1387, **2620**
- \M@cyrillicuppershape . . **734**, 2621
- \M@declare@shape . . . **761**, 775, 873
- \M@DecSymDef **668**, 671, 676
- \M@default@newmath@cmds
. **1116**, 1125, **1136**
- \M@defaultkeys . **756**, 759, **759**, 930
- \M@define@newmath@cmd
. **1112**, 1126, 1135
- \M@delimiters@num
. **2797**, 2805–2810,
2818, 2820, 2822, 2824, 2826,
2828, 2830, 2832, 2834, 2836, 2838
- \M@delimiters@set
. 984, 1392, **2795**, **2840**
- \M@delimitersshape **739**, 2796, 2841
- \M@diacritics@set 984, 1382, **2505**
- \M@diacriticsshape . . . **729**, 2506
- \M@digits@set 984, 1390, **2717**
- \M@digitsshape **737**, 2718
- \M@DoubleArgError **554**, 1065
- \M@entries@assert . 1564, 1580, 1599
- \M@extbigops@set . 984, 1395, **2888**
- \M@extbigopsshape **742**, 2889
- \M@extsymbols@set 984, 1397, **3014**
- \M@extsymbolsshape . . . **744**, 3015
- \M@fill@nfss@shapes
. **766**, 837, 845, 874
- \M@FontChangeInfo **438**, 975
- \M@ForbiddenCharmFile
. **591**, 1442, 1450
- \M@ForbiddenCharmLine
. **577**, 1444, 1452
- \M@frak@mathcodes 3495, **3500**
- \M@frak@num **3498**, 3501–3552
- \M@frak@set 984, 1401, **3488**
- \M@frakshape **748**, 3499
- \M@greeklower@set 984, 1384, **2563**
- \M@greeklowershape . . . **731**, 2564
- \M@greekupper@set 984, 1383, **2518**
- \M@greekuppershape . . . **730**, 2519
- \M@hebrew@set 984, 1389, **2688**
- \M@hebrewshape **736**, 2689
- \M@HModeError
566, 3351, 3426, 3491, 3556, 3621
- \M@index@assert 1546, 1552
- \M@IntegralItalicFactor . 685, 693
- \M@InvalidOptionError . . **465**, 877
- \M@InvalidSuboptionError **478**, 890
- \M@keys **751**, 878
- \M@keyword@info@begin . **1298**, 1355
- \M@loader
70, 342, 343, 349, 390, 392, 831
- \M@local@info@begin . . **1308**, 1361
- \M@localfonts . **697**, 1097, **1099**,
1102, 1110, **1110**, 1291, 1361
- \M@lower@set 984, 1381, **2441**, **2473**

- `\M@lowershape` 728, 2442, 2474
- `\M@MathConstantsNoAdjustWarning`
 532, 1052
- `\M@MissingControlSequenceError`
 543, 1061
- `\M@MissingNFSSShapesWarning` .
 449, 784
- `\M@MissingOptionError` . . 491, 904
- `\M@MissingSuboptionError` 500, 910
- `\M@newfont` . . . 792, 875, 933, 1089
- `\M@NewFontCommandInfo` . 439, 1090
- `\M@NoBaseModeDetectedWarning` .
 456, 811
- `\M@NoFontAdjustError` . . 606, 1179
- `\M@NoluaotfloadError` . . . 163, 191
- `\M@NoMathfontError`
 72, 86, 98, 102, 106, 110
- `\M@normal@mathcodes`
 1209, 1277, 1283
- `\M@num@localfonts`
 683, 1098, 1108, 1286, 1295, 1358
- `\M@number@assert` 1530, 1541
- `\M@operator@mathcodes`
 2733, 2789, 2793
- `\M@operator@num` . 2731, 2734–2787
- `\M@operator@set` . . 984, 1391, 2729
- `\M@operatorshape` . 738, 2732, 2792
- `\M@otf@features`
 . . 396, 402, 837, 845, 854, 859
- `\M@p@tch@decl@re` 667, 668
- `\M@parse@option` 898, 944
- `\M@radical@set`
 984, 1393, 2854, 2877
- `\M@radicalshape` . . 740, 2855, 2878
- `\M@retokenize` . . 675, 677, 677, 679
- `\M@RuleThicknessFactor` . . 684, 692
- `\M@SetMathConstants`
 1008, 1035, 1054
- `\M@split@colon` 788, 794
- `\M@strip@colon` 791, 796
- `\M@strip@equals` 897, 915
- `\M@SurdHorizontalFactor` . 687, 694
- `\M@SurdVerticalFactor` . . . 686, 695
- `\M@SymbolFontInfo` 433, 968
- `\M@symbols@set` . . . 984, 1396, 2919
- `\M@symbolsshape` 743, 2920
- `\m@th@const@nts@font`
 1034, 1040, 1043
- `\m@th@const@nts@font@sh@pe` . .
 1020, 1024, 1032, 1045
- `\m@thf@nt` 1000
- `\M@upper@set` 984, 1380, 2382, 2411
- `\M@uppershape` . . . 727, 2383, 2412
- `\M@XeTeXLuaTeXError` . . . 120, 142
- `\mapsfrom` 3258
- `\mapsto` 3227
- `\math@fonts` 1048, 1054
- `\mathand` 2948, 2991
- `\mathbackslash` 2816, 2817
- `\mathbb` 1069, 3349, 3351
- `\mathbcal` 1069, 3554, 3556
- `\mathbf` 1140
- `\mathbfit` 1141
- `\mathbffrac` 1069, 3619, 3621
- `\mathbfsc` 1144
- `\mathbfscit` 1145
- `\mathcal` 1069, 3424, 3426
- `\mathconstantsfont`
 105, 106, 514, 522,
 526, 528, 534, 1004, 1009, 1055
- `\mathdollar` 2946
- `\mathellipsis` 2922, 2988
- `\mathfrac` 1069, 3489, 3491
- `\mathhash` 2945, 2989
- `\mathit` 1139
- `\mathng` 3196, 3197
- `\mathnolimitsmode` 1405
- `\mathparagraph` 2949
- `\mathpercent` 2947, 2990
- `\mathring` 2516
- `\mathrm` 1138
- `\mathsc` 1142
- `\mathscit` 1143
- `\mathsection` 2950

