

Package **mathfont** v. 3.0a Implementation

Conrad Kosowsky

February 2026

`kosowsky.latex@gmail.com`

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

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

As of version 2.0, using \LuaTeX is recommended. Minor backwards incompatible changes in version 3.0.

Overview

The **mathfont** package adapts Unicode text fonts for math mode. The package allows the user to specify a default font for different classes of math symbols, and it enables Unicode input in math mode. The package 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 provides package default settings, and section 3 deals with errors and messaging. Section 4 contains the 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 Matthew Braham, Sergio Callegari, Daniel Flipo, Nikos Platis, 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.

At high level, the package works as follows: the font-loader `\M@newfont` is a wrapper around NFSS macros to declare fonts—namely `\DeclareFontFamily` and `\DeclareFontShape`—or, if the user requested to use `fontspec` as a backend, the macro `\fontspec_set_family:Nnn`. All font-setting macros in the package will call `\M@newfont`. The primary font-setting command `\mathfont` is a wrapper around `\DeclareSymbolFont` and calls various `\M@<keyword>@set` commands. Each `\M@<keyword>@set` macro is a wrapper around a number of `\Umathcode` declarations that do the actual work of setting default font(s). The local font-change commands are wrappers around `\DeclareMathAlphabet`, and the Lua font adjustments alter the font table through the `luaotfload.patch_font` callback when \TeX loads the font. Specifically, we change the top-level flag `nomath` to false, alter character-level entries in the table to make the font more suitable for math typesetting, and add a `MathConstants` table based on font dimensions.

1 Setup

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

```
62 \NeedsTeXFormat{LaTeX2e}
63 \ProvidesPackage{mathfont}[2026/02/07 v. 3.0a]
```

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, namely the macros that `\@gobble` their argument and the macros that `\@gobbletwo` their 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.}}

```

The macro `\M@robust@def` is an engine-dependent command to define robust control sequences. We want this part of the package to work regardless of the engine, so we need an approach that doesn't depend on ϵ -TeX support. However, I don't want to use `\DeclareRobustCommand` with X_YTeX or LuaTeX because that will leave useless macros like `\mathfont␣` defined when we fill in the proper definitions of the user-level commands later.

```

78 \ifx\protected\@undefined
79   \let\M@robust@def\DeclareRobustCommand
80 \else
81   \def\M@robust@def{\protected\def}
82 \fi

```

First the commands that normally accept a single argument—the “bad” versions `\@gobble` the argument. To keep the syntax straightforward, we expand the definition using `\edef`. (We can't use `\expanded` because that's X_YTeX/LuaTeX only.) 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`.

```

83 \@tfor\@i:=\setfont
84   \RuleThicknessFactor
85   \IntegralItalicFactor
86   \SurdVerticalFactor
87   \SurdHorizontalFactor
88   \charmline
89   \charmfile
90   \CharmLine
91   \CharmFile
92   \CharmInfo
93   \CharmType\do{%
94     \edef\@tempa{\noexpand\M@robust@def\expandafter\noexpand\@i{%
95       \noexpand\M@NoMathfontError\expandafter\noexpand\@i
96       \noexpand\@gobble}}
97     \@tempa}

```

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

```

98 \@tfor\@i:=\newmathrm
99   \newmathit
100   \newmathbf
101   \newmathbfi
102   \newmathsc
103   \newmathscit
104   \newmathbfsc
105   \newmathbfscit\do{%
106     \edef\@tempa{\noexpand\M@robust@def\expandafter\noexpand\@i{%
107       \noexpand\M@NoMathfontError\expandafter\noexpand\@i
108       \noexpand\@gobbletwo}}
109   \@tempa}

```

The two commands with weird “arguments”: `\charinfo` and `\charmtype` scan and remove the next integer, so we assign a count value instead of gobbling stuff.

```

110 \M@robust@def\charinfo{\M@NoMathfontError\charinfo
111   \begingroup
112     \afterassignment\endgroup
113     \count@}
114 \M@robust@def\charmtype{\M@NoMathfontError\charmtype
115   \begingroup
116     \afterassignment\endgroup
117     \count@}

```

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.

```

118 \@tfor\@i:=\documentfont
119   \mathfont
120   \mainfont
121   \mathfontshapes
122   \mathconstantsfont\do{%
123     \edef\@tempa{\noexpand\M@robust@def\expandafter\noexpand\@i{%
124       \noexpand\M@NoMathfontError\expandafter\noexpand\@i
125       \noexpand\@ifnextchar[%
126         {\noexpand\expandafter\noexpand\@gobble
127           \noexpand\@gobble@brackets}
128         {\noexpand\@gobble}}}
129   \@tempa}

```

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

```

130 \M@robust@def\newmathfontcommand{%

```

```
131 \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.)

```
132 \ifx\directlua\@undefined
133 \else
134 \M@XeTeXLuaTeXtrue
135 \fi
136 \ifx\XeTeXrevision\@undefined
137 \else
138 \M@XeTeXLuaTeXtrue
139 \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 \LaTeX adds to the error message. Notice that we expand the error before the \endgroup —this is because we need to switch $\text{\M@XeTeXLuaTeXError}$ 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.

```
140 \ifM@XeTeXLuaTeX\else
141 \begingroup
142 \catcode`\ =12\relax
\M@XeTeXLuaTeXError 143 \def\M@XeTeXLuaTeXError{\GenericError{}}%
144 {\MessageBreak\MessageBreak
145 Package mathfont error:%
146 \MessageBreak\MessageBreak
147 *****\MessageBreak
148 *                               *\MessageBreak
149 *      UNABLE TO                *\MessageBreak
150 *      LOAD MATHFONT            *\MessageBreak
151 *                               *\MessageBreak
152 *      Missing XeTeX            *\MessageBreak
153 *      or LuaTeX                *\MessageBreak
154 *                               *\MessageBreak
155 *****\MessageBreak\@gobbletwo}%
156 {See the mathfont package documentation for explanation.}%
```

```

157 {I need XeTeX or LuaTeX to use mathfont. It\MessageBreak
158 looks like the current engine is something\MessageBreak
159 else, so I'm going to stop reading in the\MessageBreak
160 package file now. (You won't be able to use\MessageBreak
161 commands from mathfont in your document.) To\MessageBreak
162 load mathfont correctly, please retypeset your\MessageBreak
163 document with one of those two engines.^^J}}%
164 \expandafter\endgroup
165 \M@XeTeXLuaTeXError
166 \AtEndOfPackage{%
167   \typeout{:: mathfont :: Failed to load\on@line.}}
168 \expandafter\endinput % we \endinput with a balanced conditional
169 \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.

```

170 \ifdefined\directlua
171   \M@adjust@fonttrue % if engine is LuaTeX, adjust font by default
172   \IfFileExists{luaotfload.sty}
173     {\M@Noluaotfloadfalse\RequirePackage{luaotfload}}
174     {\M@Noluaotfloadtrue}
175 \else
176   \AtEndOfPackage{\PackageWarningNoLine{mathfont}{%
177     The current engine is XeTeX, but as\MessageBreak
178     of mathfont version 2.0, LuaTeX is\MessageBreak
179     recommended. Consider compiling with\MessageBreak
180     LuaLaTeX. Certain features will not\MessageBreak
181     work with XeTeX}}
182 \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.

```

183 \ifM@Noluaotfload % true if LuaTeX AND no luaotfload.sty
184 \begingroup
185 \catcode`\ =12\relax
\M@NoluaotfloadErr 186 \def\M@NoluaotfloadError{\GenericError{}}%
187 {\MessageBreak\MessageBreak
188 Package mathfont error:%

```

```

189 \MessageBreak\MessageBreak
190 *****\MessageBreak
191 *                               *\MessageBreak
192 *      UNABLE TO               *\MessageBreak
193 *      LOAD MATHFONT          *\MessageBreak
194 *                               *\MessageBreak
195 *      Cannot find the        *\MessageBreak
196 *      file luaotfload.sty    *\MessageBreak
197 *                               *\MessageBreak
198 *****\MessageBreak\@gobbletwo}%
199 {You are likely seeing this message because you haven't^^J%
200 installed luaotfload. Check your TeX distribution for a^^J%
201 list of the packages on your system.^^J^^J%
202 See the mathfont documentation for further explanation.}%
203 {You're in trouble here. It looks like the current\MessageBreak
204 engine is LuaTeX, so I need the luaotfload package\MessageBreak
205 to make mathfont work correctly. However, I can't\MessageBreak
206 find luaotfload, which likely means something is\MessageBreak
207 wrong with your TeX installation. I'm going to stop\MessageBreak
208 reading in the mathfont package file. (You won't be\MessageBreak
209 able to use commands from mathfont in your document.)\MessageBreak
210 To load mathfont properly, make sure you installed\MessageBreak
211 luaotfload.sty in a directory searchable by TeX or\MessageBreak
212 compile with XeLaTeX.^^J}}%
213 \expandafter\endgroup
214 \M@NoluaotfloadError
215 \AtEndOfPackage{%
216   \typeout{:: mathfont :: Failed to load\on@line.}}
217 \expandafter\endinput % we \endinput with a balanced conditional
218 \fi

```

Easter egg!!

```

219 \DeclareOption{easter-egg}{%
220   \ifE@sterEggDecl@red\else
221     \E@sterEggDecl@redtrue
222     \newcount\@easter@egg@
223     \protected\def\EasterEggUpdate{%
224       \ProcessE@sterEgg\showtokens\expandafter{\E@sterEggUpd@te}}
225     \let\ProcessE@sterEgg\relax

```

Two status updates during package loading.

```

226   \edef\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
227     Okay, opening your Easter egg.^^J%

```

```

228     Type \string\EasterEggUpdate\space in your^^J%
229     document to see the status.^^J^^J}
230 \EasterEggUpdate
231 \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
232     Uh oh. It looks like your Easter^^J%
233     egg flew out the window. I don't^^J%
234     I don't suppose you know the best^^J%
235     kind of bait to lure an egg?^^J^^J}
236 \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.

```

237 \def\ProcessE@sterEgg{%
238     \edef\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
239     \ifodd\@easter@egg@
240         \ifcase\numexpr(\@easter@egg@ - 1) / 2\relax
241             An Easter bunny must be related to a^^J%
242             platypus, no? Some sort of monotreme...%
243         \or
244             Don't count your chickens before they hatch^^J%
245             out of Easter eggs! But we don't have any^^J%
246             chickens right now because there are no eggs,^^J%
247             and the supply chain is sad.%
248         \or
249             Sorry, I'm late to a meeting. Can't talk right now.%
250         \or
251             Sunday, Monday, Tuesday, Wednesday, also^^J%
252             known as hump day, as in camel humps, which^^J%
253             I must say look distinctly egg-like if you^^J%
254             squint.%
255         \or
256             I'm calling Eggs Anonymous!%
257         \or
258             Sorry, I'm on the phone. Can't talk right now.%
259         \or
260             Still haven't found your Easter egg. I know^^J%
261             it's floating around here somewhere. Like an^^J%
262             asteroid in space, hopefully without the^^J%
263             massive extinction event.%
264         \or

```



```

265         Did you know eggs are used to make certain^^J%
266         types of vaccines? PSA: get your flu shot^^J%
267         and your covid shot!%
268     \or
269         Three large eggs.^^J%
270         Three large eggs.^^J%
271         See how they crack.^^J%
272         See how they crack.^^J%
273         Their broken shells are so pearly white.^^J%
274         In simmering water they catch the light.^^J%
275         Did you ever see such a sight in your life^^J%
276         As three poached eggs?%
277     \or
278         Do gnus eat eggs? Surely they must.%
279     \or
280         Okay, I have a fishing rod, some twine, and^^J%
281         a hook, but I still haven't caught your Easter^^J%
282         egg. Apparently it's harder to catch an egg^^J%
283         than a fish.%
284     \or
285         Sorry, I'm out fishing. Can't talk right now.%
286     \or
287         Is ghoti really an acceptable phonetic^^J%
288         spelling of fish? I am skeptical.%
289     \or
290         Perhaps an Easter bunny is actually a species^^J%
291         of fish. A rabbit fish.%
292     \else
293         Sorry, I'm all out of witty things to say.^^J%
294         Check back later.%
295     \fi
296     \else
297         Still wrangling. Check back later.%
298     \fi^^J^^J}%
299     \global\advance\@easter@egg@\@ne}

```

One status update \AtBeginDocument.

```

300     \AtBeginDocument{\bgroup
301         \let\ProcessE@sterEgg\relax
302         \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
303         If we have zero eggs^^J%
304         and zero bunnies, how^^J%
305         many gnats does it take^^J%

```

```

306         to change a lightbulb??^^J^^J}
307     \EasterEggUpdate
308 \egroup}

```

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

```

309 \def\math@E@sterEggUpd@te{\begingroup
310   \let\ProcessE@sterEgg\relax
311   \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
312     Scrambled, poached, or sunny side up?^^J^^J}%
313   \EasterEggUpdate
314 \endgroup
315 \global\let\math@E@sterEggUpd@te\relax}
316 \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.

```

317 \AtEndDocument{\let\ProcessE@sterEgg\relax
318   \begingroup
319     \edef\*{\@backslashchar*}
320     \edef\/{\@backslashchar/}
321     \edef\={\@backslashchar=}
322     \catcode`\+=\active
323     \everyeof{\noexpand}
324     \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.

```

325 \def\@tempb{^^J}
326 \edef\@tempa{\scantokens{Easter Egg Status:\@tempb\@tempb
327   The egg has been retrieved. What\@tempb
328   pinnacle of pulchritude!\@tempb\@tempb
329   ++++++\@tempb
330   ++++++\@tempb

```

```

331      ++++++*****\@tempb
332      ++++++-----\@tempb
333      ++++++*****\@tempb
334      ++++++\****/\****/\****/\****\@tempb
335      ++++++\****/\****/\****/\****\@tempb
336      ++++++*****\@tempb
337      ++++++*****\@tempb
338      +***\****/\****/\****/\****\@tempb
339      +***\****/\****/\****/\****\@tempb
340      ++++++*****\@tempb
341      ++++++*****\@tempb
342      ++++++\****/\****/\****/\****\@tempb
343      ++++++\****/\****/\****/\****\@tempb
344      ++++++*****\@tempb
345      ++++++-----\@tempb
346      ++++++*****\@tempb
347      ++++++*****\@tempb
348      ++++++\=====/\@tempb
349      ++++++\=====/\@tempb
350      ++++++\=====/\@tempb
351      ++++++|====|\@tempb
352      ++++++/===== \@backslashchar\@tempb
353      ++++++(___)\@tempb}}

```

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

```

354      \expandafter\endgroup\expandafter
355      \def\expandafter\E@sterEggUpd@te\expandafter{\@tempa}
356      \EasterEggUpdate
357      \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
358        Happy, happy day! Happy,^^J%
359        happy day! Clap your hands,^^J%
360        and be glad your hovercraft^^J%
361        isn't full of eels!^^J^^J}
362      \EasterEggUpdate
363      \let\E@sterEggUpd@te\relax
364      \let\EasterEggUpdate\relax}
365      \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.

```

366 \DeclareOption{default-loader}{\M@loader\z@}
367 \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.

```
368 \DeclareOption{adjust}{\M@adjust@fonttrue}
369 \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`.

```
370 \DeclareOption*{\M@font@loadedtrue
371   \edef\M@font@load{\CurrentOption}}
372 \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.

```
373 \ifcase\M@loader
374   \@mathfontinfo{Default font-loader was
375     requested for font loading.}
376 \or
377   \@mathfontinfo{Package fontspec was
378     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.

```
379 \@ifpackageloaded{fontspec}
380   {\@mathfontinfo{Package fontspec detected.}
381     \csname bool_if:NTF\expandafter\endcsname
382       \csname g__fontspec_math_bool\endcsname
383       {\@mathfontinfo{Setting
384         \string\g__fontspec_math_bool to false.}
385         \csname bool_set_false:N\expandafter\endcsname
386         \csname g__fontspec_math_bool\endcsname}{\relax}}}
```

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

```
387   {\@mathfontinfo{Package fontspec not detected.}
388     \IfFileExists{fontspec.sty}
389       {\@mathfontinfo{File fontspec.sty was found.}
390         \@mathfontinfo{Loading fontspec.}
391         \RequirePackage[no-math]{fontspec}}
392     {\PackageError{mathfont}
393       {Missing package fontspec; ^^J%
394       using default font-loader instead}
395       {You requested fontspec as the font-loader\MessageBreak
396       for mathfont. However, I can't find the\MessageBreak
```

```

397         package file for fontspec, so I'm going to\MessageBreak
398         use mathfont's built-in font-loader. (This\MessageBreak
399         likely means that something is wrong with\MessageBreak
400         your TeX installation.) Check your TeX\MessageBreak
401         distribution for a list of the packages\MessageBreak
402         installed on your system. To resolve this\MessageBreak
403         error, make sure fontspec is installed in\MessageBreak
404         a directory searchable by TeX or load\MessageBreak
405         mathfont with the default-loader option.^~J}
406         \M@loader\z@}}
407 \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.

```

408 \ifcase\M@loader
409   \@mathfontinfo{Using default font-loader.}
410   \AtEndOfPackage{%
411     \typeout{:: mathfont :: Using default font-loader.}}
\M@otf@features 412   \def\M@otf@features{script=latn;language=dflt;+tlig;+liga;+lnum}
413 \or
414   \@mathfontinfo{Using fontspec as font-loader.}
415   \AtEndOfPackage{%
416     \typeout{:: mathfont :: Using fontspec as font-loader.}}
\M@otf@features 417   \def\M@otf@features{Script=Latin,%
418     Language=Default,%
419     Ligatures={TeX,Common},%
420     Numbers=Lining}
421 \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`.

```

422 \ifdefined\directlua
423   \ifM@adjust@font
424     \@mathfontinfo{Enabling Lua-based font adjustments.}
425     \AtEndOfPackage{%
426       \typeout{:: mathfont :: Lua-based font adjustments
427         enabled.}}
428   \else
429     \@mathfontinfo{Disabling Lua-based font adjustments.}

```

```

430 \AtEndOfPackage{%
431 \typeout{:: mathfont :: Lua-based font adjustments
432 disabled.}}
433 \fi
434 \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.

```

435 \ifM@adjust@font
436 \PackageError{mathfont}{Option^^J"adjust" ignored with XeTeX}
437 {Your package option "adjust" was ignored.\MessageBreak
438 This option works only with LuaTeX, and it\MessageBreak
439 looks like the current engine is XeTeX. To\MessageBreak
440 enable Lua-based font adjustments, typeset\MessageBreak
441 with LuaLaTeX.^^J}
442 \M@adjust@fontfalse
443 \fi
444 \@mathfontinfo{Disabling Lua-based font adjustments.}
445 \AtEndOfPackage{%
446 \typeout{:: mathfont :: Lua-based font adjustments disabled.}}
447 \fi

```

2 Default Settings

We save four macros from the L^AT_EX kernel for safe-keeping, and then we change their definitions. As of version 3.0 of `mathfont`, the new definitions for `\set@mathchar`, etc. are not necessary for implementing `mathfont`, but we keep them in the package to make `\DeclareMathSymbol` and friends compatible with Unicode. For these three control sequences, we convert the hexadecimal digits in `\count0` and `\count2` back to decimal and change the `\math` primitive to `\Umath`.

```

448 \let\@@set@mathchar\set@mathchar
449 \let\@@set@mathsymbol\set@mathsymbol
450 \let\@@set@mathaccent\set@mathaccent
451 \let\@@DeclareSymbolFont\DeclareSymbolFont
452 \let\@@DeclareSymbolFont@m@dropped\DeclareSymbolFont@m@dropped
453 \@onlypreamble\@@set@mathchar
454 \@onlypreamble\@@set@mathsymbol
455 \@onlypreamble\@@set@mathaccent
456 \@onlypreamble\@@DeclareSymbolFont
457 \@onlypreamble\@@DeclareSymbolFont@m@dropped

```

```

458 \@mathfontinfo{Adapting \noexpand\set@mathchar for Unicode.}
459 \@mathfontinfo{Adapting \noexpand\set@mathsymbol for Unicode.}
460 \@mathfontinfo{Adapting \noexpand\set@mathaccent for Unicode.}
461 \@mathfontinfo{Increasing upper bound on
462 \noexpand\DeclareSymbolFont to 256.}

```

Kernel command to set math characters from keystrokes.

```

463 \def\set@mathchar#1#2#3#4{%
464 \multiply\count\z@ by 16\relax
465 \advance\count\z@\count\tw@
466 \global\Umathcode`#2=\mathchar@type#3+#1+\count\z@\relax}

```

Kernel command to set math characters from control sequences.

```

467 \def\set@mathsymbol#1#2#3#4{%
468 \multiply\count\z@ by 16\relax
469 \advance\count\z@\count\tw@
470 \global\Umathchardef#2=\mathchar@type#3+#1+\count\z@\relax}

```

Kernel command to set accents.

```

471 \def\set@mathaccent#1#2#3#4{%
472 \multiply\count\z@ by 16\relax
473 \advance\count\z@\count\tw@
474 \protected\xdef#2{%
475 \Umathaccent\mathchar@type#3+\number#1+\the\count\z@\relax}}

```

We increase the upper bound on the number of symbol fonts to be 256. Lua \TeX and X \TeX allow up to 256 math families, but the \LaTeX kernel keeps the old upper bound of 16 symbol fonts under these two engines. We patch `\DeclareSymbolFont` to change the `\count18<15` to `\count18<\e@mathgroup@top`, where `\e@mathgroup@top` is the number of math families, which is 256 in X \TeX and Lua \TeX . 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 \LaTeX 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`.

```

476 \ifx\DeclareSymbolFont@m@dropped\@undefined
477 \edef\@tempa{\expandafter

```

```

478   \strip@prefix\meaning\DeclareSymbolFont}
479   \def\@tempb{\def\DeclareSymbolFont##1##2##3##4##5}
480   \else
481     \edef\@tempa{\expandafter
482       \strip@prefix\meaning\DeclareSymbolFont@m@dropped}
483     \def\@tempb{\def\DeclareSymbolFont@m@dropped##1##2##3##4##5}
484   \fi
\M@p@tch@decl@re 485 \def\M@p@tch@decl@re#1<15#2\@nil{#1<\e@mathgroup@top#2}
\M@DecSymDef 486 \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. If this package was `LuaTeX` only, we could use `\scantextokens`, which is nicely behaved and does what we expect. However, to make it compatible with `XYTeX`, we use `\scantokens`. 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. The trick (realized after much trial and error) is that if we scan the entire definition statement including `\def` and the macro definition, the end-of-file token doesn't end up in the macro definition, and we avoid the “file ended” error message.