<code>\mathsterling</code>	2951	<code>\npreceq</code>	3175
<code>\mem</code>	2702	<code>\nrightarrow</code>	3222
<code>\mid</code>	2980	<code>\nrightarrow</code>	3220
Missing \$ inserted	<i>p. 17</i>	<code>\nsim</code>	3162
Missing control sequence	<i>p. 17</i>	<code>\nsimeq</code>	3164
Missing option for <code>\mathfont</code>	<i>p. 15</i>	<code>\nsimeqq</code>	3165
Missing package <code>fontspec</code>	<i>p. 15</i>	<code>\nsqsubseteq</code>	3146
Missing suboption for <code>\mathfont</code>	<i>p. 15</i>	<code>\nsqsupseteq</code>	3147
missing X _Y TeX or LuaTeX	<i>p. 4</i>	<code>\nsubset</code>	3140
<code>\mkern</code>	2875	<code>\nsubseteq</code>	3142
<code>\models</code>	2998	<code>\nsucc</code>	3174
Multiple characters in argument	<i>p. 17</i>	<code>\nsucceq</code>	3176
N			
<code>\nabla</code>	2550, 2559, 3012, 3199	<code>\nsupset</code>	3141
<code>\napprox</code>	3163	<code>\nsupseteq</code>	3143
<code>\natural</code>	3037	<code>\ntriangleleft</code>	3158
<code>\Nearrow</code>	3323	<code>\ntrianglelefteq</code>	3160
<code>\nearrow</code>	3322	<code>\ntriangleright</code>	3159
<code>\neg</code>	2952	<code>\ntrianglerighteq</code>	3161
<code>\neq</code>	3022, 3150	<code>\Nu</code>	2274, 2533
<code>\nequiv</code>	3185	<code>\nun</code>	2703
<code>\neswarrow</code>	3331	<code>\Nwarrow</code>	3325
<code>\newmathbf</code>	91, 1119, 1129, 1140	<code>\nwarrow</code>	3324
<code>\newmathbf fit</code>	92, 1120, 1130, 1141	<code>\nwsearrow</code>	3330
<code>\newmathbfsc</code>	95, 1123, 1133, 1144	O	
<code>\newmathbfscit</code>	96, 1124, 1134, 1145	<code>\odiv</code>	3069
<code>\newmathfontcommand</code>		<code>\odot</code>	3071
109, 110, 1086, 1087, 1095, 1115		<code>\oiint</code>	2332, 2918
<code>\newmathit</code>	90, 1118, 1128, 1139	<code>\oiintop</code>	2908, 2918
<code>\newmathrm</code>	89, 1117, 1127, 1138	<code>\oint</code>	2331, 2917
<code>\newmathsc</code>	93, 1121, 1131, 1142	<code>\ointop</code>	2907, 2917
<code>\newmathscit</code>	94, 1122, 1132, 1143	<code>\oint</code>	2330, 2916
<code>\ngeq</code>	3153	<code>\ointop</code>	2906, 2916
<code>\ngsim</code>	3168	<code>\Omega</code>	2285, 2544
<code>\nleftarrow</code>	3253	<code>\omega</code>	2255, 2588
<code>\nleftarrow</code>	3251	<code>\Omicron</code>	2276, 2535
<code>\nLeftrightarrow</code>	3282	<code>\omicron</code>	2246, 2579
<code>\nleq</code>	3152	<code>\ominus</code>	3068
<code>\nlsim</code>	3167	<code>\operator@font</code>	2791
no previous font	<i>p. 15</i>	<code>\oplus</code>	3066
<code>\nprec</code>	3173	<code>\oslash</code>	3070
		<code>\otimes</code>	3067