```

487 \scantokens\expandafter{%
488   \expandafter\@tempb\expandafter{\M@DecSymDef}}

```

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.

```

489 \newbox\surdbox
490 \newcount\M@count
491 \newcount\M@num@localfonts
492 \newcount\rulethicknessfactor
493 \newcount\hsurdfactor
494 \newcount\vsurdfactor
495 \newmuskip\radicandoffset
496 \newread\M@Charm
497 \M@count\z@
498 \rulethicknessfactor\@m
499 \hsurdfactor\@m
500 \vsurdfactor\@m
501 \radicandoffset=1mu\relax

```


Necessary booleans and default math font shapes.

```

502 \newif\ifM@upper
503 \newif\ifM@lower
504 \newif\ifM@diacritics
505 \newif\ifM@greekupper
506 \newif\ifM@greeklower
507 \newif\ifM@agreekupper
508 \newif\ifM@agreeklower
509 \newif\ifM@cyrillicupper
510 \newif\ifM@cyrilliclower
511 \newif\ifM@hebrew
512 \newif\ifM@digits
513 \newif\ifM@operator
514 \newif\ifM@symbols
515 \newif\ifM@extsymbols
516 \newif\ifM@delimiters
517 \newif\ifM@radical
518 \newif\ifM@arrows
519 \newif\ifM@bigops
520 \newif\ifM@extbigops
521 \newif\ifM@bb
522 \newif\ifM@cal
523 \newif\ifM@frak
524 \newif\ifM@bcal
525 \newif\ifM@bfrak
526 \newif\if@optionpresent
527 \newif\if@suboptionpresent
528 \newif\ifM@arg@good
529 \newif\ifM@mode@

```

Default shapes.

```

530 \def\upperdefault{italic}           % latin upper
531 \def\lowerdefault{italic}           % latin lower
532 \def\diacriticsdefault{upright}    % diacritics
533 \def\greekupperdefault{upright}    % greek upper
534 \def\greeklowerdefault{italic}     % greek lower
535 \def\agreekupperdefault{upright}   % ancient greek upper
536 \def\agreeklowerdefault{italic}    % ancient greek lower
537 \def\cyrillicupperdefault{upright} % cyrillic upper
538 \def\cyrilliclowerdefault{italic}  % cyrillic lower
539 \def\hebrewdefault{upright}        % hebrew
540 \def\digitsdefault{upright}        % numerals
541 \def\operatordefault{upright*}      % operator font

```

```

542 \def\delimitersdefault{upright}    % delimiters
543 \def\radicaldefault{upright}        % surd
544 \def\bigopsdefault{upright}         % big operators
545 \def\extbigopsdefault{upright}      % extended big operators
546 \def\symbolsdefault{upright}       % basic symbols
547 \def\extsymbolsdefault{upright}    % extended symbols
548 \def\arrowsdefault{upright}        % arrows
549 \def\bbdefault{upright}            % blackboard bold
550 \def\caldefault{upright}           % caligraphic
551 \def\frakdefault{upright}          % fraktur
552 \def\bcalddefault{upright}         % bold caligraphic
553 \def\bfrakdefault{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 554 \def\M@keys{upper,lower,diacritics,greekupper,%
555   greeklower,agreekupper,agreeklower,cyrillicupper,%
556   cyrilliclower,hebrew,digits,operator,delimiters,%
557   radical,bigops,extbigops,symbols,extsymbols,arrows,%
558   bb,cal,frak,bcal,bfrak}
\M@defaultkeys 559 \def\M@defaultkeys{upper,lower,diacritics,greekupper,%
560   greeklower,digits,operator,symbols}

```

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

```

561 \ifM@adjust@font
\M@defaultkeys 562 \edef\M@defaultkeys{\M@defaultkeys,delimiters,radical,bigops}
563 \fi

```

A few macros that we use for assembling lists of font information and printing messages `\AtBeginDocument`.

```

564 \let\M@localfonts\@empty
565 \let\M@symbolfonts\@empty
\M@families 566 \let\M@families\@empty

```

And now the macros to add to those three control sequences. First is a helper macro that accepts two arguments and stores information about declaration of local font changes. The macro successively adds comma-separated pairs of control sequence and font name information to `\M@localfonts`. It also keeps track of the number of distinct font names in `\M@localfonts` with the count variable `\M@num@localfonts`. The `#1` argument is a control sequence (with all characters having catcode 12 from `\string`), and the `#2` argument is a font name. We have two different approaches depending on whether `\M@localfonts` is empty, i.e. if it's the first time calling `\M@addto@localfonts`. If `\M@localfonts` is

\@empty, that means we haven't added any fonts to the list yet, so we increase \M@num@localfonts. Otherwise we loop through \M@localfonts, and we increase \M@num@localfonts only if none of the entries in \M@localfonts use the #2 font. After incrementing (or not) the count, we append #1 and #2 to \M@localfonts.

```

567 \def\M@addto@localfonts#1#2#3#4{%
568   \begingroup
569     \@tempwatrue      % increase by default
570     \def\@tempa##1##2##3##4{##2}%
571     \@for\@j:=\M@localfonts\do{%
572       \edef\@tempb{\expandafter\@tempa\@j}%
573       \ifx\@tempbase\@tempb
574         \@tempwafalse % if \@tempbase is in list, don't add
575       \fi}%
576     \expandafter
577   \endgroup
578   \if@tempswa
579     \advance\M@num@localfonts\@ne
580   \fi
581   \ifx\M@localfonts\@empty
582   \else
583     \edef\M@localfonts{\M@localfonts,}%
584   \fi
585   \edef\M@localfonts{\M@localfonts{#1}{#2}{#3}{#4}}}
```

Same thing for symbol fonts.

```

586 \def\M@addto@symbolfonts#1#2#3#4{%
587   \ifx\M@symbolfonts\@empty
588   \else
589     \edef\M@symbolfonts{\M@symbolfonts,}%
590   \fi
591   \edef\M@symbolfonts{\M@symbolfonts{#1}{#2}{#3}{#4}}}
```

And font families.

```

\M@addto@families 592 \def\M@addto@families#1{%
593   \ifx\M@families\@empty
594   \else
\M@families 595     \edef\M@families{\M@families,}%
596   \fi
\M@families 597   \xdef\M@families{\M@families#1}}
```

3 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@FontChangeInfo 598 \def\M@FontChangeInfo#1#2{\@mathfontinfo{Setting #1 chars to #2!}}
\M@FontFamilyInfo 599 \def\M@FontFamilyInfo#1{\@mathfontinfo{Adding #1 to the nfss.}}
\M@SymbolFontInfo 600 \def\M@SymbolFontInfo#1#2#3{%
601   \@mathfontinfo{New symbol font uses TU/#1/#2/#3.}}
\M@NewFontCommandI 602 \def\M@NewFontCommandInfo#1#2#3#4{%
603   \@mathfontinfo{New \string#1 uses TU/#2/#3/#4.}}
```

Warnings and errors related to font declaration.

```
\M@NFSSShapesWarni 604 \def\M@NFSSShapesWarning#1#2{%
605   \PackageWarningNoLine{mathfont}
606   {The nfss family "#1"\MessageBreak
607   from line \the\inputlineno\space is missing shapes.\MessageBreak
608   You may see some substitutions\MessageBreak
609   or errors. See the log file for\MessageBreak
610   details}
611   \@mathfontinfo{Shapes missing: #2.}}
\M@NoBaseModeError 612 \def\M@NoBaseModeError#1{%
613   \PackageError{mathfont}
614   {Missing base-mode^^J%
615   version of font family "#1"}
616   {With LuaTeX, when you tell mathfont to\MessageBreak
617   use a font family from the nfss, I try to\MessageBreak
618   find a version of that font in the nfss\MessageBreak
619   that uses base mode. I couldn't do that\MessageBreak
620   here, so you may see some problems with\MessageBreak
621   your math. To resolve this error, either\MessageBreak
622   use XeTeX, or make sure the nfss contains\MessageBreak
623   a version of your font in base mode, and\MessageBreak
624   define \string\<font family>-base to be the nfss\MessageBreak
625   name for your base-mode family.^^J}}
\M@FamilyTypeError 626 \def\M@FamilyTypeError#1{%
627   \PackageError{mathfont}
628   {Invalid family type/^^J%
629   optional argument "#1" for \string\mainfont}
630   {The optional argument of
631   \string\mainfont\space should\MessageBreak
632   be one of rm, sf, or tt. You used something\MessageBreak
633   else, so I'm changing it to rm.^^J}}
```

Table 1: Package 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@FontFamilyInfo</code>	Declaring new font shape in the NFSS
<code>\M@NewFontCommandInfo</code>	New local font-change command
<code>\M@SymbolFontInfo</code>	Declare new symbol font
<code>\M@FamilyTypeError</code>	Error if bad argument for <code>\mainfont</code>
<code>\M@NFSSShapesWarning</code>	Warning if font is missing shapes
<code>\M@NoBaseModeError</code>	Error if no base-mode version of a font
<code>\M@InvalidOptionError</code>	Bad keyword for font-change command
<code>\M@InvalidSupoptionError</code>	Bad suboption for font-change command
<code>\M@MissingOptionError</code>	Missing keyword for font-change command
<code>\M@MissingSuboptionError</code>	Missing suboption for font-change command
<code>\M@FontShapesError</code>	Tried to add font shapes after preamble
<code>\M@LuaTeXOnlyWarning</code>	User called LuaTeX-only macro in XeTeX
<code>\M@HModeError</code>	Font-change command used outside math
<code>\M@MissingCSError</code>	No macro for font-change command
<code>\M@BadIntegerError</code>	Non-integer value for font adjustment
<code>\M@NoCharmFileError</code>	Bad file name for <code>\charmfile</code>
<code>\M@NoFontAdjustError</code>	Macro used without Lua font adjustments

Error and warning messages for keywords and shape identifiers.

```

\M@InvalidOptionEr 634 \def\M@InvalidOptionError#1{%
635   \PackageError{mathfont}
636   {Invalid^^Jkeyword "#1" on line \the\inputlineno}
637   {You used a character keyword that I'm\MessageBreak
638   not familiar with. Check that you spelled\MessageBreak
639   everything correctly. To resolve this\MessageBreak
640   error, make sure you use keywords that\MessageBreak
641   are listed in the documentation.^^J\@gobble}}
\M@InvalidSuboptio 642 \def\M@InvalidSuboptionError#1{%
643   \PackageError{mathfont}
644   {Invalid^^Jshape identifier "#1" on line \the\inputlineno}
645   {You used a suboption/shape identifier\MessageBreak
646   that I'm not familiar with. Check that\MessageBreak
647   you spelled everything correctly. To\MessageBreak

```

```

648 resolve this error, make sure you use\MessageBreak
649 shape identifiers that are listed in\MessageBreak
650 the documentation.^^J}}

```

```

\M@MissingOptionEr 651 \def\M@MissingOptionError{%
652   \PackageError{mathfont}
653   {Missing keyword on line \the\inputlineno}
654   {I didn't see a character keyword\MessageBreak
655   where I was expecting to. This can\MessageBreak
656   happen if you type ,, or ,= by\MessageBreak
657   mistake. To resolve this error,\MessageBreak
658   make sure you provided a comma-\MessageBreak
659   separated list of keywords.^^J}}

\M@MissingSuboptio 660 \def\M@MissingSuboptionError{%
661   \PackageError{mathfont}
662   {Missing suboption/^^Jshape identifier on line \the\inputlineno}
663   {I didn't see a suboption/shape identifier\MessageBreak
664   where I was expecting to. This can happen\MessageBreak
665   if you type ,, or =, or == by mistake. To\MessageBreak
666   resolve this error, make sure that every\MessageBreak
667   = sign comes before a suboption or that you\MessageBreak
668   provided a comma-separated list of shape\MessageBreak
669   identifiers, depending on the context.^^J}}

```

Error messages regarding arguments previously fed to `\mathfont` and friends.

```

\M@FontShapesError 670 \def\M@FontShapesError{%
671   \PackageError{mathfont}
672   {^^JCan't declare new font shapes after
673   \string\begin{document}}
674   {This error means that you (1) requested\MessageBreak
675   to use a family/series/shape combination\MessageBreak
676   for a font-change command in this package\MessageBreak
677   (2) after your document preamble (3) that\MessageBreak
678   does not match any font you used for this\MessageBreak
679   package in the preamble. To resolve this\MessageBreak
680   error, try declaring more font shapes in\MessageBreak
681   your preamble with \string\mathfontshapes.^^J}}

\M@LuaTeXOnlyWarni 682 \def\M@LuaTeXOnlyWarning#1{%
683   \PackageWarningNoLine{mathfont}
684   {Your \string#1\space on line
685   \the\inputlineno\MessageBreak
686   is mainly for use in LuaTeX with font\MessageBreak
687   adjustments enabled. In the current\MessageBreak
688   situation, it is probably not doing\MessageBreak

```

```
689 anything}}
```

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

```
\M@MissingCSError 690 \def\M@MissingCSError#1#2{%
691   \PackageError{mathfont}
692   {Missing control sequence^^J%
693   for\string#1\space on line \the\inputlineno}
694   {Your command was ignored. Instead of\MessageBreak
695   "#2,"\MessageBreak
696   I was expecting a single control\MessageBreak
697   sequence. To resolve this error,\MessageBreak
698   please use one control sequence instead.^^J}}

\M@HModeError 699 \def\M@HModeError#1{%
700   \PackageError{mathfont}
701   {Missing \string$ inserted^^J%
702   on line \the\inputlineno}
703   {I raised an error because you used\MessageBreak
704   \string#1\space outside of math mode,\MessageBreak
705   which isn't allowed. I inserted a \string$\MessageBreak
706   before your control sequence, so we\MessageBreak
707   should be all good now.^^J}}
```

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

```
\M@NoFontAdjustErr 708 \def\M@NoFontAdjustError#1{%
709   \PackageError{mathfont}
710   {\string#1^^J%
711   is invalid without Lua-based font adjustments}
712   {Your control sequence won't do anything\MessageBreak
713   without Lua-based font adjustments, but\MessageBreak
714   you didn't enable them. To resolve this\MessageBreak
715   error, load mathfont with LuaTeX and the\MessageBreak
716   package option "adjust" or remove your\MessageBreak
717   control sequence.^^J}}

\M@BadIntegerError 718 \def\M@BadIntegerError#1#2{%
719   \PackageError{mathfont}
720   {Bad argument^^J%
721   "#2" for \string#1 on line \the\inputlineno}
722   {Your command was ignored. Please make sure\MessageBreak
723   that your argument for \string#1\MessageBreak
724   is a nonnegative integer.^^J}}

\M@NoCharmFileErro 725 \def\M@NoCharmFileError#1{%
726   \PackageError{mathfont}{Missing Charm File}
727   {You requested to read the file\MessageBreak
```

```

728 "#1"\MessageBreak
729 with \string\charmfile. I can't find that file,\MessageBreak
730 so I'm ignoring your command. To resolve\MessageBreak
731 this error, make sure the filename is\MessageBreak
732 correct, and double check that the file\MessageBreak
733 is in a directory searchable by TeX.^^J}}

```

4 Font Declaration

We come to the fontloader. The main font declaration macro is `\M@newfont`, and it accepts one argument that is a combination of font name and optional OpenType feature information separated by a colon. The macro accomplishes four tasks: (1) it separates the font name and features and stores them in `\@tempbase` and `\@tempfeatures`; (2) it checks whether the argument is present as a font name in the NFSS; (3) if not, it declares the font in the NFSS, either with the built-in font loader or using `fontspec`; and (4) it stores the NFSS family name(s) in `\M@fontname` and `\M@fontnameb@se`. If `\M@newfont` reaches step 3 after `\begin{document}`, it raises a “can’t declare new font shapes” error.

Checking the NFSS happens in two steps. First, `\M@newfont` removes the spaces from its argument and checks whether the result matches a previous call to `\M@newfont`, and if yes, `\M@newfont` uses the family name from the previous call. If no, the macro checks whether the argument with spaces removed appears in the NFSS. (Properly declared NFSS font families shouldn’t have spaces in their names because \LaTeX ignores spaces when scanning a font family declaration.) If `\M@newfont` finds an NFSS family, it looks at the font shapes associated with that family using the helper macro `\M@check@nfss@shapes`. This command prints a warning if the NFSS is missing any standard series/shape combinations (`m/n`, `m/it`, `b/n`, and `b/it`) for the font family. The package does not add any font shapes to the NFSS at this point in the font-loading process.

If both checks of the NFSS fail, `mathfont` assumes the NFSS does not contain the user’s desired font and proceeds to declare it in the NFSS. For the built-in font loader, font declaration uses the helper macros `\M@fill@nfss@shapes` and `\M@declare@shape`. The first of these two macros loops through series/shape pairs and feeds them to the second macro, which acts as a wrapper around `\DeclareFontShape`. If the user requested to use `fontspec` for the font loader, `\M@newfont` feeds the font name and requested features to `\fontspec_set_family:Nnn`, which modifies the NFSS accordingly. Advanced users should keep in mind the difference between font declaration and font load-

ing in \LaTeX . Font declaration, the subject of this section, means adding font information to the NFSS through `\DeclareFontFamily` and `\DeclareFontShape`, and font loading is when \TeX reads a font file into memory via the `\font` primitive. \LaTeX handles font loading automatically at a `\selectfont` command or upon entering math mode, so a call to `\M@newfont` will not actually load any fonts into memory, just prepare \LaTeX to do so at a later time.

In \LuaTeX , users can load fonts in one of three modes, namely **node** (the default), **base**, or **harf**. Node mode works well for text, but it has more limited capabilities for math. Harf mode uses the HarfBuzz renderer and is appropriate for more complicated scripts. Accordingly, when the engine is \LuaTeX , **mathfont** loads fonts once in base mode for math and once in unspecified (so likely node) mode for text. If the user specifies a font family already in the NFSS, `\M@newfont` tries to find a base-mode version of the font family and raises an error if it cannot. When `\M@newfont` declares the font, it does so twice, once in unspecified mode and once in base mode. This is why we have two control sequences for font family names: `\M@fontn@me` is the font family, and `\M@fontn@me@base` contains the NFSS family name of the base-mode version of the font. In \XeTeX , these control sequences will be identical because **mathfont** does not load a font multiple times in that case.

During font declaration, **mathfont** links several pieces of information as follows:

- Given an $\langle argument \rangle$ with spaces removed, `\M@newfont` stores the corresponding NFSS family name in `\M@fontfamily@ $\langle argument \rangle$`
- The NFSS family name for the base-mode version of the font goes in `\M@fontfamily@base@ $\langle argument \rangle$`
- Given a $\langle family name \rangle$, `\M@newfont` stores the corresponding base-mode font family name in the control sequence `\ $\langle family name \rangle$ -base`. Users who want to declare their own fonts in the NFSS prior to using them with **mathfont** should manually define this control sequence so that **mathfont** knows where your base-mode font lives.
- Each NFSS font family is assigned a unique value of `\M@count` that is stored in `\M@fontid@ $\langle family \rangle$` .

These macro assignments are global. The difference relative to `\M@fontn@me` and `\M@fontn@me@base` is that `\M@fontn@me` and `\M@fontn@me@base` are temporary and always hold the font family from the most recent call to `\M@newfont`.

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 734 \def\M@check@nfss@shapes#1{%
735   \let\@tempb\@empty
736   \let\@tempwarning\@gobble
737   \@for\@i:=\mddefault/\shapedefault,%
738     \mddefault/\itdefault,%
739     \bfdefault/\shapedefault,%
740     \bfdefault/\itdefault\do{%
741     \expandafter\ifx\csname TU/#1/\@i\endcsname\relax
742       \def\@tempwarning{\M@NFSSShapesWarning{#1}}%
743       \edef\@tempb{\@tempb, \@i}%
744     \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`.

```

745   \@tempwarning{\expandafter\@gobble\@tempb}}

```

Next we have 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 746 \def\M@declare@shape#1#2#3#4#5#6{%
747   \ifcsname TU/#1/#5/#6\endcsname
748   \else
749     \DeclareFontShape{TU}{#1}{#5}{#6}{<->"\@tempbase#2:#3;#4"}{ }%
750   \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 751 \def\M@fill@nfss@shapes#1#2#3{%
752   \@for\@i:={#1}{#2;-smcp}{#3}{\mddefault}{\shapedefault},%
753   {#1}{/I}{#2;-smcp}{#3}{\mddefault}{\itdefault},%
754   {#1}{/B}{#2;-smcp}{#3}{\bfdefault}{\shapedefault},%
755   {#1}{/BI}{#2;-smcp}{#3}{\bfdefault}{\itdefault},%
```

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

```
756   {#1}{#2;+smcp}{#3}{\mddefault}{\scdefault},%
757   {#1}{/I}{#2;+smcp}{#3}{\mddefault}{\scdefault\itdefault},%
758   {#1}{/B}{#2;+smcp}{#3}{\bfdefault}{\scdefault},%
759   {#1}{/BI}{#2;+smcp}{#3}{\bfdefault}{\scdefault\itdefault}%
760   \do{\expandafter\M@declare@shape\@i}}
```

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 761 \def\M@split@colon#1:#2\@nil{%
762   \def\@tempbase{#1}%
763   \def\@tempfeatures{#2}}
\M@strip@colon 764 \def\M@strip@colon#1:{#1}
```

The main font-loading macro. It takes a single argument, which should look like either `⟨NFSS family⟩` or `⟨font name⟩:⟨optional features⟩`. The first thing `\M@newfont` does is split the font name and OpenType features and store each portion in `\@tempbase` and `\@tempfeatures`. If `\@tempfeatures` is not empty, it has an extra colon at the end, so we remove it.

```
\M@newfont 765 \def\M@newfont#1{%
766   \expandafter\M@split@colon\expanded{#1}:\@nil
767   \ifx\@tempfeatures\@empty\else
768     \edef\@tempfeatures{\expandafter\M@strip@colon\@tempfeatures}%
769     \fi
```

Then we find the font family name. We remove spaces from the argument, store it in `\@tempa`, and check whether the result matches a previous call to

`\M@newfont`. If yes, we retrieve the font family and base-mode family names and store them in the appropriate control sequences.

```

770 \edef\nospace\@tempa{#1}%
771 \ifcsname M@fontfamily@\@tempa\endcsname
772   \edef\M@f@ntn@me
773     {\csname M@fontfamily@\@tempa\endcsname}%
774   \edef\M@f@ntn@meb@se
775     {\csname M@fontfamily@base@\@tempa\endcsname}%
776 \else

```

Next check whether `\@tempa` appears as a font family in the NFSS.

```

777 \ifcsname TU+\@tempa\endcsname% is #1 font family in the nfss?
778 \let\M@f@ntn@me\@tempa

```

Check that the NFSS contains some font shapes. If any are missing, we issue a warning but do not fill them.

```

779 \M@check@nfss@shapes\M@f@ntn@me

```

With LuaTeX, we want a proper base-mode version of the font. In this situation, `mathfont` expects to find a second 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.

```

780 \ifdefined\directlua % if LuaTeX?
781   \ifcsname\M@f@ntn@me-base\endcsname % if base-mode version
782     \edef\M@f@ntn@meb@se
783       {\csname\M@f@ntn@me-base\endcsname}%

```

If the package found a base-mode font, again check that it contains some font shapes, and issue a warning if not.

```

784 \M@check@nfss@shapes\M@f@ntn@meb@se
785 \else

```

Raise an error if we can't find a base-mode font family, and use the regular font instead.

```

786 \M@NoBaseModeError\M@f@ntn@me
787 \expandafter\xdef
788   \csname\M@f@ntn@me-base\endcsname{\M@f@ntn@me}%
789 \let\M@f@ntn@meb@se\M@f@ntn@me
790 \fi

```

Base mode is a LuaTeX-only feature version of a font, so we link the base-mode identifier to the same font.

```

791 \else % if XeTeX?
792   \expandafter\xdef
793     \csname\M@f@ntn@me-base\endcsname{\M@f@ntn@me}%

```

```

794     \let\M@fontn@meb@se\M@fontn@me
795     \fi

```

Now save the font families for reference later.

```

796     \expandafter\xdef
797     \csname M@fontfamily@\@tempa\endcsname
798     {\M@fontn@me}%
799     \expandafter\xdef
800     \csname M@fontfamily@base@\@tempa\endcsname
801     {\M@fontn@meb@se}%
802     \else                                     % if #1 is not in nfss

```

If the argument does not match a known font family, we have to load it ourselves. First, we check that we are still in the document preamble, and if not, we issue an error.

```

803     \ifx\@onlypreamble\@notprerr % if after \begin{document}
804     \M@FontShapesError
805     \else                                     % if in preamble

```

If we are in the document preamble, we can still add information to the NFSS. We store the font family name in `\M@fontn@me` and print messages in the log file. Then we declare the font family and call `\M@fill@nfss@shapes` to declare all the shapes.

```

806     \ifcase\M@loader % are we using default font-loader?
807     \let\M@fontn@me\@tempa
808     \M@FontFamilyInfo\M@fontn@me
809     \DeclareFontFamily{TU}{\M@fontn@me}{}%
810     \M@fill@nfss@shapes{\M@fontn@me}{\M@otf@features}
811     {\@tempfeatures}%

```

If the engine is Lua_T_E_X, we load a separate version of the font with `mode=base` and `script=math`. We need to set the `script` because `luaotfload` processes math information only for fonts with the script set to `math`, and we definitely want that information loaded if present. Then we link the base-mode and regular versions.

```

812     \ifdefined\directlua
813     \edef\M@fontn@meb@se{\M@fontn@me-base}%
814     \M@FontFamilyInfo\M@fontn@meb@se
815     \DeclareFontFamily{TU}{\M@fontn@meb@se}{}%
816     \M@fill@nfss@shapes{\M@fontn@meb@se}{\M@otf@features}
817     {\@tempfeatures;-nomathparam;mode=base}%
818     \else
819     \let\M@fontn@meb@se\M@fontn@me
820     \fi
821     \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` and `Script=Math`.

```

822      \@mathfontinfo{Passing \@tempbase\space
823        to fontspec for handling!}%
824      \csname fontspec_set_family:Nnn\endcsname\M@f@ntn@me
825      {\M@otf@features,\@tempfeatures}\@tempbase}%
826      \ifdefined\directlua
827        \@mathfontinfo{Passing \@tempbase\space
828          with Renderer=Base to fontspec for handling!}%
829        \csname fontspec_set_family:Nnn\endcsname
830        \M@f@ntn@meb@se
831        {\M@otf@features,\@tempfeatures,%
832          RawFeature=-nomathparam,Renderer=Base}
833        {\@tempbase}%
834      \else
835        \edef\M@f@ntn@meb@se{\M@f@ntn@me}%
836      \fi
837    \fi

```

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

```

838      \expandafter\xdef\csname\M@f@ntn@me-base\endcsname
839      {\M@f@ntn@meb@se}%
840      \expandafter\xdef\csname M@fontfamily@\@tempa\endcsname
841      {\M@f@ntn@me}%
842      \expandafter\xdef
843      \csname M@fontfamily@base@\@tempa\endcsname
844      {\M@f@ntn@meb@se}%
845    \fi
846  \fi
847 \fi

```

Finally, assign `\M@count` values to the font family(ies) if needed and save their names in `\M@families`.

```

848  \ifcsname M@fontid@\M@f@ntn@me\endcsname % need new \M@count?
849  \else
850    \expandafter\xdef
851    \csname M@fontid@\M@f@ntn@me\endcsname{\the\M@count}%
852    \global\advance\M@count\@ne
853    \M@addto@families{\M@f@ntn@me}%
854  \fi

```

Same thing with the base-mode version of the font.

```

855 \ifcsname M@fontid@\M@fontn@meb@se\endcsname
856 \else
857   \expandafter\xdef
858     \csname M@fontid@\M@fontn@meb@se\endcsname{\the\M@count}%
859   \global\advance\M@count\@ne
860   \M@addto@families{\M@fontn@meb@se}%
861 \fi}

```

The font-loading commands should appear only in the preamble.

```

862 \@onlypreamble\M@declare@shape
863 \@onlypreamble\M@fill@nfss@shapes

```

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@opt` 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@opt` finds a valid keyword-option, it changes `\if@optionpresent` to true.

```

\M@check@opt 864 \def\M@check@opt#1{%
865   \@optionpresentfalse           % set switch to false by default
866   \ifx#1\@empty
867     \M@MissingOptionError
868   \else
869     \let\@temperror\M@InvalidOptionError % error by default
870     \@for\@j:=\M@keys\do{%
871       \ifx\@j#1%
872         \let\@temperror\@gobble % eliminate error
873         \@optionpresenttrue      % set switch to true
874       \fi}%
875     \@temperror{#1}%
876   \fi}

```

Now we have to parse the optional argument of `\mathfont`. The macro `\M@parse@option` carries out the following tasks:

1. Store the keyword in `\@temp@opt` and the suboption (if present) in `\@temp@sub`. Set the boolean corresponding to the presence of a suboption.
2. Check that `\@temp@opt` is actually a keyword and set the corresponding boolean.
3. Convert `\@temp@sub` into NFSS series and shape keywords.

We want to allow the user to specify options using an `xkeyval`-type syntax. However, we do not need the full package; a slim few 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 `=`.

```
\M@strip@equals 877 \def\M@strip@equals#1={#1}
\M@parse@option 878 \def\M@parse@option#1=#2\@nil{%
879   \def\@temp@opt{#1}%      % store option
880   \def\@temp@sub{#2}%      % store suboption
```

After storing the option and suboption, check for errors. If the user specified a suboption, `\@temp@sub` contains $\langle suboption \rangle$, and we use `\M@strip@equals` to get rid of the extra `=`. If the user does not specify a suboption, `\@temp@sub` will be empty. After `\M@parse@sub`, the NFSS series and shape codes for the suboption (if provided) will be stored in `\@tempseries` and `\@tempshape`.

```
881   \M@check@opt\@temp@opt
```

At this point, we have three possibilities for `\@temp@sub`:

1. `\@temp@sub` is `=`, which means the user wrote something like $\langle keyword \rangle =$, and indicates a missing suboption.
2. `\@temp@sub` is empty, which indicates the user didn't provide a suboption, and we should use the default setting.
3. Otherwise, we try to extract series and shape information from `\@temp@sub`.

We process `\@temp@sub` in that order. In case 3, we expect `\@temp@sub` to look like $\langle shape identifier \rangle =$, so we have to strip the final `=` before processing.

```
882   \begingroup
883     \def\@tempa{=}%
884     \expandafter
885   \endgroup
886   \@suboptionpresentfalse % set switch to false by default
887   \ifx\@temp@sub\@tempa   % if missing suboption
888     \M@MissingSuboptionError
889   \else
```



```

890 \ifx\@temp@sub\@empty % if no suboption provided
891 \else % if suboption provided, parse it
892 \suboptionpresenttrue % set switch to true
893 \edef\@temp@sub{\expandafter\M@strip@equals\@temp@sub}%
894 \M@parse@sub\@temp@sub
895 \fi
896 \fi}

```

Now a macro to convert a shape identifier into NFSS series and shape codes. Here the #1 argument is a control sequence such as \@temp@sub. The first thing we do is check whether #1 ends in an asterisk and set \M@base@ accordingly. This boolean is true if we use base mode (if no asterisk) and false otherwise (if asterisk). Note that \M@parse@sub should always be called when \@suboptionpresent is set to true. If we encounter a bad shape identifier in #1, we change \@suboptionpresent to false. (So it may be more accurate to name the conditional something like \@suboptioncheck, but I'm keeping the name as is to match \@optionpresent.)

```

\M@parse@sub 897 \def\M@parse@sub#1{%
898 \expanded{\noexpand\in@*{#1}}%

```

If #1 contains an asterisk, we check that it is the final character in #1 and strip it.

```

899 \ifin@
900 \begingroup
901 \expandafter\M@split@star#1\@nil
902 \ifx\@tempb\@empty
903 \expanded{\endgroup % first branch \endgroup
904 \def\noexpand#1{\@tempa}}%
905 \M@mode@true

```

If the asterisk is not the final character, that probably means something went wrong. (But we'll catch the problem later.)

```

906 \else
907 \endgroup % second branch \endgroup
908 \M@mode@false
909 \fi
910 \else
911 \M@mode@false
912 \fi

```

If the shape identifier contains a /, we interpret it as NFSS identifiers and do not check further, and otherwise, we check that the argument is one of `upright`, `italic`, `bold`, or `bolditalic`. We also accept `roman` for backwards compatibility. We store NFSS information in \@tempseries and \@tempshape.

```

913 \expanded{\noexpand\in@/{#1}}%
914 \ifin@
915 \expandafter\M@split@slash#1\@nil
916 \else

```

If the user wrote out a shape identifier, we have a bit more checking to do: we have to check whether #1 is one of `roman`, `upright`, `italic`, `bold`, or `bolditalic`. We let `\@tempa` be equal to different strings inside a group, and for each possibility, if #1 is that string, we set `\@tempseries` and `\@tempshape` to the correct definitions. We have multiple `\endgroups` for the same `\begingroup` because they occur on different branches of the compound conditional.

```

917 \begingroup
918 \def\@tempa{roman}%
919 \ifx#1\@tempa
920 \endgroup % first branch \endgroup

```

If `\@temp@sub` is `roman`, we change it to `upright`.

```

921 \def#1{upright}%
922 \let\@tempseries\mddefault
923 \let\@tempshape\shapedefault
924 \else
925 \def\@tempa{upright}%
926 \ifx#1\@tempa
927 \endgroup % second branch \endgroup
928 \let\@tempseries\mddefault
929 \let\@tempshape\shapedefault
930 \else
931 \def\@tempa{italic}%
932 \ifx#1\@tempa
933 \endgroup % third branch \endgroup
934 \let\@tempseries\mddefault
935 \let\@tempshape\itdefault
936 \else
937 \def\@tempa{bold}%
938 \ifx#1\@tempa
939 \endgroup % fourth branch \endgroup
940 \let\@tempseries\bfdefault
941 \let\@tempshape\shapedefault
942 \else
943 \def\@tempa{bolditalic}%
944 \ifx#1\@tempa
945 \endgroup % fifth branch \endgroup

```

```

946             \let\@tempseries\bfdefault
947             \let\@tempshape\itdefault
948         \else
Otherwise, the user specified a bad suboption.
949             \endgroup % sixth branch \endgroup
950             \@suboptionpresentfalse
951             \M@InvalidSuboptionError{#1}%
952         \fi
953     \fi
954 \fi
955 \fi
956 \fi
957 % no \fi at this level of indentation
958 \fi}

```

Helper macro to parse the shape identifier if it contains a / character.

```

\M@split@slash 959 \def\M@split@slash#1/#2\@nil{%
960     \def\@tempseries{#1}%
961     \def\@tempshape{#2}}

```

Helper macros for processing asterisks in the shape identifier. The first macro here should be called inside a group since it uses `\@tempa` and `\@tempb` and therefore will mess with temporary assignments otherwise.

```

\M@split@star 962 \def\M@split@star#1*#2\@nil{%
963     \def\@tempa{#1}%
964     \def\@tempb{#2}}
\M@strip@star 965 \def\M@strip@star#1*{#1}

```

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

```

966 \long\def\edef@nospace#1#2{%
967     \edef#1{\expandafter\zap@space\expanded{#2} \@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 font changes. We have three main user-level commands in this section: `\mathfont`, which makes changes for math mode; `\mainfont`, which makes changes for horizontal mode; and `\documentfont`, which calls both `\mathfont` and `\mainfont`. The `\mainfont` command is straightforward: set `\rmdefault` to be the font family corresponding to the

user’s argument and, when necessary, call `\selectfont` to change the font family in use. (So unlike `\mathfont`, `\mainfont` can result in font loading rather than just font declaration.)

The `\mathfont` command serves as the primary font-changing command for this package and is more complicated than `\mainfont`. This command is a wrapper around `\@mathfont`, the internal command that does the actual font changing, and when a user calls `\mathfont`, the `\@mathfont` macro carries out the following tasks:

1. Call `\M@newfont` on the mandatory argument of `\mathfont`, and store `\M@count` values.
2. Loop through the optional argument of `\mathfont` and determine NFSS series and shape codes from any suboptions using macros from the previous section.
3. On each iteration, check whether `mathfont` added a symbol font to the NFSS that uses the series and shape corresponding to the current suboption (or the default series and shape if there is no suboption). If not, call `\DeclareSymbolFont` to add the symbol font or raise a “Can’t declare new font shapes after `\begin{document}`” error.
4. Call `\M@<keyword>@set` to actually change the font.

Each `\M@<keyword>@set` macro is a wrapper around `\Umathcode` declarations and is defined in the last section of this document.

For a given `<keyword>`, `\M@fontinfo@<keyword>` stores the human-readable name of the new default font for `<keyword>` as well as the NFSS series and shape identifiers. (This is the name in `\@tempbase`, not `\M@fontn@me`.) We use this information for writing messages to the user. Additionally, `\M@<keyword>@shape` holds the series and shape pair for `<keyword>`. If the user specified a suboption, the contents of this macro come from the suboption via `\M@parse@sub`, and if the user did not specify a suboption, the information comes from `\<keyword>@default`.

For a combination of ``, `<series>`, and `<shape>` identifiers, `mathfont` calls the associated symbol font `M<count>-<series>/<shape>`, where `<count>` is the `\M@count` value associated to the ``, i.e. the contents of `\M@fontid@`. The `<series>` and `<shape>` values should be entries in the NFSS. For example, calls to `\mathfont` will often result symbol font names like `M0-m/n`. Including a count value in the symbol font name serves two purposes. First, it enables consistent formatting of symbol font names regardless of underlying NFSS family names. In particular, using the built-in fontloader vs. `fontspec` will result in different family names for the same font face with the same OpenType features, but the symbol font names will still match. Second,

it simplifies the symbol font names and makes them human readable regardless of the underlying NFSS family names, which may be complicated.

The last two user-level commands in this section are `\mathconstantsfont` and `\mathfontshapes`. The first of these two commands only works in Lua \TeX and makes \TeX use the math parameters from a given font when formatting equations. Traditional \TeX expects to see extra parameters in the font(s) in `\langle math style \rangle font2` and `\langle math style \rangle font3`, and it uses those parameters to format equations. Lua \TeX can pull these extra parameters from the fonts in any math family, and `\mathconstantsfont` tells Lua \TeX to do so for a given font family. The command `\mathfontshapes` declares extra font shapes for the NFSS as well as extra symbol fonts. The purpose of this command is to allow the user to declare symbol fonts in the document preamble for use after `\begin{document}`.

We begin by coding `\mainfont`. This command is a wrapper around `\@mainfont`.

```
\mainfont 968 \protected\def\mainfont{\@testopt{\@mainfont}{rm}}
```

Now the internal `\@mainfont` command. This command doesn't do anything in math mode, so we include `\@nomath`. We check whether `#1` is one of `rm`, `sf`, `tt`, or empty. If not, we issue an error and change `\@tempa` to `rm`. We use `\@tempswa` to check whether we are storing the font family name in `\langle type \rangle default` macros.

```
\@mainfont 969 \def\@mainfont[#1]#2{%
970   \@nomath\mainfont
971   \M@newfont{#2}%
972   \edef@nospace\@tempa{#1}%
973   \@tempswatrue
974   \def\@tempb{rm}%
975   \ifx\@tempa\@tempb
976   \else
977     \def\@tempb{sf}%
978     \ifx\@tempa\@tempb
979     \else
980       \def\@tempb{tt}%
981       \ifx\@tempa\@tempb
982       \else
983         \ifx\@tempa\@empty
984         \@tempswafalse
985         \else
986         \M@FamilyTypeError\@tempa
987         \def\@tempa{rm}%
988       \fi
```

```

989     \fi
990   \fi
991 \fi

```

Now close the group and save the font family in `\(#1)default` and change the default family to #1.

```

992 \if@tempswa
993   \expandafter\let\csname\@tempa default\endcsname\M@f@ntn@me
\familydefault 994   \edef\familydefault{\expandafter\noexpand
995     \csname\@tempa default\endcsname}%
996 \fi

```

If the current font is not `\nullfont` or `\M@f@ntn@me`, select `\M@f@ntn@me` as the font family.

```

997 \expandafter\ifx\the\font\nullfont
998 \else
999   \ifx\f@family\M@f@ntn@me
1000 \else
1001   \fontfamily\M@f@ntn@me\selectfont
1002 \fi
1003 \fi}

```

Now we come to `\mathfont`. This macro is a wrapper around `\@mathfont` that we use to check for an optional argument. The default argument is `\M@defaultkeys`.

```

1004 \protected\def\mathfont{\@testopt{\@mathfont}{\M@defaultkeys}}

```

The internal font-changing command. We call `\M@newfont` on the mandatory argument of `\mathfont`, which stores the NFSS family name(s) in `\M@f@ntn@me` and `\M@f@ntn@meb@se`. We check whether each family name corresponds to a value of `\M@newcount`, and if not, we define it. Throughout the definition of `\mathfont`, `\@tempa` stores the value of `\M@count` that corresponds to the #1 font, and `\@tempb` stores the value of `\M@count` that corresponds to the #1 font in base mode.

```

1005 \def\@mathfont[#1]#2{%
1006   \wlog{}%
1007   \M@newfont{#2}%

```

Temporarily store values of `\M@count`.

```

1008   \edef\@tempa{\csname M@fontid@\M@f@ntn@me\endcsname}%
1009   \edef\@tempb{\csname M@fontid@\M@f@ntn@meb@se\endcsname}%

```

Expand, zap spaces from, and store the optional argument in `\@tempc`, and then perform the loop. (At that point, we do not need `\@tempc` anymore.) We store the current keyword-suboption pair in `\@ci` and feed it to `\M@parse@option`.

```

1010 \edef@nospace\@tempc{#1}%
1011 \@for\@i:=\@tempc\do{%
1012     \expandafter\M@parse@option\@i=\@nil
1013     \if@optionpresent

```

If the user did not specify a suboption, parse the default option, and use that instead. We set `\@suboptionpresent` to true before calling `\M@parse@sub` so that we can check whether the default shape identifier is valid. If `\@suboptionpresent` is false after `\M@parse@sub`, we use `m/n` as the series/shape pair.

```

1014     \if@suboptionpresent
1015     \else
1016         \@suboptionpresenttrue
1017         \expandafter\M@parse@sub
1018         \csname\@temp@opt default\endcsname
1019         \if@suboptionpresent
1020         \else
1021             \let\@tempseries\mddefault
1022             \let\@tempshape\shapedefault
1023         \fi
1024     \fi

```

Now store the series and shape in `\M@<option>shape`.

```

1025     \expandafter\edef\csname M@\@temp@opt shape\endcsname{%
1026         \@tempseries/\@tempshape}%

```

At this point we have the information we need to declare the symbol font, namely the NFSS family (`\M@f@ntn@me` or `\M@f@ntn@meb@se`), series (`\@tempseries`), and shape (`\@tempshape`). We check if the symbol font we want to use is defined, and if not, we define it. We have two cases to consider: if `\M@base@` is true, we use the base-mode version of the font (corresponding to information in `\@tempb` and `\M@f@ntn@meb@se`), and if `\M@base@` is false, we use the default-mode version of the font (corresponding to information in `\@tempa` and `\M@f@ntn@me`). We let `\@tempc` be the count value in use for the current iteration of the loop.

```

1027     \ifM@mode@ % if default/node mode
1028         \let\@tempc\@tempa
1029         \ifcsname symM\@tempa-\@tempseries/\@tempshape\endcsname

```

If the symbol font has not been declared, check that we are still in the preamble. If no, issue an error message.

```

1030     \else
1031         \ifx\@onlypreamble\@notprerr
1032         \M@FontShapesError

```

Otherwise, we declare the symbol font.

```

1033         \else
1034             \M@SymbolFontInfo{\M@f@ntn@me}
1035             {\@tempseries}{\@tempshape}%
1036             \M@addto@symbolfonts
1037             {M\@tempa-\@tempseries/\@tempshape}
1038             {\@tempbase}{\@tempseries}{\@tempshape}%
1039             \DeclareSymbolFont
1040             {M\@tempa-\@tempseries/\@tempshape}{TU}
1041             {\M@f@ntn@me}{\@tempseries}{\@tempshape}%
1042         \fi
1043     \fi

```

Now do the same thing for default (node) mode.

```

1044     \else          % if default/node mode
1045         \let\@tempc\@tempb
1046         \ifcsname
1047             symM\@tempb-\@tempseries/\@tempshape\endcsname
1048         \else
1049             \ifx\@onlypreamble\@notprerr
1050                 \M@FontShapesError

```

The only difference is we use different font family and symbol font names.

```

1051         \else
1052             \M@SymbolFontInfo{\M@f@ntn@meb@se}
1053             {\@tempseries}{\@tempshape}%
1054             \M@addto@symbolfonts
1055             {M\@tempb-\@tempseries/\@tempshape}
1056             {\@tempbase(base)}{\@tempseries}
1057             {\@tempshape}%
1058             \DeclareSymbolFont
1059             {M\@tempb-\@tempseries/\@tempshape}{TU}
1060             {\M@f@ntn@meb@se}{\@tempseries}{\@tempshape}%
1061         \fi
1062     \fi
1063 \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.