P

Package mathfont Info *p. 13*
 \backslash parallel 2981
 parse \backslash mathfont arguments . . . *p. 31*
 \backslash partial 2955
 \backslash Phi 2282, 2541
 \backslash phi 2252, 2585
 \backslash Pi 2277, 2536
 \backslash pi 2247, 2580
 \backslash pm 2966
 \backslash prec 3125
 \backslash precapprox 3133
 \backslash preceq 3127
 \backslash preceqq 3129
 \backslash precnapprox 3183
 \backslash precneq 3177
 \backslash precneqq 3179
 \backslash precnsim 3181
 \backslash precprec 3135
 \backslash precsim 3131
 \backslash prime 2943
 \backslash prod 2314, 2884, 2886
 \backslash proportion 3096
 \backslash propto 3090
 \backslash Psi 2284, 2543
 \backslash psi 2254, 2587

Q

\backslash qeq 3122
 \backslash qof 2708

R

\backslash r@@t **2859**
 \backslash radicandoffset . . . 688, 696, 2875
 \backslash ratio 3095
 \backslash rbrace 2821
 \backslash rcirclearrow 3333
 \backslash Relbar 2928,
 2932, 2936, 3339, 3343, 3346
 \backslash relbar 2926,
 2931, 2935, 3338, 3342, 3345
 \backslash resh 2709
 \backslash rguil 1813, 2296, 2825, 2847

\backslash Rho 2278, 2537
 \backslash rho 2248, 2581
 \backslash Rightarrow 3221
 \backslash rightarrow 3218, 3219
 \backslash rightarrowtail 3236
 \backslash rightarrowtobar 3245
 \backslash Rightbararrow 3228
 \backslash rightbararrow 3226, 3227
 \backslash rightbrace 2804, 2851
 \backslash rightdasharrow 3233
 \backslash rightharpoonup 3235
 \backslash rightharpoonup 3234
 \backslash rightleftarrows 3289
 \backslash rightleftharpoons . . . 3217, 3290
 \backslash rightplusarrow 3237
 \backslash rightrightarrow 3247
 \backslash rightrightarrowrightarrow . . . 3248
 \backslash rightsquigarrow 3239
 \backslash rightwarrow 3238
 \backslash rightwhitearrow 3246
 \backslash ringeq 3115
 \backslash rootbox 2867, 2870, 2872
 \backslash rrguil 1815, 2298, 2829, 2849
 \backslash Rightarrow 3223
 \backslash RuleThicknessFactor
 79, 1160, 1175, 1182

S

\backslash samekh 2704
 \backslash Sampi 2599
 \backslash sampi 2611
 \backslash San 2604
 \backslash san 2616
 \backslash Searrow 3327
 \backslash searrow 3326
 \backslash seq 3111
 \backslash setfont 78, **1002**, 1007, 1373, 1378
 \backslash setmathfontcommands
 1005, **1137**, 1146
 \backslash setminus 2971
 \backslash sharp 3038
 \backslash shin 2710