```

1064     \expandafter\edef
1065     \csname M@fontinfo@\@temp@opt\endcsname{%
1066         {\@tempbase}{\@tempseries}{\@tempshape}}%
1067     \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. We make sure that information gets added to `\M@localfonts` (a macro that tracks the font names used for local font changes).

```

1068     \@tfor\@j:={bb}{cal}{frak}{bcal}{bfrak}\do{%
1069         \ifx\@temp@opt\@j
1070             \M@addto@localfonts{\expandafter\string
1071                 \csname math\@temp@opt\endcsname}
1072             {\@tempbase}{\@tempseries}{\@tempshape}%
1073         \@break@tfor
1074     \fi}%

```

And now the magic happens!

```

1075     \csname M@\@temp@opt @set\endcsname % set default font
1076     \csname M@\@temp@opt true\endcsname % set switch to true
1077 \fi}%

```

Display concluding messages for the user.

```

1078 \ifx\@tempa\@empty
1079     \wlog{The \string\mathfont\space command on line
1080         \the\inputlineno\space did not change the font
1081         for any characters!}%
1082 \else
1083     \typeout{:: mathfont :: Using font \@tempbase\space
1084         on line \the\inputlineno.}%
1085 \fi
1086 \wlog{}}

```

Using `\documentfont` calls `\mainfont`, `\mathfont`, and `\mathconstantsfont`. It also calls `\mathfontcommands` if the user is still in the preamble. The optional argument gets fed directly to `\@mainfont`.

```
\documentfont 1087 \protected\def\documentfont{\@testopt{\@documentfont}{rm}}
```

The internal command.

```

\@documentfont 1088 \def\@documentfont[#1]#2{%
1089     \@mainfont[#1]{#2}%
1090     \mathfont{#2}%
1091     \ifdefined\directlua
1092         \mathconstantsfont{#2}%
1093     \fi
1094     \ifx\@onlypreamble\@notprerr
1095     \else
1096         \mathfontcommands{#2}%
1097     \fi}

```

For backwards compatibility, we make `\setfont` expand to `\documentfont` (plus a warning message).

```
\setfont 1098 \protected\def\setfont{%
1099   \PackageWarningNoLine{mathfont}
1100   {Using \string\setfont\space is deprecated; I\MessageBreak
1101    replaced it with \string\documentfont}%
1102   \documentfont}
```

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 Math-Constants table. It issues a warning if called without font adjustments enabled. First, we check for an optional argument, which should be a shape identifier.

```
\M@SetMathConstant 1103 \let\M@SetMathConstants\relax
\mathconstantsfont 1104 \protected\def\mathconstantsfont{%
1105   \@testopt{\@mathconstantsfont}{upright}}
```

The internal command that does the processing. We begin by feeding the #2 argument to `\M@newfont` and parsing the #1 argument.

```
\@mathconstantsfon 1106 \def\@mathconstantsfont[#1]#2{%
1107   \M@newfont{#2}%
1108   \edef@nospace\@tempa{#1}%
1109   \M@parse@sub\@tempa
```

Store the family, series, and shape information. Because it doesn't make sense to use a font that is loaded with node mode, we force use of the base-mode version of the font regardless of the value of `\ifM@mode@`.

```
\m@th@const@nts@f@ 1110 \let\m@th@const@nts@f@mily\M@fontn@meb@se
\m@th@const@nts@se 1111 \let\m@th@const@nts@series\@tempseries
\m@th@const@nts@sh 1112 \let\m@th@const@nts@sh@pe\@tempshape
```

Temporarily store the value of `\M@count`.

```
1113 \edef\@tempa{\csname M@fontid@\m@th@const@nts@f@mily\endcsname}%
```

Now check whether the desired symbol font has been declared. If no, we declare it or issue an error.

```
1114 \ifcsname symM\@tempa-\@tempseries/\@tempshape\endcsname
1115 \else
1116   \ifx\@onlypreamble\@notprerr
1117     \M@FontShapesError
1118   \else
```

Declare the symbol font.

```
1119   \M@SymbolFontInfo{\m@th@const@nts@f@mily}
1120   {\@tempseries}{\@tempshape}%
1121   \M@addto@symbolfonts
```

```

1122      {M\@tempb-\@tempseries/\@tempshape}
1123      {\@tempbase(base)}{\@tempseries}{\@tempshape}%
1124      \DeclareSymbolFont
1125      {M\@tempa-\@tempseries/\@tempshape}{TU}
1126      {\m@th@const@nts@f@mily}{\@tempseries}{\@tempshape}%
1127      \fi
1128      \fi

```

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

We initialize `\M@SetMathConstants` to be `\relax`, 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`.

```

1129      \ifx\M@SetMathConstants\relax
\M@SetMathConstant 1130      \protected\def\M@SetMathConstants{%
1131          \begingroup
1132          \escapechar\m@ne
1133          \expandafter\getanddefine@fonts
1134          \csname symM%
1135              \csname M@fontid@\m@th@const@nts@f@mily\endcsname
1136              -\m@th@const@nts@series/\m@th@const@nts@sh@pe
1137          \expandafter
1138          \endcsname % expands to e.g. \symM0-m/n
1139          \csname TU/\m@th@const@nts@f@mily
1140              /\m@th@const@nts@series

```

```

1141          /\m@th@const@nts@sh@pe
1142      \endcsname % expands to \TU/<family>/<series>/<shape>
1143      \globaldefs\@ne
1144      \expandafter\@gobbletwo\math@fonts % avoid infinite loop
1145      \endgroup}%
1146  \fi
1147  \ifM@adjust@font
1148  \else
1149      \M@LuaTeXOnlyWarning\mathconstantsfont
1150  \fi}
\math@fonts 1151 \def\math@fonts{\aftergroup\M@SetMathConstants}

```

Now `\mathfontshapes`. This macro adds extra font shapes to the NFSS and defines symbol fonts. Its purpose is to allow the user to easily declare symbol fonts in the preamble without using them right away. The user-level command is a wrapper around `\@mathfontshapes`.

```

\mathfontshapes 1152 \protected\def\mathfontshapes{\@testopt{\@mathfontshapes}
1153     {upright,upright*,italic,bold}}

```

For the internal command, we feed the font name to `\M@newfont` and then loop through the optional argument. For each shape identifier in the optional argument, we parse it and then use it to declare a symbol font. This macro is very similar to parts of `\@mathfont`.

```

\@mathfontshapes 1154 \protected\def\@mathfontshapes[#1]#2{%
1155     \wlog{}%
1156     \M@newfont{#2}%

```

As in `\@mathfont`, we temporarily store values of `\M@count`.

```

1157     \edef\@tempa{\csname M@fontid@\M@fontn@me\endcsname}%
1158     \edef\@tempb{\csname M@fontid@\M@fontn@meb@se\endcsname}%

```

Expand, zap spaces, and loop through the optional argument.

```

1159     \edef\@nospace\@tempc{#1}%
1160     \@for\@i:=\@tempc\do{%
1161         \@suboptionpresenttrue
1162         \M@parse@sub\@i

```

Then check whether the symbol font exists and if not, declare it. We start with default/node mode. We print a message in the log file, add the information to `\M@symbolfonts`, and call `\DeclareSymbolFont`.

```

1163     \if@suboptionpresent
1164         \ifM@mode@ % if default/node mode
1165             \ifcsname symM\@tempa-\@tempseries/\@tempshape\endcsname
1166             \else
1167                 \M@SymbolFontInfo{\M@fontn@me}

```

```

1168         {\@tempseries}{\@tempshape}%
1169     \M@addto@symbolfonts
1170         {M\@tempa-\@tempseries/\@tempshape}
1171         {\@tempbase}{\@tempseries}{\@tempshape}%
1172     \DeclareSymbolFont
1173         {M\@tempa-\@tempseries/\@tempshape}{TU}
1174         {\M@f@ntn@me}{\@tempseries}{\@tempshape}%
1175     \fi

```

And do the same thing for base mode.

```

1176     \else      % if base mode
1177         \ifcsname symM\@tempb-\@tempseries/\@tempshape\endcsname
1178     \else
1179         \M@SymbolFontInfo{\M@f@ntn@meb@se}
1180         {\@tempseries}{\@tempshape}%
1181     \M@addto@symbolfonts
1182         {M\@tempb-\@tempseries/\@tempshape}
1183         {\@tempbase(base)}{\@tempseries}{\@tempshape}%
1184     \DeclareSymbolFont
1185         {M\@tempb-\@tempseries/\@tempshape}{TU}
1186         {\M@f@ntn@meb@se}{\@tempseries}{\@tempshape}%
1187     \fi
1188 \fi
1189 \fi}}
1190 \@onlypreamble\mathfontshapes
1191 \@onlypreamble\mathfontshapes

```

7 Local Font Changes

This section deals with local font changes. The main user-level macro in this section is `\newmathfontcommand`, which creates macros that change the font for math alphabet characters and is basically a wrapper around `\DeclareMathAlphabet`. Other user-level commands are a special case of this one.

We begin with two helper macros. First is `\M@check@csarg`, which accepts two arguments and handles some error checking. The `#1` argument is a user-level command that we use in 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 (number of tokens) by seeing if `\@gobble` eats it completely; and (2) check that the argument is a control sequence. If the user specifies an argument of the form `{. .}`, i.e. extra text inside braces,

the `\ifcat` will catch it and issue an error. If `\M@check@csarg` likes the input, it sets `\ifM@arg@good` to true, and otherwise, it sets `\ifM@arg@good` to false.

```
\M@check@csarg 1192 \def\M@check@csarg#1#2{%
1193   \expandafter\ifx\expandafter\@nnil\@gobble#2\@nnil % good
1194   \ifcat\relax\noexpand#2 % good
1195     \M@arg@goodtrue
1196   \else % if #2 not a control sequence
1197     \M@MissingCSError#1{\detokenize{#2}}
1198     \M@arg@goodfalse
1199   \fi
1200 \else % if #2 is multiple tokens
1201   \M@MissingCSError#1{\detokenize{#2}}
1202   \M@arg@goodfalse
1203 \fi}
```

The macro `\M@checkspecials` 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, for example, `\mathfont[bb]`, so we do some processing analogous to what happens inside `\@mathfont`.

```
\M@checkspecials 1204 \def\M@checkspecials#1#2{%
1205   \in@#1{\mathbb\mathcal\mathfrak\mathbbcal\mathbbfrak}
1206   \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`.

```
1207   \begingroup
1208   \escapechar\m@ne
1209   \expandafter
1210 \endgroup
1211 \expandafter\edef\expandafter\@tempa\expandafter{%
1212   \expandafter\@gobblefour\string#1}%
```

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

```
1213   \@mathfontinfo{Interpreting your new macro \string#1\space
1214     as \@tempa\space chars.}%
1215   \@mathfontinfo{Setting \expandafter\string
1216     \csname ifM@\@tempa\endcsname\space to true.}%
1217   \csname M@\@tempa true\endcsname
```

And store the information to write to the log file `\AtBeginDocument`.

```
1218   \expandafter\edef\csname M@fontinfo@\@tempa\endcsname{%
```

```

1219      {\@tempbase}{\@tempseries}{\@tempshape}}%
1220      \expandafter\edef\csname M@\@tempa shape\endcsname
1221      {\@tempseries/\@tempshape}%
1222      \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. Finally, add #1 and #2 to the list of local font-change commands.

```

\newmathfontcommand 1223 \protected\def\newmathfontcommand#1#2#3#4{%
1224     \M@check@csarg\newmathfontcommand{#1}%
1225     \ifM@arg@good
1226         \M@newfont{#2}
1227         \M@NewFontCommandInfo{#1}{\M@f@ntn@meb@se}{#3}{#4}
1228         \DeclareMathAlphabet{#1}{TU}{\M@f@ntn@meb@se}{#3}{#4}
1229         \M@checkspecials{#1}{\@tempbase}
1230         \M@addto@localfonts{\string#1}{\@tempbase}{#3}{#4}
1231     \fi}
1232 \@onlypreamble\newmathfontcommand

```

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@ 1233 \def\M@define@newmath@cmd#1#2#3{%
1234     \protected\def#1##1##2{%
\M@check@csarg 1235         \M@check@csarg{#1}{##1}
1236         \newmathfontcommand{##1}{##2}{#2}{#3}}}
\M@default@newmath 1237 \def\M@default@newmath@cmds{%

```

```

1238 \newmathrm{\mddefault}{\shapedefault},%
1239 \newmathit{\mddefault}{\itdefault},%
1240 \newmathbf{\bfdefault}{\shapedefault},%
1241 \newmathbfit{\bfdefault}{\itdefault},%
1242 \newmathsc{\mddefault}{\scdefault},%
1243 \newmathscit{\mddefault}{\scdefault\itdefault},%
1244 \newmathbfsc{\bfdefault}{\scdefault},%
1245 \newmathbfscit{\bfdefault}{\scdefault\itdefault}}
1246 \@for\@i:=\M@default@newmath@cmds\do{%
1247   \expandafter\M@define@newmath@cmd\@i}
1248 \@onlypreamble\newmathrm
1249 \@onlypreamble\newmathit
1250 \@onlypreamble\newmathbf
1251 \@onlypreamble\newmathbfit
1252 \@onlypreamble\newmathsc
1253 \@onlypreamble\newmathscit
1254 \@onlypreamble\newmathbfsc
1255 \@onlypreamble\newmathbfscit
1256 \@onlypreamble\M@define@newmath@cmd
\M@default@newmath 1257 \let\M@default@newmath@cmds\@undefined

```

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

```

\mathfontcommands 1258 \protected\def\mathfontcommands#1{%
1259   \newmathrm\mathrm{#1}
1260   \newmathit\mathit{#1}
1261   \newmathbf\mathbf{#1}
1262   \newmathbfit\mathbfit{#1}
1263   \newmathsc\mathsc{#1}
1264   \newmathscit\mathscit{#1}
1265   \newmathbfsc\mathbfsc{#1}
1266   \newmathbfscit\mathbfscit{#1}}
1267 \@onlypreamble\mathfontcommands

```

8 Miscellaneous

We begin this section with the user-level macros that provide information for Lua-based font adjustments. We define a macro `\M@check@int` to determine if `#1` is a nonnegative integer and set the switch `\M@arg@good` accordingly. Checking happens in three stages inside a group. Immediately after the group, we will check whether `\@tempb` is `\@empty`, so to force the switch to be false,

we sometimes set `\@tempb` to `\@nnil` inside the group. The first stage is a check whether `#1` is empty.

```

1268 \ifM@adjust@font
\M@check@int 1269 \def\M@check@int#1{%
1270     \begingroup
1271     \def\@tempa{#1}%
1272     \ifx\@tempa\@empty
1273     \let\@tempb\@nnil

```

If `#1` is not empty, we check whether it is a nonnegative integer. If `#1` is a nonnegative integer, the entirety of `0#1` becomes the value of `\count@`, and `\@tempb` will end up empty. Otherwise, `\@tempb` will be nonempty. The use of `\afterassignment` here is inspired by `\@defaultunits` from the kernel.

```

1274     \else
1275     \def\@tempa##1\relax\@nil{\def\@tempb{##1}}%
1276     \afterassignment\@tempa
1277     \count@=0#1\relax\@nil

```

If `\@tempb` is nonempty, we handle the case where `#1` is an octal or hexadecimal integer. We use `\if` to check whether the first character of `#1` is `'` or `"`, and if yes, we replace it with `'0` or `"0` and repeat the same check with `\@tempb`. The `\relax` prevents `\if` from scanning past `#1` if `#1` expands to something empty. The `\remove@to@nnil` gobbles everything in `#1` (except the `'` or `"`, which is eaten by `\if`) when the test is successful, and it gobbles the `\@nnil` after the `\else` when the test is unsuccessful.

```

1278     \ifx\@tempb\@empty
1279     \else
1280     \expandafter\remove@to@nnil\if'#1\relax
1281     \@nnil
1282     \afterassignment\@tempa
1283     \count@'0\expandafter\@gobble
1284     \expanded{#1}\relax\@nil
1285     \else
1286     \@nnil
1287     \expandafter\remove@to@nnil\if"#1\relax
1288     \@nnil
1289     \afterassignment\@tempa
1290     \count@"0\expandafter\@gobble
1291     \expanded{#1}\relax\@nil

```

If `#1` is neither octal nor hexadecimal, we check whether it starts with `\numexpr`. This case is good, so we set `\@tempb` to `\@empty`.

```

1292     \else

```

```

1293         \@nnil
1294         \expandafter\remove@to@nnil\ifx\numexpr#1\@nnil
1295         \let\@tempb\@empty

```

The last possibility is if #1 has the form ``<character>`. To allow for the possibility of, for example, ``\%`, we check whether #1 begins with ``` and contains two tokens. This is not foolproof, and sufficiently bad input, such as ``\relax`, will break this macro. (So don't do that.) Checking the catcode of the second token in #1 doesn't seem worth the effort. If #1 is a single ``` character, then `\@gobbletwo` will eat #1 and `\@nnil`, so the `\ifx` will compare `\@nnil` and `\relax`.

```

1296         \else
1297         \@nnil
1298         \expandafter\remove@to@nnil\if`#1\relax
1299         \@nnil
1300         \expandafter\ifx\expandafter\@nnil
1301         \@gobbletwo#1\@nnil\relax
1302         \let\@tempb\@empty
1303         \else
1304         \let\@tempb\@nnil
1305         \fi
1306         \else
1307         \@nnil
1308         \let\@tempb\@nnil
1309         \fi
1310         \fi
1311         \fi
1312         \fi
1313         \fi
1314         \fi
1315         \expandafter
1316         \endgroup
1317         \ifx\@tempb\@empty
1318         \M@arg@goodtrue
1319         \else
1320         \M@arg@goodfalse
1321         \fi}

```

Making the `\rulethicknessfactor`, etc. counts accessible to the user means that we don't need `\RuleThicknessFactor` and friends anymore, but we keep them for backwards compatibility and convenience. Each of these commands uses `\M@check@int` to check its argument, then calls the appropriate other commands.

```

\RuleThicknessFact 1322 \protected\def\RuleThicknessFactor#1{%
1323     \M@check@int{#1}%
1324     \ifM@arg@good
1325         \rulethicknessfactor=#1\relax
1326     \else
1327         \M@BadIntegerError\RuleThicknessFactor{\detokenize{#1}}%
1328     \fi}

\SurdHorizontalFac 1329 \protected\def\SurdHorizontalFactor#1{%
1330     \M@check@int{#1}%
1331     \ifM@arg@good
1332         \hsurdfactor=#1\relax
1333     \else
1334         \M@BadIntegerError\SurdHorizontalFactor{\detokenize{#1}}%
1335     \fi}

\SurdVerticalFacto 1336 \protected\def\SurdVerticalFactor#1{%
1337     \M@check@int{#1}%
1338     \ifM@arg@good
1339         \vsurdfactor=#1\relax
1340     \else
1341         \M@BadIntegerError\SurdVerticalFactor{\detokenize{#1}}%
1342     \fi}

```

For the integral italic factor, we input the information to `\charmline`

```

\IntegralItalicFac 1343 \protected\def\IntegralItalicFactor#1{%
1344     \M@check@int{#1}%
1345     \ifM@arg@good
1346         \charmline{0x222B * * * * * * * * *
1347                        * * * * * * * * *
1348                        * * * * * * * * *
1349                                * * #1}%
1350     \else
1351         \M@BadIntegerError\IntegralItalicFactor{\detokenize{#1}}%
1352     \fi}

```

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

```

1353 \else
1354     \@tfor\@i:=\RuleThicknessFactor\IntegralItalicFactor
1355         \SurdHorizontalFactor\SurdVerticalFactor\charmline
1356         \charmfile\CharmLine\CharmFile\CharmInfo\CharmType
1357         \do{%
1358             \protected\expandafter\edef\@i{%

```

```

1359         \noexpand\M@NoFontAdjustError\expandafter\noexpand\@i
1360         \noexpand\@gobble}}
\charminfo 1361 \protected\def\charminfo{\M@NoFontAdjustError\charminfo
1362         \begingroup
1363         \afterassignment\endgroup
1364         \count@}
\charmtype 1365 \protected\def\charmtype{\M@NoFontAdjustError\charmtype
1366         \begingroup
1367         \afterassignment\endgroup
1368         \count@}
1369 \fi

```

These commands should appear in the preamble only.

```

1370 \@onlypreamble\charmline
1371 \@onlypreamble\charmfile
1372 \@onlypreamble\CharmLine
1373 \@onlypreamble\CharmFile
1374 \@onlypreamble\RuleThicknessFactor
1375 \@onlypreamble\IntegralItalicFactor
1376 \@onlypreamble\SurdHorizontalFactor
1377 \@onlypreamble\SurdVerticalFactor

```

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. This is due to fundamental aspects of how \TeX processes math-mode material.

```

1378 \protected\gdef\clap#1{\hb@xt@{z@{\hss#1\hss}}
\stack@flatrel 1379 \protected\def\stack@flatrel#1#2{\expandafter
1380     \st@ck@fl@trel\expandafter#1\@firstofone#2}
\st@ck@fl@trel 1381 \protected\def\st@ck@fl@trel#1#2#3{%
1382     {\setbox0\hbox{#1#2\m@th$}% contains \mathrel symbol
1383     \setbox1\hbox{#1#3\m@th$}% gets raised over \box0
1384     \if\wd0>\wd1\relax
1385         \hb@xt@{\wd0}{%
1386             \hfil
1387             \clap{\raise0.7\ht0\box1}%
1388             \clap{\box0}\hfil}%
1389     \else
1390         \hb@xt@{\wd1}{%
1391             \hfil
1392             \clap{\raise0.7\ht0\box1}%

```

```

1393     \clap{\box0}\hfil}%
1394 \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.

```

1395 \ifnum\tracinglostchars<\tw@
1396   \tracinglostchars\tw@
1397 \fi

```

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

```

1398 \AtBeginDocument{%
1399   \ifcase\M@num@localfonts
1400   \or
1401     \def\@tempa#1#2#3\@nil{#2}
1402     \wlog{}
1403     \typeout{:: mathfont :: Using
1404       \expandafter\@tempa\M@localfonts\@nil\space
1405       for local font changes.}
1406   \else
1407     \wlog{}
1408     \typeout{:: mathfont :: Using \the\M@num@localfonts\space
1409       fonts for local font changes.}
1410   \fi}

```

A helper macro for printing messages `\AtBeginDocument`. This macro adds characters from `#1` into `\@tempa` until the length of `\@tempa`, as measured by `\count@` becomes `#2 - 2`. If `\@tempa` has too many characters to fit, `\M@addtab@count@tempa` instead ends `\@tempa` with an ellipsis. Then the macro appends spaces to the end of `\@tempa` until it is `#2` characters long, again measured using `\count@`.

```

1411 \def\M@addtab@count@tempa#1#2{%
1412   \let\@tempb\@empty
1413   \@tempcntb\z@
1414   \@tfor\@i:=#1\do{%
1415     \ifnum\count@=\numexpr#2 - 5\relax
1416       \edef\@tempb{\@tempb\@i}
1417       \advance\@tempcntb\@ne
1418     \else
1419       \edef\@tempa{\@tempa\@i}
1420       \advance\count@\@ne
1421     \fi}
1422   \ifnum\@tempcntb<4\relax
1423     \edef\@tempa{\@tempa\@tempb}

```

```

1424     \advance\count@\@tempcntb
1425 \else
1426     \edef\@tempa{\@tempa...}
1427     \advance\count@\thr@@
1428 \fi
1429 \@whilenum\count@<#2\do{%
1430     \edef\@tempa{\@tempa\space}
1431     \advance\count@\@ne}}

```

Write to the log file `\AtBeginDocument` all font changes carried out by `mathfont`. The command `\M@fontinfo@begin` accepts a keyword as its `#1` argument and prints a message on the log file showing whether `mathfont` set a default font for that keyword and, if yes, the name, series, and shape for that font.

```

\M@fontinfo@begin 1432 \def\M@fontinfo@begin#1{%
1433     \expandafter\ifx % next lines are two cs to be compared
1434         \csname ifM@#1\expandafter\endcsname
1435         \csname iftrue\endcsname
1436     \expanded{\noexpand\M@fontinfo@begin@{#1}}
1437     \csname M@fontinfo@#1\endcsname}
1438 \else
1439     \bgroup
1440     \let\@tempa\@empty
1441     \count@\z@
1442     \M@addtab@count@tempa{#1}{35}
1443     \M@addtab@count@tempa{No\space change}{78}
1444     \wlog{\@tempa}
1445     \egroup
1446 \fi}

```

Helper macro that handles printing the message. The four arguments appear sequentially on the same line in the log file: `#1` is the keyword, `#2` is the font name, `#3` is the series, and `#4` is the shape. We allocate 18, 30, 15, and 15 characters respectively, for a total of 78 characters. The default width of the log file is 80 characters, so we should fit everything on one line that way.

```

\M@fontinfo@begin@ 1447 \def\M@fontinfo@begin@#1#2#3#4{%
1448     \bgroup
1449     \let\@tempa\@empty
1450     \count@\z@
1451     \M@addtab@count@tempa{#1}{18}
1452     \M@addtab@count@tempa{font:\space#2}{48}
1453     \M@addtab@count@tempa{series:\space#3}{63}
1454     \M@addtab@count@tempa{shape:\space#4}{78}

```

And print the message.

```
1455 \wlog{\@tempa}
1456 \egroup}
```

The macro `\M@localfonts@begin` does the same thing except for the local font-change commands.

```
\M@localfonts@begin 1457 \let\M@localfonts@begin\M@fontinfo@begin@
```

And for symbol fonts declared.

```
\M@symbolfonts@beg 1458 \let\M@symbolfonts@begin\M@fontinfo@begin@
```

The command for font families is different because we only have two pieces of information to display.

```
\M@families@begin 1459 \def\M@families@begin#1{%
1460 \bgroup
1461 \let\@tempa\@empty
1462 \count@z@
1463 \expanded{%
1464 \noexpand\M@addtab@count@tempa
1465 {\csname M@fontid@#1\endcsname}{5}
1466 \noexpand\M@addtab@count@tempa{#1}{78}}
1467 \wlog{\@tempa}
1468 \egroup}
```

Now print the messages. We start with the font families that `mathfont` uses. We don't need to store a list of font families in `\M@families` because we can simply increment the `fontid` count until we reach a value that is large enough to not have a corresponding font.

```
1469 \AtBeginDocument{%
1470 \wlog{^^J----- Changes made by mathfont
1471 in the preamble -----}
1472 \wlog{}
1473 \wlog{*****^^J%
1474 * Font families used *^^J%
1475 *****}
1476 \ifx\M@families\@empty
1477 \wlog{No font families declared by mathfont.}
1478 \else
1479 \@for\@i:=\M@families\do{%
1480 \expandafter\M@families@begin\expandafter{\@i}}
1481 \fi
1482 \wlog{}
```

Same thing for symbol fonts.

```
1483 \wlog{*****^^J%
```

```

1484      * Symbol fonts declared *^^J%
1485      *****}
1486 \ifx\M@symbolfonts\@empty
1487   \wlog{No symbol fonts declared by mathfont.}
1488 \else
1489   \@for\@i:=\M@symbolfonts\do{%
1490     \expandafter\M@symbolfonts@begin\@i}
1491 \fi
1492 \wlog{}

```

Character keywords.

```

1493 \wlog{*****^^J%
1494      * Keywords *^^J%
1495      *****}
1496 \@for\@i:=\M@keys\do{%
1497   \expandafter\M@fontinfo@begin\expandafter{\@i}}
1498 \wlog{}

```

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

```

1499 \wlog{*****^^J%
1500      * Local font-change commands *^^J%
1501      *****}
1502 \ifnum\M@num@localfonts=\z@
1503   \wlog{No local font change commands declared.}
1504 \else
1505   \@for\@j:=\M@localfonts\do{%
1506     \expandafter\M@localfonts@begin\@j}
1507 \fi
1508 \wlog{}
1509 \wlog{----- End of changes
1510      -----}
1511 \wlog{}

```

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

```

1512 \ifdefined\XeTeXrevision
1513   \ifM@font@loaded
1514     \AtEndOfPackage{%
1515       \PackageWarningNoLine{mathfont}
1516       {It looks like you specified a font\MessageBreak
1517        when you loaded mathfont. If you run\MessageBreak
1518        into problems with a font whose name\MessageBreak
1519        is multiple words, try using LuaLaTeX\MessageBreak
1520        or call \string\documentfont\space or

```



```

1521     \string\mathfont\MessageBreak
1522     manually}}
1523   \fi
1524 \fi

```

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

```

1525 \ifM@font@loaded
1526   \AtEndOfPackage{\documentfont\M@font@load}
1527 \fi

```

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 \TeX side of things and is documented next, and everything else happens inside `\directlua`. On the Lua side of things, we store all the functions and character metric information in the table `mathfont`. With the exception of a handful of integers used to track encoding slots, every entry in `mathfont` is either a function or a subtable indexed by an *integer*. The *integer* is a Unicode encoding number and indicates which Unicode character the subtable corresponds to. See tables 2 and 3 for a list of the functions in `mathfont` and the fields in character subtables. See section 10 for discussion of the callbacks for editing `fontdata` objects.

Changing the top-level `nomath` flag in a font object is easy. Creating a `MathConstants` table is complicated but largely self-contained. We take a

Table 2: Fields of Character Subtables in `mathfont`

Field	Type	In <code>a</code>	In <code>e</code>	Used For
<code>type</code>	string	Yes	Yes	Type is <code>a</code> or <code>e</code>
<code>data_rm</code>	table	Yes	Yes	Information for upright font shapes
<code>data_it</code>	table	Yes	Yes	Information for italic font shapes
<code>num_variants</code>	integer	No	Yes	Number of large variants
<code>smash</code>	integer	No	Yes	Encoding slot for smashed character
<code>next</code>	table	No	Yes	Encoding slots for large variants

few parameters that the user has set, define traditional \TeX math parameters based on the essential parameters of the font, and assign their values to the corresponding entries in a `MathConstants` table. However, editing character metrics during font loading is convoluted with many moving parts. For every glyph that we want modify, we store character metric information for that glyph as a subtable in `mathfont`. The entries of the subtable describe how to scale the bounding box, scale the glyph itself, or determine math accent placement. For characters of type `a` (“alphabet”), we specify information to stretch the bounding box (not the glyph) horizontally, so we equivalently add extra space around the character. For type `e` (“extensible”), we stretch the bounding box and glyph, so we create an ensemble of scaled versions, which we use as a family of large variants.

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

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

Table 3: Functions in mathfont

Function	Argument(s)	Used For
<code>strint</code>	number	Format number as a string
<code>new_type_a</code>	index, data	Add type a entry to mathfont
<code>new_type_e</code>	index, data	Add type e entry to mathfont
<code>add_to_charm</code>	string of charm info	Add charm info to mathfont
<code>parse_charm</code>	string of charm info	Split string, validate inputs
<code>parse_num</code>	numeric string	Parse numeric value
<code>empty</code>	none	Does nothing
<code>glyph_info</code>	character subtable	Get height, width, depth, italic
<code>make_a_commands</code>	index, offset	Return virtual font commands
<code>make_a_table</code>	index, charm data, fontdata	Make new subtable for type a character
<code>make_e_commands</code>	index, scale factors	Return virtual font commands
<code>make_e_table</code>	index, charm data, fontdata	Make new subtable for type e character
<code>smash_glyph</code>	index, fontdata	Make table for smashed char
<code>utf_16BE</code>	integer	Return UTF-16BE format
<code>adjust_font</code>	fontdata	Call callbacks
<code>apply_charm_info</code>	fontdata	Change character metrics in font
<code>math_constants</code>	fontdata	Create MathConstants table
<code>set_nomath_false</code>	fontdata	Set nomath (top-level flag in the font) to false
<code>get_font_name</code>	fontdata	Return font name
<code>info</code>	string	Write message in the log file

```

1528 \ifM@adjust@font
1529 \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.

```

1530 \bgroup
1531   \catcode`\~=0
1532   ~catcode`~\=12
1533   ~@firstofone{%
1534 ~egroup

```

```

1535 ~let~@tempa~%
1536 ~let~%~@percentchar
1537 ~def~M@empty@assert{"\n\n%
1538   Package mathfont error: Empty charm information.\n\n%
1539   Your argument for \\charmline is empty, or a\n%
1540   line in your \\charmfile is blank. Make sure\n%
1541   your calls to \\charmline and all lines in\n%
1542   your \\charmfile contain integers, floats,\n%
1543   and asterisks separated by commas or spaces.\n"}
1544 ~def~M@missing@assert{"\n\n%
1545   Package mathfont error: Missing charm entries.\n\n%
1546   I'm having trouble with a character metric.\n%
1547   Your \\charmline or \\charmfile contains\n%
1548   \"\".. temp_string .. ",\n%
1549   which looks to me like you provided an index\n%
1550   without any commas or spaces to specify the\n%
1551   numbers for charm values. Make sure that you\n%
1552   use commas or spaces to separate each entry\n%
1553   in your charm information.\n"}
1554 ~def~M@number@assert{"\n\n%
1555   Package mathfont error: Nonnumeric charm value.\n\n%
1556   I'm having trouble with a character metric.\n%
1557   Your \\charmline or \\charmfile contains \"\"
1558   .. s .. ",\n%
1559   which is not a number. Make sure that your\n%
1560   charm information is all integers, floats,\n%
1561   and asterisks separated by commas or spaces.\n"}
1562 ~def~M@index@assert{"\n\n%
1563   Package mathfont error: Invalid Unicode index.\n\n%
1564   The Unicode index \"\"
1565   .. temp_string .. "\" is invalid. Make sure\n%
1566   that the first entry in your \\charmline and in each\n%
1567   line of your \\charmfile is an integer between 0 and\n%
1568   1,114,111 (0x10FFFF), possibly with an exclamation\n%
1569   point or question mark.\n"}
1570 ~def~M@bound@assert{"\n\n%
1571   Package mathfont error: Exceeded Unicode table.\n\n%
1572   You asked me to do something with an encoding slot\n%
1573   number that exceeds the number of slots in the\n%
1574   Unicode table. You are probably seeing this error\n%
1575   because you declared too many type e characters.\n"}

```

We previously defined \% to contain a single % with catcode 12, so when we

put `\%` in an `\edef`, it becomes easy to get `%`'s inside the definition of our next two macros. This is important because it lets us use `string.format` to create informative error and warning messages using straightforward syntax. To insert the Unicode character in the messages, we use `utf8.char` function, which is the Lua code for producing arbitrary Unicode characters.

```

1576 ~edef~M@entries@assert{string.format("\n\n%
1577 Package mathfont error: Charm values too short.\n\n%
1578 Your charm information for U+~%X ~%s (index ~%d)\n%
1579 needs more entries. Right now you have ~%d\n%
1580 entries (besides the index), but you need at\n%
1581 least ~%d. If you aren't sure what to do, try\n%
1582 adding asterisks to your \\charmline or line in\n%
1583 your \\charmfile. See the user guide for more\n%
1584 information.\n",
1585 index, utf8.char(index), index,
1586 number_of_entries, entries_needed)}
1587 ~edef~M@entries@warning{string.format("\n\n%
1588 Package mathfont warning: Charm values too long.\n\n%
1589 Your charm information for U+~%X ~%s (index ~%d)\n%
1590 has more entries than it needs. Right now you\n%
1591 have ~%d entries (besides the index), but you\n%
1592 only need ~%d. This isn't a problem per se\n%
1593 because I can easily ignore the extra numbers,\n%
1594 but it may indicate confusion about Unicode\n%
1595 characters and charm values. See the user\n%
1596 guide for more information.\n\n",
1597 index, utf8.char(index), index,
1598 number_of_entries, entries_needed)}

```

Error message if the user tries to adjust the left side of the bounding box on a virtual character.

```

1599 ~edef~M@virtual@assert{string.format("\n\n%
1600 Package mathfont error: Can't adjust left side\n%
1601 of the bounding box on a virtual character.\n\n%
1602 Your charm information for U+~%X ~%s (index ~%d)\n%
1603 instructs me to change the left side of the bounding\n%
1604 box around this character. However, in the font\n%
1605 ~%s,\n%
1606 that character is a virtual character, and I'm not\n%
1607 programmed to change the left side of the bounding\n%
1608 box on a virtual character. To resolve this error,\n%
1609 try including\n%

```

```

1610  \\charmline{~%d 0 * * *}\\n%
1611  at the end of your document preamble.\\n\\n",
1612  index, utf8.char(index), index,
1613  mathfont.get_font_name(fontdata), index)}}
1614  \\let\\%\\@tempa

```

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`. The macros `\charminfo` and `\charmtype` print information from `mathfont` about the charm information currently in memory for certain characters.

```

1615  \\newluafunction\\addtocharm@
1616  \\newluafunction\\charminfo@
1617  \\newluafunction\\charmtype@
1618  \\directlua{%
1619    local t = lua.get_functions_table()
1620    t[\\number\\addtocharm@] = function()
1621      mathfont:add_to_charm(token.scan_string())
1622    end

```

We also define the Lua function `\charminfo@` for use in `\charminfo`. This function scans the following integer and prints the charm information for the Unicode character whose index is that integer.

```

1623  t[\\number\\charminfo@] = function()
1624    local temp = token.scan_int()
1625    temp = mathfont[temp]

```

If `mathfont` contains an entry for index `temp`, gather the charm information from that entry. We will print to \TeX a string in the same format as the argument of `\charmline`. We loop through `data_rm`, and on each iteration, we add that entry to a temporary string and then add italic charm information if it is different.

```

1626    if temp then
1627      local temp_str = ""
1628      local temp_rm = {}
1629      local temp_it = {}

```

We have two possibilities depending on the type of the input. For type `a` characters, all entries in `data_rm` and `data_it` are integers, and we can loop the list without issue. For type `e` characters, `data_rm` and `data_it` contain 15 two-entry subtables and three integers, so we have to flatten the list before we use it.

```

1630      for k,v in pairs(temp.data_rm) do

```

```

1631         if type(v) == "number" then
1632             temp_rm[k] = v * 1000
1633             temp_it[k] = temp.data_it[k] * 1000
1634         elseif type(v) == "table" then
1635             temp_rm[2*k-1] = v[1] * 1000
1636             temp_rm[2*k]   = v[2] * 1000
1637             temp_it[2*k-1] = temp.data_it[k][1] * 1000
1638             temp_it[2*k]   = temp.data_it[k][2] * 1000
1639         end
1640     end

```

Now we loop through `temp_rm` and add the contents to `temp_str`.

```

1641     for k,v in pairs(temp_rm) do
1642         if temp_str \noexpand~= "" then
1643             temp_str = temp_str .. " "
1644         end
1645         temp_str = temp_str .. mathfont.strint(v)

```

Now check the corresponding charm entry for italic fonts and add that if different from `v`.

```

1646         if temp_it[k] \noexpand~= v then
1647             temp_str = temp_str .. "/" ..
1648                 mathfont.strint(temp_it[k])
1649         end
1650     end

```

Now print the result.

```

1651     tex.print(temp_str)
1652 else
1653     tex.print("none")
1654 end
1655 end

```

Same thing for `\charmtype@`.

```

1656 t[\number\charmtype@] = function()
1657     local temp = token.scan_int()
1658     temp = mathfont[temp]
1659     if temp then
1660         tex.print(temp.type)
1661     else
1662         tex.print("none")
1663     end
1664 end}

```

```
\charmline 1665 \protected\def\charmline{\luafunction\addtocharm@}
```

The argument of `\charmfile` should be a valid filename, and we open it in `\M@Charm`. The macro processes each line of the file as a piece of charm information.

```
\charmfile 1666 \protected\def\charmfile#1{%
1667   \IfFileExists{#1}{%
1668     \begingroup
1669     \endlinechar\m@ne
1670     \immediate\openin\M@Charm{#1}
```

The macro `\next` reads a line into `#1`, feeds it to `\charmline`, and calls itself if the file has more lines. \TeX adds an extra line to the end of files it reads (why??), so we check whether the current line is empty before feeding it to `\charmline`. (The last line of the file is empty because we set `\endlinechar` to `-1`. Otherwise, the file would have a spurious `\par` at the end.)

```
1671   \def\next{%
1672     \read\M@Charm to \@tempa
1673     \ifx\@tempa\@empty
1674     \else
1675       \charmline\@tempa
1676     \fi
1677     \ifeof\M@Charm\else % if file has more lines?
1678       \expandafter\next
1679     \fi}
```

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

```
1680   \next
1681   \immediate\closein\M@Charm
1682   \endgroup}
```

If the file does not exist, raise an error.

```
1683   {\M@NoCharmFileError{\detokenize{#1}}}}
```

Alternative names.

```
\CharmLine 1684 \let\CharmLine\charmline
\CharmFile 1685 \let\CharmFile\charmfile
```

Now the macros `\charminfo` and `\charmtype`. The structure is a bit different because we want them to be fully expandable. The macros don't have an argument. Instead, they call the appropriate `\luafunction`, which scans the next integer and processes it.

```
\charminfo 1686 \def\charminfo{\luafunction\charminfo@}
\charmtype 1687 \def\charmtype{\luafunction\charmtype@}
```

The wrapped versions are more user-friendly. For `\CharmInfo`, we first check whether `mathfont` contains an entry with index `#1`.


```

\CharmInfo 1688 \protected\def\CharmInfo#1{%
1689   \M@check@int{#1}%
1690   \ifM@arg@good
1691     \begingroup                % \begingroup
1692     \edef\@tempa{\charinfo#1}%
1693     \def\@tempb{none}%
1694     \edef\@tempc{\number#1}%

```

Now the actual check. If there is no entry, we print a message saying so.

```

1695     \ifx\@tempa\@tempb
1696       \expandafter\endgroup % first branch \endgroup
1697       \expanded{\showtokens{no charm info assigned to
1698         index \@tempc}}}%

```

If yes, we print the charm information to the terminal.

```

1699     \else
1700       \expandafter\endgroup % second branch \endgroup
1701       \expanded{\showtokens{index \@tempc\space
1702         has charm info: \@tempa}}}%
1703     \fi
1704   \else
1705     \M@BadIntegerError\CharmInfo{\detokenize{#1}}%
1706   \fi}

```

Now for \CharmType.

```

\CharmType 1707 \protected\def\CharmType#1{%
1708   \M@check@int{#1}%
1709   \ifM@arg@good
1710     \begingroup
1711     \edef\@tempa{\number#1}%
1712     \expandafter\endgroup
1713     \expanded{\showtokens{index \@tempa\space has type
1714       \charmtype\numexpr\@tempa\relax}}}%
1715   \else
1716     \M@BadIntegerError\CharmType{\detokenize{#1}}%
1717   \fi}

```

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. We use `mathfont.encoding_slot` in `new_type_e` to keep track of the encoding slots where we will artificially add large versions of type `e` characters during loading.