<code>\Sho</code>	2603	<code>\succnsim</code>	3182
<code>\Sigma</code>	2279, 2538	<code>\succsim</code>	3132
<code>\sigma</code>	2249, 2582	<code>\succsucc</code>	3136
<code>\sim</code>	2977, 2994, 2996	<code>\sum</code>	2315, 2883, 2885
<code>\simeq</code>	2993, 3017, 3105	<code>\supset</code>	3079
<code>\simeqq</code>	3107, 3108	<code>\supseteq</code>	3081
<code>\simneqq</code>	3166	<code>\supsetneq</code>	3145
<code>\spadesuit</code>	3049	<code>\surd</code>	2326, 2856, 2879
<code>\sqcap</code>	3057	<code>\surdbox</code>	681, 2861, 2863, 2864, 2866–2868, 2873
<code>\sqcup</code>	3058	<code>\SurdHorizontalFactor</code>	82, 1161, 1176, 1184
<code>\sqdot</code>	3075	<code>\SurdVerticalFactor</code>	81, 1161, 1176, 1185
<code>\sqminus</code>	3074	<code>\Swarrow</code>	3329
<code>\sqplus</code>	3072	<code>\swarrow</code>	3328
<code>\sqrtsign</code>	2860, 2875	T	
<code>\sqsubset</code>	3018, 3082		
<code>\sqsubseteq</code>	3084	<code>\Tau</code>	2280, 2539
<code>\sqsubsetneq</code>	3148	<code>\tau</code>	2250, 2583
<code>\sqsupset</code>	3019, 3083	<code>\tav</code>	2711
<code>\sqsupseteq</code>	3085	terminal	<i>p. 35, p. 43</i>
<code>\sqsupsetneq</code>	3149	<code>\tet</code>	2698
<code>\sqtimes</code>	3073	<code>\textbackslash</code>	2816
<code>\ssharp</code>	3040	<code>\textng</code>	3195, 3197
<code>\sssim</code>	3110	The nfss family	<i>p. 13</i>
<code>\st@ck@fl@trel</code>	1190, 1191	<code>\therefore</code>	3093
<code>\stack@flatrel</code> ..	1189 , 2994, 2996	<code>\Theta</code>	2269, 2528
<code>\star</code>	3062	<code>\theta</code>	2239, 2572
<code>\stareq</code>	3119	<code>\tilde</code>	2517
<code>\Stigma</code>	2602	<code>\times</code>	2312, 2962
<code>\stigma</code>	2614	<code>\tracinglostchars</code>	1205, 1206
suboption italic	<i>p. 30, p. 34</i>	<code>\triangleeq</code>	3120
suboption roman	<i>p. 30, p. 34</i>	<code>\triangleleft</code>	3086
suboption upright	<i>p. 30</i>	<code>\trianglelefteq</code>	3088
<code>\subset</code>	3078	<code>\triangleright</code>	3087
<code>\subseteq</code>	3080	<code>\trianglerighteq</code>	3089
<code>\subsetneq</code>	3144	<code>\tsadi</code>	2707
<code>\succ</code>	3126	<code>\twoheaddownarrow</code>	3312
<code>\succapprox</code>	3134	<code>\twoheadleftarrow</code>	3275
<code>\succeq</code>	3128	<code>\twoheadrightarrow</code>	3244
<code>\succeqq</code>	3130	<code>\twoheaduparrow</code>	3298
<code>\succnapprox</code>	3184		
<code>\succneq</code>	3178		
<code>\succneqq</code>	3180		

U

unable to load *p. 4, 5*
 \backslash unexpanded 676
 \backslash Uparrow 3204, 3292
 \backslash uparrow 3203, 3291
 \backslash uparrowto bar 3299
 \backslash upbararrow 3294
 \backslash updasharrow 3295
 \backslash Updownarrow 3208, 3317
 \backslash updownarrow 3207, 3316
 \backslash updownarrows 3318
 \backslash updownharpoons 3320
 \backslash upharpoonleft 3296
 \backslash upharpoonright 3297
 \backslash Upsilon 2281, 2540
 \backslash upsilon 2251, 2584
 \backslash upuparrows 3302
 \backslash upwhitearrow 3300
 \backslash upwhitebararrow 3301
 \backslash Uradical 2857
 \backslash Uparrow 3293

V

\backslash varbeta 2256, 2589
 \backslash varcdot 3064
 \backslash varDigamma 2606
 \backslash vardigamma 2618
 \backslash varepsilon 2257, 2590
 \backslash varIm 3188
 \backslash varkaf 2712
 \backslash varkappa 2591
 \backslash varKoppa 2607
 \backslash varkoppa 2619
 \backslash varmem 2713
 \backslash varnun 2714
 \backslash varpe 2715
 \backslash varphi 2261, 2595

\backslash varRe 3187
 \backslash varrho 2259, 2593
 \backslash varSampi 2605
 \backslash varsampi 2617
 \backslash varsetminus 3065
 \backslash varsigma 2260, 2594
 \backslash varTheta 2286, 2545
 \backslash vartheta 2258, 2592
 \backslash vartsadi 2716
 \backslash vav 2695
 \backslash vdash 3034
 \backslash vee 3054
 \backslash veeeq 3118
 \backslash vert 2814

W

\backslash wclubsuit 3045
 \backslash wdiamondsuit 3046, 3051
 \backslash wedge 3053
 \backslash wedgeeq 3117
 \backslash wheartsuit 3047, 3050
 \backslash wp 3023
 \backslash wspadesuit 3048

X

\backslash Xi 2275, 2534

Y

\backslash yod 2699
Your command is invalid without
 Lua-based *p. 18*
Your \backslash mathconstants on line . . . *p. 16*

Z

\backslash zayin 2696
 \backslash Zeta 2267, 2526
 \backslash zeta 2237, 2570
 \backslash zigzagarrow 3308, 3309