```

1718 \directlua{
1719 mathfont = {}

```

```

1720 mathfont.extra_chars    = 0xFA000
1721 mathfont.encoding_slot = 0xFA010
1722 mathfont.fakel = mathfont.extra_chars + 4
1723 mathfont.faker = mathfont.extra_chars + 5
1724 mathfont.fakell = mathfont.extra_chars + 6
1725 mathfont.fakerr = mathfont.extra_chars + 7

```

Helper function for use in `\charminfo@`.

```

1726 function mathfont.strint(i)
1727   if i == (i // 1) then
1728     return string.format("@percentchar d", i)
1729   else
1730     return tostring(i)
1731   end
1732 end

```

Each character whose metrics we want to change will have one of two types: **a** for alphabet or **e** for extensible. We begin with type **a**. The `index` is the base-10 Unicode encoding value of the character that we will later modify. The `data` arguments are tables with 4 entries that store sizing information and information regarding accent placement. We divide the information by 1000 as is standard in \TeX .

```

1733 function mathfont:new_type_a(index, data_rm, data_it)
1734   self[index] = {}
1735   self[index].type = "a"
1736   self[index].data_rm = {}
1737   self[index].data_it = {}
1738   for i = 1, 4, 1 do
1739     self[index].data_rm[i] = data_rm[i] / 1000
1740     self[index].data_it[i] = data_it[i] / 1000
1741   end
1742 end

```

Initializing type **e** characters is more complicated. The `index` and `data` arguments are the same as in the type **a** case, and we process them similarly. The entries in `mathfont` for type **e** characters contain additional information. The `smash` value is a Unicode slot where we store a smashed version of the glyph with no height, depth, or width, which we need to scale the glyph correctly. The `num_variants` attribute is the number of slots in `next`, which we shorten to `v` for notational convenience.

```

1743 function mathfont:new_type_e(index, data_rm, data_it)
1744   self[index] = {}
1745   self[index].type = "e"

```

```

1746 self[index].smash = self.encoding_slot
1747 local v = (\string# data_rm - 3) / 2
1748 self[index].num_variants = v

```

Check that the user hasn't exhausted the Unicode table. (Unlikely, but you never know.)

```

1749 if self.encoding_slot + v + 1 > 0x10FFFF then
1750     error(\M@bound@assert)
1751 end

```

Now make lists that store encoding slots and scale factors. We have the following lists:

- `next`: list of encoding slots
- `data_rm`: scale factors for upright font shapes
- `data_it`: scale factors for italic font shapes

Start with making blank lists in the subtable in `mathfont`.

```

1752 self[index].next = {}
1753 self[index].data_rm = {}
1754 self[index].data_it = {}

```

We assemble these lists in a single loop, and they all have `v` elements. For `next`, we append consecutive integers to the list. For the scale factors, we expect `data_rm` and `data_it` to have $2v + 3$ entries, which we consider in pairs. The i th pair (i.e. entries $2i - 1$ and $2i$ of `data_rm` or `data_it`) encodes the horizontal and vertical scale factors for the i th large variant, and we add those scale factors as two-element sublists to the new lists on the i th iteration.

```

1755 for i = 1, v, 1 do
1756     self[index].next[i] = self.encoding_slot + i
1757     self[index].data_rm[i] = {data_rm[2*i-1] / 1000,
1758                             data_rm[2*i] / 1000}
1759     self[index].data_it[i] = {data_it[2*i-1] / 1000,
1760                             data_it[2*i] / 1000}
1761 end

```

The final entries of `data_rm` and `data_it` contain information about accent placement and italic correction. We add those values to the subtable as well.

```

1762 for i = 1, 3, 1 do
1763     self[index].data_rm[2*v+i] = data_rm[2*v+i] / 1000
1764     self[index].data_it[2*v+i] = data_it[2*v+i] / 1000
1765 end

```

Finally, update the `encoding_slot`.

```

1766 self.encoding_slot = self.encoding_slot + v + 1
1767 end

```

Interim processing. We let the user edit resizing and accent information for the characters in `mathfont`. The main editing function is `mathfont:add_to_charm`, which incorporates the user's information into the tables already in `mathfont`. It expects a single string of integers, floats, or asterisks separated by spaces or commas, and it immediately passes the argument to `parse_charm`, which processes it into tables that we incorporate into subtables of `mathfont`. We begin with a helper function to parse a (numeric) string. If this function returns a number, the number is properly scaled (divided by 1000).

```

1768 function mathfont.parse_num(s)
1769   if s == "*" then
1770     return s
1771   else
1772     local temp = tonumber(s)
1773     if temp then
1774       return temp / 1000
1775     else
1776       error("\M@number@ssert")
1777     end
1778   end
1779 end

```

Now `parse_charm`. We begin by setting up tables to store the parsed string contents. We store the Unicode index value in `index`.

```

1780 function mathfont.parse_charm(charm_input)
1781   local index = 0
1782   local charm_string = charm_input
1783   local temp_string = ""

```

Some preprocessing before we parse the string. Specifically, we

1. Get rid of any duplicate spaces
2. Remove any leading or trailing space, if present
3. Remove any spaces around slashes or commas
4. Replace any remaining spaces with commas

After completion of the replacements, we should have a new string with same numeric/override information as the original `charm_input` except without any spaces and all (pairs of) entries separated by commas. Step 1: duplicate spaces.

```

1784   while string.find(charm_string, "\space\space") do
1785     charm_string = string.gsub(charm_string, "\space\space", " ")
1786   end

```

Step 2: leading/trailing spaces.

```

1787   if string.sub(charm_string, 1, 1) == " " then

```

```

1788     charm_string = string.sub(charm_string, 2)
1789 end
1790 if string.sub(charm_string, -1) == " " then
1791     charm_string = string.sub(charm_string, 1, -2)
1792 end

```

Step 3: space around punctuation. We don't replace !_□ or ?_□ because the space in those cases could be separating two charm entries.

```

1793 charm_string = string.gsub(charm_string, " ", ",")
1794 charm_string = string.gsub(charm_string, ", ", ",")
1795 charm_string = string.gsub(charm_string, "/ ", "/")
1796 charm_string = string.gsub(charm_string, " /", "/")

```

Step 4: add commas.

```

1797 charm_string = string.gsub(charm_string, " ", ",")

```

Check that `charm_string` is not empty.

```

1798 if charm_string == "" then
1799     error(\M@empty@assert)
1800 end

```

Check that `charm_string` contains at least one comma. If it does not, we raise an error.

```

1801 if not string.find(charm_string, ",") then
1802     temp_string = charm_input
1803     error(\M@missing@assert)
1804 end

```

We're ready to parse the entries. We remove the first entry manually since it is the index and has different formatting possibilities from the other entries.

```

1805 local sep = string.find(charm_string, ",")
1806 temp_string = string.sub(charm_string, 1, sep-1)

```

Now check that the index is a (valid) number. Handle the case of possible asterisk at the end of `temp_string`.

```

1807 local exc
1808 if string.sub(temp_string, -1, -1) == "!" then
1809     index = tonumber(string.sub(temp_string, 1, -2))
1810     exc = 1
1811 elseif string.sub(temp_string, -1, -1) == "?" then
1812     index = tonumber(string.sub(temp_string, 1, -2))
1813     exc = 2
1814 else
1815     index = tonumber(temp_string)
1816     exc = 0
1817 end

```

```

1818 if index then
1819     assert(index == index // 1 and
1820            index >= 0           and
1821            index <= 1114111,    \M@index@ssert)
1822 else
1823     error(\M@index@ssert)
1824 end
1825 charm_string = string.sub(charm_string, sep+1)

```

Create the lists that we will use to store the information from `charm_string`.

```

1826 local split_string_rm = {}
1827 local split_string_it = {}

```

We loop through `charm_string` as long as it contains characters. At each iteration, we store the location of the first comma in `sep`. We remove the portion of `charm_string` preceding the first comma and store it in `temp_string`, and we save the remaining portion of `charm_string` for processing on the next iteration of the loop. We use `i` as a dummy variable to track loop iterations.

```

1828 local i = 1
1829 while charm_string do
1830     sep = string.find(charm_string, ",")
1831     if sep then
1832         temp_string = string.sub(charm_string, 1, sep-1)
1833         charm_string = string.sub(charm_string, sep+1)

```

If the current value of `charm_string` does not contain a comma, then it must be the last portion, and we set `charm_string` to `nil`.

```

1834     else
1835         temp_string = charm_string
1836         charm_string = nil
1837     end

```

First check whether `temp_string` contains a `/` character. If yes, we have two values to process, and if not, we have one.

```

1838     sep = string.find(temp_string, "/")
1839     if sep then

```

The information for upright font shapes comes from the first portion of `temp_string`.

```

1840         split_string_rm[i] = mathfont.parse_num(
1841             string.sub(temp_string, 1, sep-1))

```

Information for italic shapes comes from the latter portion of `temp_string`.

```

1842         split_string_it[i] = mathfont.parse_num(
1843             string.sub(temp_string, sep+1))

```

For the case without a /, the same information goes in both sets of lists.

```

1844     else
1845         local temp = mathfont.parse_num(temp_string)
1846         split_string_rm[i] = temp
1847         split_string_it[i] = temp
1848     end

```

Increment `i`, end the loop, and return the charm information.

```

1849     i = i + 1
1850 end
1851 return {index, exc, split_string_rm, split_string_it}
1852 end

```

We feed the user's charm information directly to `mathfont:add_to_charm`, which processes the information and stores it in `mathfont`. It first calls `parse_charm` to parse the input and then modifies `mathfont` accordingly. After being parsed, the user's input lives in `charm_rm` and `charm_it`. The `index` is the Unicode value of the character whose information we want to modify, and the `number_of_entries` is the length of `charm_metrics`.

```

1853 function mathfont:add_to_charm(charm_string)
1854     local temp = self.parse_charm(charm_string)
1855     local index      = temp[1]
1856     local force_type = temp[2]
1857     local data_rm    = temp[3]
1858     local data_it    = temp[4]
1859     local number_of_entries = \string# data_rm

```

If `mathfont` does not already have an entry for the Unicode character `index`, we create an entry with type `a` or `e` depending on the value of `force_type`.

```

1860     if not self[index] then
1861         mathfont.info(string.format("Setting up charm entries for
1862             U+\@percentchar X \@percentchar s (index \@percentchar d)",
1863             index, utf8.char(index), index))
1864         if force_type == 1 then
1865             temp = {}
1866             for i = 1, 30, 1 do
1867                 temp[i] = 1000
1868             end
1869             for i = 31, 33, 1 do
1870                 temp[i] = 0
1871             end
1872             self:new_type_e(index, temp, temp)
1873         else
1874             self:new_type_a(index, {0, 0, 0, 0}, {0, 0, 0, 0})

```

```

1875     end
1876 else

```

If `mathfont` does already have an entry for the character and `force_type` is positive, check whether the type of the entry in `mathfont` matches the value of `force_type`. If not, change the type by resetting the entry in `mathfont`. First, the case where the character has type `a`, and we want to change it to type `e`. We save the top and bottom accent values to use in setting up the new charm information.

```

1877     if (self[index].type == "a") and (force_type == 1) then
1878         local temp_rm = {}
1879         local temp_it = {}
1880         for i = 1, 30, 1 do
1881             temp_rm[i] = 1000
1882             temp_it[i] = 1000
1883         end
1884         temp_rm[31] = self[index].data_rm[3]
1885         temp_rm[32] = self[index].data_rm[4]
1886         temp_rm[33] = 0
1887         temp_it[31] = self[index].data_it[3]
1888         temp_it[32] = self[index].data_it[4]
1889         temp_it[33] = 0
1890         self:new_type_e(index, temp_rm, temp_it)

```

The process to convert to type `a` is simpler but fairly similar.

```

1891     elseif (self[index].type == "e") and (force_type == 2) then
1892         self:new_type_a(index, {0, 0,
1893                                 self[index].data_rm[31],
1894                                 self[index].data_rm[32]},
1895                                {0, 0,
1896                                 self[index].data_it[31],
1897                                 self[index].data_it[32]})
1898     end
1899 end

```

Handling the user's input depends on the type of entry `index`. The basic procedure is to first check the number of inputs, and if the user provided enough entries, we update each entries in the `mathfont` subtable. For every asterisk, we leave the corresponding subtable entries unaltered. For type `a`, we need four entries besides the `index`. The first two determine the left and right offset, and the last two determine accent placement.

```

1900 if self[index].type == "a" then
1901     local entries_needed = 4

```


Check number of entries. If it is too small, we issue an error, and if it is too large, we print a warning.

```

1902     if number_of_entries < entries_needed then
1903         error(\M@entries@assert)
1904     elseif number_of_entries > entries_needed then
1905         texio.write_nl(\M@entries@warning)
1906     end

```

Now update the table entries. The data outputs from `parse_charm` have been properly scaled (divided by 1000), so we don't have to worry about rescaling in this function.

```

1907     for i = 1, 4, 1 do
1908         if data_rm[i] \noexpand~="*" then
1909             self[index].data_rm[i] = data_rm[i]
1910         end
1911         if data_it[i] \noexpand~="*" then
1912             self[index].data_it[i] = data_it[i]
1913         end
1914     end

```

Now do type e. The number of entries in the `data` lists must be at least $2 * \text{tot_variants} + 3$. 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.

```

1915     elseif self[index].type == "e" then
1916         local tot_variants = self[index].num_variants
1917         local entries_needed = 2 * tot_variants + 3

```

Again check number of entries.

```

1918     if number_of_entries < entries_needed then
1919         error(\M@entries@assert)
1920     elseif number_of_entries > entries_needed then
1921         texio.write_nl(\M@entries@warning)
1922     end

```

Now store the charm information. Again, we scaled the `data_rm` and `data_it` values in `parse_charm`, so we don't have to divide by 1000 here.

```

1923     for i = 1, tot_variants, 1 do
1924         if data_rm[2*i-1] \noexpand~="*" then
1925             self[index].data_rm[i][1] = data_rm[2*i-1]
1926         end
1927         if data_rm[2*i] \noexpand~="*" then
1928             self[index].data_rm[i][2] = data_rm[2*i]
1929         end

```

```

1930     if data_it[2*i-1] \noexpand~="*" then
1931         self[index].data_it[i][1] = data_it[2*i-1]
1932     end
1933     if data_it[2*i] \noexpand~="*" then
1934         self[index].data_it[i][2] = data_it[2*i]
1935     end
1936 end

```

The final entries for type `e` are the accent and italic correction information.

```

1937     for i = tot_variants + 1, tot_variants + 3, 1 do
1938         if data_rm[i] \noexpand~="*" then
1939             self[index].data_rm[i] = data_rm[i]
1940         end
1941         if data_it[i] \noexpand~="*" then
1942             self[index].data_it[i] = data_it[i]
1943         end
1944     end
1945 end
1946 end

```

We end this section with three general-purpose Lua functions. The first function, `utf_16BE`, accepts a nonnegative integer and returns its representation in UTF-16 big-endian format. Let x be a nonnegative integer at most $0x10FFFF$. Here are the steps to convert x to its big-endian representation:

1. If $x \leq 0xFFFF$, keep x as is. Its representation is a single four-digit hexadecimal number.
2. If $x \geq 0x10000$, we represent x as two four-digit hexadecimal numbers. First, subtract $0x10000$ from x , and the result y is a number between 0 and $0xFFFFF$.
3. Equivalently, we can think of y as a twenty-digit binary number. (Five hexadecimal digits store the same information as twenty binary digits.)
4. Take the 10 left-most digits of y (integer divide by $2^{10} = 1024$), and add them to $0xD800$. The result z_1 is the first hexadecimal number.
5. Take the 10 right-most digits of y (remainder after dividing by $2^{10} = 1024$), and add them to $0xDC00$. The result z_2 is the second hexadecimal number.
6. The big-endian representation of x is the string z_1z_2 .

The purpose of big-endian representation is to encode Unicode characters beyond $U+FFFF$ while still using four-digit hexadecimal numbers.

```

1947 function mathfont.utf_16BE(integer)
1948     if integer > 0x10FFFF then

```

```

1949     error(\M@bound@assert)
1950 end
1951 local temp = string.format("\@percentchar X", integer)
1952 if \string# temp <= 4 then
1953     while \string# temp < 4 do
1954         temp = "0" .. temp
1955     end
1956 else
1957     temp = integer - 0x10000
1958     local left_bits = 0xD800 + temp // 1024
1959     local right_bits = 0xDC00 + temp \@percentchar 1024
1960     temp = string.format("\@percentchar X\@percentchar X",
1961         left_bits, right_bits)
1962 end
1963 return temp
1964 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.

```

1965 function mathfont.glyph_info(char)
1966     local glyph_width = char.width or 0
1967     local glyph_height = char.height or 0
1968     local glyph_depth = char.depth or 0
1969     local glyph_italic = char.italic or 0
1970     return glyph_width, glyph_height, glyph_depth, glyph_italic
1971 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 we typeset the glyph virtually using a `char` command.

```

1972 function mathfont.smash_glyph(index)
1973     local smash_table = {}
1974     smash_table.width = 0
1975     smash_table.height = 0
1976     smash_table.depth = 0
1977     smash_table.commands = {"char", index}
1978     return smash_table
1979 end

```

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

```

1980 function mathfont.empty(arg)
1981 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 seven 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 create new character subtables for inclusion in a font object, and we think of these new subtables as modified versions of characters already present in a given font. The functions for assembling character tables take three arguments. The `index` argument is the Unicode index of the modified character. The `charm_data` argument is the subtable in `mathfont` of charm information that corresponds to `index`, and the `fontdata` argument is a font object. For type `a`, we change the width of the bounding box and horizontal glyph positioning, and for type `e`, we scale the glyph to create large variants and change the italic correction. For both types, we modify accent placement. We add five categories of information into our new character tables: glyph dimensions, Unicode encoding bits, (possible) virtual font commands, accent placement dimensions, and math kerning.

The `:make_a_table` returns a character table for type `a` characters. We

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
<code>"mathfont.finishing_touchees"</code>	None

build up the subtable in the variable `a_table`, and we eventually return `a_table` at the end of the function. We let `char` be a shorthand for the subtable at `index` in `fontdata`, and `slant` is the font's `slant` parameter. In upright fonts, `slant` is generally 0, and in italic fonts, `slant` is generally positive.

```
1982 function mathfont:make_a_table(index, charm_data, fontdata)
1983   local a_table = {}
1984   local char = fontdata.characters[index] or {}
1985   local ex = fontdata.parameters.x_height or 0
1986   local slant = (fontdata.parameters.slant or 0)/ 65536
```

Get the dimensions of the character. We determine the new bounding box dimensions, horizontal glyph placement, and accent placement in terms of the character's original width (plus italic correction).

```
1987   local width, height, depth, italic = self.glyph_info(char)
```

Incorporate the italic correction into the character width.

```
1988   width = width + italic
```

We extract charm information from `charm_data` depending on whether the font is slanted or not.

```
1989   local left_stretch
1990   local right_stretch
1991   local top_accent
1992   local bot_accent
1993   if slant == 0 then
1994     left_stretch = charm_data.data_rm[1]
1995     right_stretch = charm_data.data_rm[2]
1996     top_accent = charm_data.data_rm[3]
1997     bot_accent = charm_data.data_rm[4]
1998   else
1999     left_stretch = charm_data.data_it[1]
2000     right_stretch = charm_data.data_it[2]
2001     top_accent = charm_data.data_it[3]
2002     bot_accent = charm_data.data_it[4]
2003   end
```

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

```
2004   local offset = width * left_stretch
2005   a_table.width = width * (1 + left_stretch + right_stretch)
2006   a_table.height = height
2007   a_table.depth = depth
```

```
2008  a_table.italic = 0
```

Unicode information attached to the character.

```
2009  a_table.unicode = index
```

```
2010  a_table.tounicode = self.utf_16BE(index)
```

```
2011  a_table.index = char.index
```

If `left_stretch` is nonzero, we have to turn the character into a virtual character that typesets the glyph through a `char` command—that is the only way to add space on the left of the glyph. (A nonzero `right_stretch` is easier because we only have to extend the bounding box.)

```
2012  if offset \noexpand~= 0 then
```

```
2013    if char.commands then
```

```
2014        error(\M@virtual@assert)
```

```
2015    else
```

```
2016        a_table.commands = {"right", offset}, {"char", index}}
```

```
2017    end
```

```
2018  end
```

We calculate accent placement in two steps. The first step is to calculate a “base” accent position from character and font properties, and then we modify the base position according to charm information. If the font contains a `top_accent` value for the character, we take that value as our base accent position.

```
2019  local top_base
```

```
2020  if char.top_accent then
```

```
2021    top_base = char.top_accent
```

If the font does not contain a `top_accent` value, which for text fonts is the more likely possibility, we have to create the base accent position ourselves. We use an approach similar to how \TeX positions accents in text mode. If the character is less than 1ex tall, the base accent position is halfway across the character’s bounding box, i.e. any horizontal offset plus half the original width and half the italic correction. If the character is taller than 1ex, we move the base accent position right by the font’s `slant` value times the character’s height above 1ex. The base accent position in this approach does not change when the user changes the `left_stretch` or `right_stretch` values in the charm information.

```
2022  else
```

```
2023    top_base = offset + 0.5 * width
```

```
2024    if height > ex then
```

```
2025        top_base = top_base + slant * (height - ex)
```

```
2026    end
```

```
2027  end
```

We take a similar approach for the bottom accent. For the base accent position, we subtract the `slant` times `lex`.

```

2028  local bot_base
2029  if char.bot_accent then
2030      bot_base = char.bot_accent
2031  else
2032      bot_base = offset + 0.5 * width - slant * ex
2033  end

```

Now add the accent information to the font.

```

2034  a_table.top_accent = top_base + top_accent * width
2035  a_table.bot_accent = bot_base + bot_accent * width

```

TEX shifts superscripts (but not subscripts) right by the italic correction. We added italic correction to the character's width. Effectively, that shifts any superscript right relative to the character's original bounding box, so our changes to the character result in superscripts that behave the way we expect. However, the larger bounding box also affects subscripts, which we don't want, so we implement a mathkern table that moves any subscripts left by the italic correction. 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 lower right corner.

```

2036  a_table.mathkern = {}
2037  a_table.mathkern.top_right = {{height = 0, kern = 0}}
2038  a_table.mathkern.bottom_right = {{height = 0, kern = -italic}}
2039  a_table.mathkern.top_left = {{height = 0, kern = 0}}
2040  a_table.mathkern.bottom_left = {{height = 0, kern = 0}}
2041  return a_table
2042 end

```

For type e characters, we add several virtual characters to the font, and we use `make_e_commands` to produce their commands tables. The commands tables from this function produces a scaled version of the glyph in slot `index`. The `pdf` command sends code directly to the pdf backend that handles the transformation. The `q` specification induces a linear transformation of the output, which in this case is a dilation by `h_stretch` and `v_stretch` factors. The `Q` command restores the original coordinate system.

```

2043 function mathfont.make_e_commands(index, h_stretch, v_stretch)
2044  local c_1 = {"pdf", "origin",
2045      string.format("q \@percentchar s 0 0 \@percentchar s 0 0 cm",
2046          h_stretch, v_stretch)}
2047  local c_2 = {"char", index}

```

```

2048 local c_3 = {"pdf", "origin", "Q"}
2049 return {c_1, c_2, c_3}
2050 end

```

The function for type e characters returns a list of character subtables because we need to create multiple characters at once. Specifically, the function returns a new subtable for the original character plus one subtable for each larger variant. The structure is similar to `:make_type_a`, except that we scale the glyph instead of enlarging the bounding box.

```

2051 function mathfont:make_e_table(index, charm_data, fontdata)
2052 local e_table = {}
2053 local char = fontdata.characters[index] or {}
2054 local ex = fontdata.parameters.x_height or 0
2055 local slant = (fontdata.parameters.slant or 0) / 65536
2056 local width, height, depth, italic = self.glyph_info(char)

```

Extract the charm information other than scale factors. Here `_sc` is short for scale.

```

2057 local v = charm_data.num_variants
2058 local smash = charm_data.smash
2059 local next = charm_data.next
2060 local top_accent_sc
2061 local bot_accent_sc
2062 local italic_sc
2063 if slant == 0 then
2064     top_accent_sc = charm_data.data_rm[2*v+1]
2065     bot_accent_sc = charm_data.data_rm[2*v+2]
2066     italic_sc = charm_data.data_rm[2*v+3]
2067 else
2068     top_accent_sc = charm_data.data_it[2*v+1]
2069     bot_accent_sc = charm_data.data_it[2*v+2]
2070     italic_sc = charm_data.data_it[2*v+3]
2071 end

```

Calculate accent placement for the original glyph, and rescale the italic correction. We calculate accent placement similarly to type a, and we change the italic correction afterwards to ensure that accent placement is independent of `italic_scale`. Unlike with type a, we do not enlarge the bounding box on the left side of the character, so the base accent placement does not contain `offset`.

```

2072 local top_base
2073 if char.top_accent then
2074     top_base = char.top_accent

```



```

2075 else
2076     top_base = 0.5 * width
2077     if height > ex then
2078         top_base = top_base + slant * (height - ex)
2079     end
2080 end
2081 local bot_base = char.bot_accent or (0.5 * width - slant * ex)
2082 local top_accent = top_base + top_accent_sc * (width + italic)
2083 local bot_accent = bot_base + bot_accent_sc * (width + italic)
2084 italic = italic + italic_sc * (width + italic)

```

Store the Unicode encoding slot for reference later

```

2085 local tounicode = self.utf_16BE(index)

```

We create a number of entries in `e_table` equal to one plus the number of variants we want, which is stored in `charm_data.num_variants`. We begin with the first entry. This isn't a full character subtable because for the small version of the big operator, we won't replace the subtable already in `fontdata` but rather will add the information here into that subtable.

```

2086 e_table[1] = {}
2087 e_table[1].italic = italic
2088 e_table[1].unicode = index
2089 e_table[1].tounicode = self.utf_16BE(index)
2090 e_table[1].top_accent = top_accent
2091 e_table[1].bot_accent = bot_accent
2092 e_table[1].next = next[1]

```

Now loop through the large variants, and add each one to `e_table`. We begin by extracting the scale factors for the particular large variant.

```

2093 local h_stretch
2094 local v_stretch
2095 for i = 2, v + 1, 1 do
2096     if slant == 0 then
2097         h_stretch = charm_data.data_rm[i-1][1]
2098         v_stretch = charm_data.data_rm[i-1][2]
2099     else
2100         h_stretch = charm_data.data_it[i-1][1]
2101         v_stretch = charm_data.data_it[i-1][2]
2102     end

```

Now add the entries for the subtable.

```

2103 e_table[i] = {}
2104 e_table[i].width = width * h_stretch
2105 e_table[i].height = height * v_stretch

```

```

2106     e_table[i].depth = depth * v_stretch
2107     e_table[i].italic = italic * h_stretch

```

Add the Unicode information.

```

2108     e_table[i].unicode = index
2109     e_table[i].tounicode = tounicode

```

Accent placement.

```

2110     e_table[i].top_accent = top_accent * h_stretch
2111     e_table[i].bot_accent = bot_accent * h_stretch

```

Add the commands.

```

2112     e_table[i].commands =
2113         self.make_e_commands(smash, h_stretch, v_stretch)

```

If we aren't dealing with the last entry in the table, we need to add the character's `next` field. 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`.

```

2114     if i <= v then
2115         e_table[i].next = charm_data.next[i]
2116     end
2117 end
2118 return e_table
2119 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 idea is to transform a guillemet such that the top 90% of the original bounding box lines up with the full bounding box (height plus depth) of the left parenthesis (U+28). We make the fake angle brackets as follows:

1. Take a (single or double) guillemet, and let h be its height. We assume the guillemet has no depth.
2. Let h_p and d_p be the height and depth respectively of the left parenthesis.
3. Our transformation of the guillemet will involve a translation and a dilation. The translation should lower the basepoint of the guillemet such that if p is the proportion of the guillemet that is below the baseline after translation, then $(p - 0.1\sigma)h = d_p$, where σ is the scale factor, assuming we apply the translation before the dilation. (The dilation fixes the current position within the virtual character regardless of its relation to the baseline.) This means $ph = d_p + 0.1h\sigma$.
4. The scale factor for the dilation is

$$\sigma = \frac{h_p + d_p}{0.9h},$$

so 90% of the guillemet height becomes equal to the size of the parenthesis.

The function accepts the index of a guillemet as `index` and the index of the smashed guillemet as `smash`. The `slot` argument is the encoding slot of the angle bracket. First, extract relevant character and font dimensions.

```

2120 function mathfont:make_fake_angle(index, smash, slot, fontdata)
2121   local fab_table = {}           % fab for fake angle bracket
2122   local chars = fontdata.characters
2123   local lp = chars[40] or {}     % lp for left parenthesis
2124   local lp_width, lp_height, lp_depth, lp_italic =
2125     self.glyph_info(lp)
2126   local g = chars[index] or {}   % g for guillemet
2127   local g_width, g_height, g_depth, g_italic = self.glyph_info(g)
2128   local ex = fontdata.parameters.x_height or 0
2129   local slant = (fontdata.parameters.slant or 0) / 65536

```

We compute the dimensions of our fake angle bracket.

```

2130   local factor
2131   if g_height \noexpand~= 0 then
2132     factor = (lp_height + lp_depth) / (0.9 * g_height)
2133   else
2134     factor = 1
2135   end
2136   local shift = lp_depth + 0.1 * factor * g_height

```

Now populate `fab_table`.

```

2137   fab_table.height = lp_height
2138   fab_table.depth = lp_depth
2139   fab_table.width = g_width
2140   fab_table.italic = g_italic

```

Unicode information.

```

2141   fab_table.unicode = slot
2142   fab_table.tounicode = self.utf_16BE(slot)

```

We calculate accent placement information the same way as we have been doing for type a and type e characters.

```

2143   if g.top_accent then
2144     fab_table.top_accent = g.top_accent
2145   else
2146     if g_height > ex then
2147       fab_table.top_accent = 0.5 * g_width +
2148                             slant * (g_height - ex)
2149     else

```

```

2150     fab_table.top_accent = 0.5 * g_width
2151     end
2152 end
2153 fab_table.bot_accent = g.bot_accent or
2154     (0.5 * g_width - slant * ex)

```

Commands.

```

2155 fab_table.commands = {
2156     {"down", shift},
2157     {"pdf", "origin",
2158         string.format("q 1 0 0 \@percentchar s 0 0 cm", factor)},
2159     {"char", smash},
2160     {"pdf", "origin", "Q"},
2161     {"down", -shift}}
2162 return fab_table
2163 end

```

Similar function that returns the character subtable for a nabla (inverted increment symbol/capital Delta). Again, `index` is the encoding slot for an increment symbol, and `smash` is the encoding slot for a smashed version of an increment.

```

2164 function mathfont:make_nabla(index, smash, slot, fontdata)
2165     local n_table = {}           % n for nabla
2166     local i = fontdata.characters[index] % i for increment
2167     local i_width, i_height, i_depth, i_italic = self.glyph_info(i)

```

Now populate `n_table`.

```

2168     n_table.width = i_width
2169     n_table.height = i_height
2170     n_table.depth = i_depth
2171     n_table.italic = i_italic

```

Unicode information.

```

2172     n_table.unicode = slot
2173     n_table.tounicode = self.utf_16BE(slot)

```

Take accent placement values from the increment symbol if they exist.

```

2174     if i.top_accent then
2175         n_table.bot_accent = i_width - i.top_accent
2176     end
2177     if i.bot_accent then
2178         n_table.top_accent = i_width - i.bot_accent
2179     end

```

Commands. We reflect the increment glyph horizontally and vertically.

```

2180     n_table.commands = {

```

```

2181     {"down", -i_height},
2182     {"right", i_width},
2183     {"pdf", "origin", "q -1 0 0 -1 0 0 cm"},
2184     {"char", smash},
2185     {"pdf", "origin", "Q"},
2186     {"right", -i_width},
2187     {"down", i_height}}
2188   return n_table
2189 end

```

We come to the 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.

```

2190 function mathfont.set_nomath_false(fontdata)
2191   fontdata.nomath = false
2192   fontdata.oldmath = false
2193 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.

```

2194 function mathfont.apply_charm_info(fontdata)
2195   local chars = fontdata.characters or {}

```

Before we loop through the charm data, we need to make a few changes to the font, namely add fake angle brackets, add nabla, and trim the bounding box on the surd. First, we add fake angle brackets. We use `mathfont.extra_chars` to track where we put the extra (virtual) characters in the font.

```

2196   local temp = mathfont.extra_chars
2197   chars[temp] = mathfont.smash_glyph(0x2039) % smashed \lguil
2198   chars[temp+1] = mathfont.smash_glyph(0x203A) % smashed \rguil
2199   chars[temp+2] = mathfont.smash_glyph(0xAB) % smashed \llguil
2200   chars[temp+3] = mathfont.smash_glyph(0xBB) % smashed \rrguil

```

Now add the characters to the font.

```

2201   chars[mathfont.fakel] = mathfont:make_fake_angle( %\fakelangle
2202     0x2039, temp, 0x27E8, fontdata)
2203   chars[mathfont.faker] = mathfont:make_fake_angle( %\fakerangle
2204     0x203A, temp+1, 0x27E9, fontdata)
2205   chars[mathfont.fakell] = mathfont:make_fake_angle( %\fakellangle

```

```

2206     0xAB, temp+2, 0x27EA, fontdata)
2207 chars[mathfont.fakerr] = mathfont:make_fake_angle( %\fakerrangle
2208     0xBB, temp+3, 0x27EB, fontdata)

```

If the function doesn't have an increment character, copy a capital Delta if it exists in the font.

```

2209 if chars[0x394] and (not chars[0x2206]) then
2210     chars[0x2206] = {}
2211     for k,v in pairs(chars[0x394]) do
2212         chars[0x2206][k] = v
2213     end
2214     chars[0x2206].commands = {"char", 0x394}}
2215 end

```

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

```

2216 if chars[0x2206] and (not chars[0x2207]) then
2217     chars[temp+8] = mathfont.smash_glyph(0x2206)
2218     chars[0x2207] = mathfont:make_nabla(
2219         0x2206, temp+8, 0x2207, fontdata)
2220 end

```

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 `\vsurdfactor` and `\hsurdfactor`.

```

2221 if chars[0x221A] then
2222     local hsurd = tex.getcount("hsurdfactor") / 1000
2223     local vsurd = tex.getcount("vsurdfactor") / 1000
2224     chars[0x221A].width = hsurd * chars[0x221A].width
2225     chars[0x221A].height = vsurd * chars[0x221A].height
2226 end

```

Perform the loop. We care about entries `info` whose index is a number because those entries are the charm information that determines how we modify the font object. We ignore charm information for any characters not present in the font.

```

2227 for index, info in pairs(mathfont) do
2228     if type(index) == "number" and chars[index] then

```

One each iteration of the loop, we start by checking the `type` of the current entry because the handling of the font object varies based on the character type. For characters of type `a`, we insert our character subtable into the font object.

```

2229         if info.type == "a" then
2230             chars[index] = mathfont:make_a_table(index,info,fontdata)

```

For type `e` we have to add entries to `chars[index]` and insert multiple character subtables into the font, namely one for the smashed version of the base glyph and others corresponding to the large variants.

```
2231     elseif info.type == "e" then
2232         local variants_table =
2233             mathfont:make_e_table(index, info, fontdata)
```

First add entries to the subtable for the base glyph.

```
2234         for k,v in pairs(variants_table[1]) do
2235             chars[index][k] = v
2236         end
```

Smashed version of the glyph.

```
2237         chars[info.smash] = mathfont.smash_glyph(index)
```

Now add the large variants.

```
2238         for i = 1, info.num_variants, 1 do
2239             chars[info.next[i]] = variants_table[i+1]
2240         end
2241     end
2242 end
2243 end
2244 end
```

The `populate_math_constants` function is even longer because we need to add a full `MathConstants` table to the font object, which has some fifty parameters that we need to set. (But the mechanics behind the function are simpler than `apply_charm_info`.) 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. For brevity, we let `MC` be a shortcut for the `MathConstants` table.

```
2245 function mathfont.math_constants(fontdata)
2246     fontdata.MathConstants = fontdata.MathConstants or {}
2247     local MC = fontdata.MathConstants
```

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.

```
2248     local size = fontdata.size or 0
2249     local ex = fontdata.parameters.x_height or 0
2250     local em = fontdata.parameters.quad or 0
```

```

2251 local A_height = 0
2252 local y_depth = 0
2253 if fontdata.characters[65] then
2254     A_height = fontdata.characters[65].height or 0 % A
2255 end
2256 if fontdata.characters[121] then
2257     y_depth = fontdata.characters[121].depth or 0 % y
2258 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.

```

2259 local axis = 0
2260 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 `\rulethicknessfactor`.

```

2261 if MC.FractionRuleThickness then
2262     rule_thickness = MC.FractionRuleThickness
2263 else
2264     local temp =
2265         tex.getcount("rulethicknessfactor") / 1000
2266     rule_thickness = (size / 18) * temp
2267     MC.FractionRuleThickness = rule_thickness
2268 end

```

If the font has an `AxisHeight`, we take that value as the axis. If the font does not have `AxisHeight` already set, we set the axis to be the height of a minus sign, which has position U+2212 (8722 in decimal). As a fallback, we set the axis to 0.8ex if the font does not have a character in slot U+2212.

```

2269 if MC.AxisHeight then
2270     axis = MC.AxisHeight
2271 else
2272     if fontdata.characters[8722] then
2273         axis = fontdata.characters[8722].height - rule_thickness / 2
2274     else
2275         axis = 0.8 * ex
2276     end
2277     MC.AxisHeight = axis
2278 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.

```

2279 local xi_9 = 2 * rule_thickness      % top limit min clearance
2280 local xi_10 = xi_9 + 0.35 * y_depth % bottom limit try placement
2281 local xi_11 = xi_10                  % top limit min clearance
2282 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.

```

2283 local sigma_8 = axis + 1.5 * rule_thickness + y_depth +
2284     0.1 * A_height
2285 local sigma_9 = axis + 1.35 * rule_thickness + 0.7 * y_depth +
2286     0.07 * A_height
2287 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.

```

2288 local sigma_11 = -axis + 1.5 * rule_thickness +
2289     0.5 * y_depth + 1.1 * A_height
2290 local sigma_12 = -axis + 1.35 * rule_thickness +
2291     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.

```

2292 local sigma_13 = 0.6 * A_height    % attempted superscript height
2293 local sigma_15 = 0.5 * A_height    % attempt for cramped scripts
2294 local sigma_16 = 1.1 * y_depth     % attempted subscript lower
2295 if sigma_16 < 0.2 * A_height then
2296     sigma_16 = 0.2 * A_height
2297 end
2298 local sigma_17 = sigma_16           % sigma_16 when superscript
2299 local sigma_18 = 0.5 * A_height     % superscript lower for boxed
2300 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.

```

2301 if not MC.DisplayOperatorMinHeight then
2302     MC.DisplayOperatorMinHeight = 1.8 * A_height
2303 end
2304 if not MC.FractionDelimiterDisplayStyleSize then
2305     MC.FractionDelimiterDisplayStyleSize = 2 * size

```

```

2306 end
2307 if not MC.FractionDelimiterSize then
2308     MC.FractionDelimiterSize = 1.3 * size
2309 end
2310 if not MC.FractionDenominatorDisplayStyleShiftDown then
2311     MC.FractionDenominatorDisplayStyleShiftDown = sigma_11
2312 end
2313 if not MC.FractionDenominatorShiftDown then
2314     MC.FractionDenominatorShiftDown = sigma_12
2315 end

```

We set the minimum clearance for the numerator to be twice the rule height in `\displaystyle` and the rule height in other styles. Our approach in setting the attempted height of the numerator (σ_8 and σ_9) was to add the minimum clearance plus the descender depth plus a small extra space, so in general, we do not expect the numerator to run into the minimum clearance. For the denominator, we do the same thing except add half the descender depth to the clearance, which balances the amount of color above and below the fraction rule and is similar to what we did for the lower limits on big operators when we set ξ_{11} larger than ξ_9 .

```

2316 if not MC.FractionDenominatorDisplayStyleGapMin then
2317     MC.FractionDenominatorDisplayStyleGapMin =
2318         rule_thickness + 0.5 * y_depth
2319 end % that MathConstants entry has a long name lol
2320 if not MC.FractionDenominatorGapMin then
2321     MC.FractionDenominatorGapMin =
2322         rule_thickness + 0.35 * y_depth
2323 end
2324 if not MC.FractionNumeratorDisplayStyleShiftUp then
2325     MC.FractionNumeratorDisplayStyleShiftUp = sigma_8
2326 end
2327 if not MC.FractionNumeratorShiftUp then
2328     MC.FractionNumeratorShiftUp = sigma_9
2329 end
2330 if not MC.FractionNumeratorDisplayStyleGapMin then
2331     MC.FractionNumeratorDisplayStyleGapMin = rule_thickness
2332 end
2333 if not MC.FractionNumeratorGapMin then
2334     MC.FractionNumeratorGapMin = rule_thickness
2335 end

```

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

```

2336 if not MC.SkewedFractionHorizontalGap then
2337   MC.SkewedFractionHorizontalGap = 0.5 * em
2338 end
2339 if not MC.SkewedFractionVerticalGap then
2340   MC.SkewedFractionVerticalGap = ex
2341 end

```

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

```

2342 if not MC.UpperLimitBaselineRiseMin then
2343   MC.UpperLimitBaselineRiseMin = xi_11
2344 end
2345 if not MC.UpperLimitGapMin then
2346   MC.UpperLimitGapMin = xi_9
2347 end
2348 if not MC.LowerLimitBaselineDropMin then
2349   MC.LowerLimitBaselineDropMin = xi_12
2350 end
2351 if not MC.LowerLimitGapMin then
2352   MC.LowerLimitGapMin = xi_10
2353 end

```

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

```

2354 if not MC.StretchStackGapBelowMin then
2355   MC.StretchStackGapBelowMin = xi_10
2356 end
2357 if not MC.StretchStackTopShiftUp then
2358   MC.StretchStackTopShiftUp = xi_11
2359 end
2360 if not MC.StretchStackGapAboveMin then
2361   MC.StretchStackGapAboveMin = xi_9
2362 end
2363 if not MC.StretchStackBottomShiftDown then
2364   MC.StretchStackBottomShiftDown = xi_12
2365 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.

```

2366 if not MC.OverbarExtraAscender then
2367   MC.OverbarExtraAscender = rule_thickness
2368 end
2369 if not MC.OverbarRuleThickness then

```

```

2370     MC.OverbarRuleThickness = rule_thickness
2371 end
2372 if not MC.OverbarVerticalGap then
2373     MC.OverbarVerticalGap = 2 * rule_thickness
2374 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.

```

2375 if not MC.RadicalExtraAscender then
2376     MC.RadicalExtraAscender = rule_thickness
2377 end
2378 if not MC.RadicalRuleThickness then
2379     MC.RadicalRuleThickness = rule_thickness
2380 end
2381 if not MC.RadicalDisplayStyleVerticalGap then
2382     MC.RadicalDisplayStyleVerticalGap = 2 * rule_thickness
2383 end
2384 if not MC.RadicalVerticalGap then
2385     MC.RadicalVerticalGap = rule_thickness
2386 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.

```

2387 if not MC.RadicalKernBeforeDegree then
2388     MC.RadicalKernBeforeDegree = (5/18) * em
2389 end
2390 if not MC.RadicalKernAfterDegree then
2391     MC.RadicalKernAfterDegree = (10/18) * em
2392 end
2393 if not MC.RadicalDegreeBottomRaisePercent then
2394     MC.RadicalDegreeBottomRaisePercent = 60
2395 end

```

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

```

2396 if not MC.SpaceAfterScript then
2397     MC.SpaceAfterScript = 0.1 * em
2398 end

```

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

```

2399 if not MC.StackBottomDisplayStyleShiftDown then
2400     MC.StackBottomDisplayStyleShiftDown = sigma_11

```

```

2401 end
2402 if not MC.StackBottomShiftDown then
2403   MC.StackBottomShiftDown = sigma_12
2404 end
2405 if not MC.StackTopDisplayStyleShiftUp then
2406   MC.StackTopDisplayStyleShiftUp = sigma_8
2407 end
2408 if not MC.StackTopShiftUp then
2409   MC.StackTopShiftUp = sigma_10
2410 end

```

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

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

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

```

2411 if not MC.StackDisplayStyleGapMin then
2412   MC.StackDisplayStyleGapMin =
2413     5 * rule_thickness + 0.5 * y_depth
2414 end
2415 if not MC.StackGapMin then
2416   MC.StackGapMin = 3 * rule_thickness + 0.35 * y_depth
2417 end

```

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

```

2418 if not MC.SubscriptShiftDown then
2419   MC.SubscriptShiftDown = sigma_16
2420 end
2421 if not MC.SubscriptBaselineDropMin then
2422   MC.SubscriptBaselineDropMin = sigma_19
2423 end
2424 if not MC.SubscriptShiftDownWithSuperscript then
2425   MC.SubscriptShiftDownWithSuperscript = sigma_17
2426 end

```

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

```

2427 if not MC.SubscriptTopMax then

```

```

2428     MC.SubscriptTopMax = 0.5 * A_height
2429 end

```

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

```

2430 if not MC.SubSuperscriptGapMin then
2431     MC.SubSuperscriptGapMin = rule_thickness
2432 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.

```

2433 if not MC.SuperscriptBottomMin then
2434     MC.SuperscriptBottomMin = sigma_15 - 0.7 * y_depth
2435 end
2436 if not MC.SuperscriptBaselineDropMax then
2437     MC.SuperscriptBaselineDropMax = sigma_18
2438 end
2439 if not MC.SuperscriptShiftUp then
2440     MC.SuperscriptShiftUp = sigma_13
2441 end
2442 if not MC.SuperscriptShiftUpCramped then
2443     MC.SuperscriptShiftUpCramped = sigma_15
2444 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.

```

2445 if not MC.SuperscriptBottomMaxWithSubscript then
2446     MC.SuperscriptBottomMaxWithSubscript =
2447         -sigma_16 + 0.7 * A_height + rule_thickness
2448 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.

```

2449 if not MC.UnderbarExtraDescender then
2450     MC.UnderbarExtraDescender = rule_thickness

```

```

2451 end
2452 if not MC.UnderbarRuleThickness then
2453     MC.UnderbarRuleThickness = rule_thickness
2454 end
2455 if not MC.UnderbarVerticalGap then
2456     MC.UnderbarVerticalGap = 2 * rule_thickness
2457 end

```

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

```

2458 if not MC.MinConnectorOverlap then
2459     MC.MinConnectorOverlap = 0
2460 end
2461 end

```

Time for callbacks! We create seven of them.

```

2462 luatexbase.create_callback("mathfont.inspect_font",
2463     "simple", mathfont.empty)
2464 luatexbase.create_callback("mathfont.pre_adjust",
2465     "simple", mathfont.empty)
2466 luatexbase.create_callback("mathfont.disable_nomath",
2467     "simple", mathfont.set_nomath_false)
2468 luatexbase.create_callback("mathfont.add_math_constants",
2469     "simple", mathfont.math_constants)
2470 luatexbase.create_callback("mathfont.fix_character_metrics",
2471     "simple", mathfont.apply_charm_info)
2472 luatexbase.create_callback("mathfont.post_adjust",
2473     "simple", mathfont.empty)
2474 luatexbase.create_callback("mathfont.finishing_touches",
2475     "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.

```

2476 function mathfont.info(msg)
2477     texio.write_nl("log", "Package mathfont Info: " .. msg)
2478 end
2479 function mathfont.get_font_name(fontdata)
2480     return fontdata.psname or
2481         fontdata.fullname or
2482         fontdata.name or "<unknown font name>"
2483 end

```

The `adjust_font` function is what actually goes in `luaotfload.patch_font`. This function calls the six callbacks at appropriate times and writes informa-

tional messages in the log file. We adjust the font object when `nomath` is `false` and the font is loaded in `base` mode. I am assuming that the user will usually load text-only fonts in `node` or `harf` mode, and then `mathfont` does not need to (and probably should not) alter that particular font. Unfortunately, there does not appear to be a better way to notate that we will use a font for text versus math when declaring it with `\DeclareFontSize`. I would ideally set `script=math` with the rest of the OpenType font features, but `luaotfload` ignores `script` declarations that aren't built into the font.

```

2484 function mathfont.adjust_font(fontdata)
2485   luatexbase.call_callback("mathfont.inspect_font", fontdata)
2486   local the_font = mathfont.get_font_name(fontdata)
2487   if fontdata.nomath and
2488     fontdata.properties and
2489     fontdata.properties.mode and
2490     fontdata.properties.mode == "base" then
2491     luatexbase.call_callback("mathfont.pre_adjust",
2492       fontdata)
2493     luatexbase.call_callback("mathfont.disable_nomath",
2494       fontdata)
2495     luatexbase.call_callback("mathfont.add_math_constants",
2496       fontdata)
2497     luatexbase.call_callback("mathfont.fix_character_metrics",
2498       fontdata)
2499     luatexbase.call_callback("mathfont.post_adjust",
2500       fontdata)
2501   end
2502   luatexbase.call_callback("mathfont.finishing_touches", fontdata)
2503 end

```

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

```

2504 luatexbase.add_to_callback("luaotfload.patch_font",
2505   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 start with uppercase Latin letters. The first set of numbers applies to upright fonts, and the second set applies to italic/slanted fonts.

```

2506 mathfont:new_type_a(0x41, {0,0,0,0},    {50,0,150,0})    %A
2507 mathfont:new_type_a(0x42, {0,0,0,0},    {0,0,0,0})        %B

```

```

2508 mathfont:new_type_a(0x43, {0,0,50,0}, {0,0,50,0}) %C
2509 mathfont:new_type_a(0x44, {0,0,0,0}, {50,0,0,0}) %D
2510 mathfont:new_type_a(0x45, {0,0,0,0}, {50,0,0,0}) %E
2511 mathfont:new_type_a(0x46, {0,0,0,0}, {50,0,0,0}) %F
2512 mathfont:new_type_a(0x47, {0,0,0,0}, {0,0,0,0}) %G
2513 mathfont:new_type_a(0x48, {0,0,0,0}, {50,0,0,0}) %H
2514 mathfont:new_type_a(0x49, {0,0,0,0}, {50,0,50,0}) %I
2515 mathfont:new_type_a(0x4A, {0,0,100,0}, {50,0,100,0}) %J
2516 mathfont:new_type_a(0x4B, {0,0,0,0}, {50,0,0,0}) %K
2517 mathfont:new_type_a(0x4C, {0,0,-200,0}, {50,0,-100,0}) %L
2518 mathfont:new_type_a(0x4D, {0,0,0,0}, {50,0,0,0}) %M
2519 mathfont:new_type_a(0x4E, {0,0,0,0}, {50,0,0,0}) %N
2520 mathfont:new_type_a(0x4F, {0,0,0,0}, {0,0,50,0}) %O
2521 mathfont:new_type_a(0x50, {0,0,0,0}, {50,0,0,0}) %P
2522 mathfont:new_type_a(0x51, {0,0,0,0}, {0,0,50,0}) %Q
2523 mathfont:new_type_a(0x52, {0,0,0,0}, {50,0,0,0}) %R
2524 mathfont:new_type_a(0x53, {0,0,0,0}, {0,0,0,0}) %S
2525 mathfont:new_type_a(0x54, {0,0,0,0}, {0,0,0,0}) %T
2526 mathfont:new_type_a(0x55, {0,0,0,0}, {0,0,0,0}) %U
2527 mathfont:new_type_a(0x56, {0,0,0,0}, {0,0,0,0}) %V
2528 mathfont:new_type_a(0x57, {0,0,0,0}, {0,0,0,0}) %W
2529 mathfont:new_type_a(0x58, {0,0,0,0}, {50,0,0,0}) %X
2530 mathfont:new_type_a(0x59, {0,0,0,0}, {0,0,0,0}) %Y
2531 mathfont:new_type_a(0x5A, {0,0,0,0}, {50,0,0,0}) %Z

```

Lowercase Latin letters.

```

2532 mathfont:new_type_a(0x61, {0,0,0,0}, {0,0,0,0}) %a
2533 mathfont:new_type_a(0x62, {0,0,0,0}, {0,0,0,0}) %b
2534 mathfont:new_type_a(0x63, {0,0,0,0}, {0,0,100,0}) %c
2535 mathfont:new_type_a(0x64, {0,0,0,0}, {0,0,0,0}) %d
2536 mathfont:new_type_a(0x65, {0,0,0,0}, {0,0,100,0}) %e
2537 mathfont:new_type_a(0x66, {0,400,300,0}, {150,0,50,0}) %f
2538 mathfont:new_type_a(0x67, {0,0,0,0}, {0,0,0,0}) %g
2539 mathfont:new_type_a(0x68, {0,0,0,0}, {0,0,0,0}) %h
2540 mathfont:new_type_a(0x69, {0,0,0,0}, {0,0,0,0}) %i
2541 mathfont:new_type_a(0x6A, {100,0,0,0}, {200,0,0,0}) %j
2542 mathfont:new_type_a(0x6B, {0,0,0,0}, {0,0,0,0}) %k
2543 mathfont:new_type_a(0x6C, {0,0,0,0}, {0,0,0,0}) %l
2544 mathfont:new_type_a(0x6D, {0,0,0,0}, {0,0,50,0}) %m
2545 mathfont:new_type_a(0x6E, {0,0,0,0}, {0,0,50,0}) %n
2546 mathfont:new_type_a(0x6F, {0,0,0,0}, {0,0,100,0}) %o
2547 mathfont:new_type_a(0x70, {0,0,0,0}, {0,0,100,0}) %p
2548 mathfont:new_type_a(0x71, {0,0,0,0}, {0,0,0,0}) %q

```

```

2549 mathfont:new_type_a(0x72, {0,0,0,0}, {0,0,50,0}) %r
2550 mathfont:new_type_a(0x73, {0,0,0,0}, {0,0,0,0}) %s
2551 mathfont:new_type_a(0x74, {0,0,0,0}, {0,0,0,0}) %t
2552 mathfont:new_type_a(0x75, {0,0,0,0}, {0,0,0,0}) %u
2553 mathfont:new_type_a(0x76, {0,0,0,0}, {0,0,50,0}) %v
2554 mathfont:new_type_a(0x77, {0,0,0,0}, {0,0,50,0}) %w
2555 mathfont:new_type_a(0x78, {0,0,0,0}, {0,0,50,0}) %x
2556 mathfont:new_type_a(0x79, {0,0,0,0}, {0,0,50,0}) %y
2557 mathfont:new_type_a(0x7A, {0,0,0,0}, {0,0,0,0}) %z
2558 mathfont:new_type_a(0x131, {0,0,0,0}, {0,0,50,0}) %\imath
2559 mathfont:new_type_a(0x237, {0,0,0,0}, {200,0,50,0}) %\jmath

```

Uppercase Greek characters.

```

2560 mathfont:new_type_a(0x391, {0,0,0,0}, {50,0,150,0}) %\Alpha
2561 mathfont:new_type_a(0x392, {0,0,0,0}, {50,0,0,0}) %\Beta
2562 mathfont:new_type_a(0x393, {0,0,0,0}, {50,0,0,0}) %\Gamma
2563 mathfont:new_type_a(0x394, {0,0,0,0}, {50,0,150,0}) %\Delta
2564 mathfont:new_type_a(0x395, {0,0,0,0}, {50,0,0,0}) %\Epsilon
2565 mathfont:new_type_a(0x396, {0,0,0,0}, {50,0,0,0}) %\Zeta
2566 mathfont:new_type_a(0x397, {0,0,0,0}, {50,0,0,0}) %\Eta
2567 mathfont:new_type_a(0x398, {0,0,0,0}, {0,0,50,0}) %\Theta
2568 mathfont:new_type_a(0x399, {0,0,0,0}, {50,0,0,0}) %\Iota
2569 mathfont:new_type_a(0x39A, {0,0,0,0}, {50,0,0,0}) %\Kappa
2570 mathfont:new_type_a(0x39B, {0,0,0,0}, {0,0,150,0}) %\Lambda
2571 mathfont:new_type_a(0x39C, {0,0,0,0}, {50,0,0,0}) %\Mu
2572 mathfont:new_type_a(0x39D, {0,0,0,0}, {50,0,0,0}) %\Nu
2573 mathfont:new_type_a(0x39E, {0,0,0,0}, {0,0,0,0}) %\Xi
2574 mathfont:new_type_a(0x39F, {0,0,0,0}, {0,0,50,0}) %\Omicron
2575 mathfont:new_type_a(0x3A0, {0,0,0,0}, {50,0,0,0}) %\Pi
2576 mathfont:new_type_a(0x3A1, {0,0,0,0}, {50,0,0,0}) %\Rho
2577 mathfont:new_type_a(0x3A3, {0,0,0,0}, {50,0,0,0}) %\Sigma
2578 mathfont:new_type_a(0x3A4, {0,0,0,0}, {0,0,0,0}) %\Tau
2579 mathfont:new_type_a(0x3A5, {0,0,0,0}, {0,0,0,0}) %\Upsilon
2580 mathfont:new_type_a(0x3A6, {0,0,0,0}, {0,0,50,0}) %\Phi
2581 mathfont:new_type_a(0x3A7, {0,0,0,0}, {50,0,0,0}) %\Chi
2582 mathfont:new_type_a(0x3A8, {0,0,0,0}, {0,0,0,0}) %\Psi
2583 mathfont:new_type_a(0x3A9, {0,0,0,0}, {0,0,50,0}) %\Omega
2584 mathfont:new_type_a(0x3F4, {0,0,0,0}, {0,0,50,0}) %\varTheta

```

Lowercase Greek characters.

```

2585 mathfont:new_type_a(0x3B1, {0,0,0,0}, {0,0,0,0}) %\alpha
2586 mathfont:new_type_a(0x3B2, {0,0,0,0}, {0,0,0,0}) %\beta
2587 mathfont:new_type_a(0x3B3, {0,0,0,0}, {0,0,0,0}) %\gamma

```

```

2588 mathfont:new_type_a(0x3B4, {0,0,0,0}, {0,0,0,0}) %\delta
2589 mathfont:new_type_a(0x3B5, {0,0,50,0}, {0,0,50,0}) %\epsilon
2590 mathfont:new_type_a(0x3B6, {0,0,50,0}, {0,0,0,0}) %\zeta
2591 mathfont:new_type_a(0x3B7, {0,0,50,0}, {0,0,0,0}) %\eta
2592 mathfont:new_type_a(0x3B8, {0,0,0,0}, {0,0,100,0}) %\theta
2593 mathfont:new_type_a(0x3B9, {0,0,0,0}, {0,0,50,0}) %\iota
2594 mathfont:new_type_a(0x3BA, {0,0,0,0}, {0,0,0,0}) %\kappa
2595 mathfont:new_type_a(0x3BB, {0,0,-150,0}, {0,0,-100,0}) %\lambda
2596 mathfont:new_type_a(0x3BC, {0,0,0,0}, {0,0,100,0}) %\mu
2597 mathfont:new_type_a(0x3BD, {0,0,0,0}, {0,0,50,0}) %\nu
2598 mathfont:new_type_a(0x3BE, {0,0,0,0}, {0,0,50,0}) %\xi
2599 mathfont:new_type_a(0x3BF, {0,0,0,0}, {0,0,50,0}) %\omicron
2600 mathfont:new_type_a(0x3C0, {0,0,0,0}, {0,0,0,0}) %\pi
2601 mathfont:new_type_a(0x3C1, {0,0,0,0}, {0,0,50,0}) %\rho
2602 mathfont:new_type_a(0x3C3, {0,0,0,0}, {0,0,0,0}) %\sigma
2603 mathfont:new_type_a(0x3C4, {0,0,0,0}, {0,0,50,0}) %\tau
2604 mathfont:new_type_a(0x3C5, {0,0,0,0}, {0,0,0,0}) %\upsilon
2605 mathfont:new_type_a(0x3C6, {0,0,0,0}, {0,0,0,0}) %\phi
2606 mathfont:new_type_a(0x3C7, {0,0,0,0}, {50,0,50,0}) %\chi
2607 mathfont:new_type_a(0x3C8, {0,0,0,0}, {0,0,0,0}) %\psi
2608 mathfont:new_type_a(0x3C9, {0,0,0,0}, {0,0,50,0}) %\omega
2609 mathfont:new_type_a(0x3D0, {0,0,0,0}, {0,0,0,0}) %\varbeta
2610 mathfont:new_type_a(0x3F5, {0,0,0,0}, {0,0,50,0}) %\varepsilon
2611 mathfont:new_type_a(0x3D1, {0,0,150,0}, {0,0,100,0}) %\vartheta
2612 mathfont:new_type_a(0x3F1, {0,0,0,0}, {0,0,50,0}) %\varrho
2613 mathfont:new_type_a(0x3C2, {0,0,100,0}, {0,0,50,0}) %\varsigma
2614 mathfont:new_type_a(0x3D5, {0,0,0,0}, {0,0,100,0}) %\varphi

```

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`.

```

2615 local delim_glyphs = {40, % (
2616   41, % )
2617   47, % /
2618   91, % [
2619   92, % backslash
2620   93, % ]
2621  123, % {
2622  125, % }
2623 8249, % \lguil

```

```

2624 8250, % \rguil
2625 171, % \llguil
2626 187, % \rrguil
2627 mathfont.fakel, % \fakelangle
2628 mathfont.faker, % \fakerangle
2629 mathfont.fakell, % \fakellangle
2630 mathfont.fakerr} % \fakerrangle
2631 local big_op_glyphs = {33, % !
2632 35, % #
2633 36, % $
2634 37, % %
2635 38, % &
2636 43, % +
2637 63, % ?
2638 64, % @
2639 167, % \S
2640 215, % \times
2641 247, % \div
2642 8719, % \prod
2643 8721, % \sum
2644 8720, % \coprod
2645 8897, % \bigvee
2646 8896, % \bigwedge
2647 8899, % \bigcup
2648 8898, % \bigcap
2649 10753, % \bigoplus
2650 10754, % \bigotimes
2651 10752, % \bigodot
2652 10757, % \bigsqcap
2653 10758} % \bigsqcup
2654 local vert_glyphs = {124, 8730} % | and \surd
2655 local int_glyphs = {8747, % \intop
2656 8748, % \iint
2657 8749, % \iiint
2658 8750, % \oint
2659 8751, % \oiint
2660 8752} % \oiiint

```

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

```

2661 local delim_scale = {}
2662 local big_op_scale = {}
2663 local vert_scale = {}

```

```
2664 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.

```
2665 for i = 1, 15, 1 do
2666   delim_scale[2*i-1] = 1000 + 100*i % delimiters - horizontal
2667   delim_scale[2*i] = 1000 + 500*i   % delimiters - vertical
2668   vert_scale[2*i-1] = 1000
2669   vert_scale[2*i] = 1000 + 500*i    % vertically scaled chars
2670   big_op_scale[2*i-1] = 1000 + 100*i % big operators - horizontal
2671   big_op_scale[2*i] = 1000 + 100*i  % big operators - vertical
```

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

```
2672   int_scale[2*i-1] = 1000 + 500*i    % integral sign - horizontal
2673   int_scale[2*i] = 1000 + 1500*i    % integral sign - vertical
2674 end
```

We do not modify accent placement or italic corrections.

```
2675 delim_scale[31] = 0
2676 delim_scale[32] = 0
2677 delim_scale[33] = 0
2678 big_op_scale[31] = 0
2679 big_op_scale[32] = 0
2680 big_op_scale[33] = 0
2681 vert_scale[31] = 0
2682 vert_scale[32] = 0
2683 vert_scale[33] = 0
2684 int_scale[31] = 0
2685 int_scale[32] = 0
2686 int_scale[33] = 0
```

The wrapper for `:new_type_e`. We feed it a list of characters to create charm information for and a table of scaling information.

```
2687 function mathfont:add_extensible_variants(char_list, scale_list)
2688   local variants = (\string# scale_list - 3) / 2
2689   for i = 1, \string# char_list, 1 do
2690     self:new_type_e(char_list[i], scale_list, scale_list)
2691   end
2692 end
```

Add the charm information for the type e characters.

```
2693 mathfont:add_extensible_variants(delim_glyphs, delim_scale)
2694 mathfont:add_extensible_variants(big_op_glyphs, big_op_scale)
2695 mathfont:add_extensible_variants(vert_glyphs, vert_scale)
2696 mathfont:add_extensible_variants(int_glyphs, int_scale)
```

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

```
2697 }
2698 \fi % matches previous \ifM@adjust@font
```

12 Unicode Hex Values

For this section, we don't want any `\endlinechars` present when \TeX scans things because we want to eliminate any extra spaces, so before anything else, we set `\endlinechar` to `-1`.

```
2699 \count@\endlinechar
2700 \endlinechar\m@ne
```

We have to save `\mathchar@type` to use after `\begin{document}` because \LaTeX feeds it to `\@onlypreamble`.

```
2701 \let\@@mathchar@type\mathchar@type
```

We define `\M@sym@`, which is a wrapper around `\Umathchardef` or `\Umathcode` and is `mathfont`'s version of `\DeclareMathSymbol`. Before version 3.0, `mathfont` used `\DeclareMathSymbol`, but we create our own version to support Unicode input. The command first checks whether `#1` is a control sequence. If yes, we define it using `\Umathchardef` and again using `\Umathcode`. Otherwise, we define it once with `\Umathcode`.

```
2702 \def\M@sym@#1#2#3#4{
2703   \ifcat\relax\noexpand#1
```

Check if we're redefining a previously declared math symbol. We put `mathchar` in `\if@` so we don't have to check for `\mathchar` and `\Umathchar` separately. We use `\string` so that the letters in `mathchar` have catcode 12 when we check

for their presence in `\meaning#1` with `\in@`. If `#1` is undefined, `\in@` will set `\ifin@` to false.

```
2704 \expandafter\in@\expanded
2705     {\expandafter\@gobble\string\mathchar}\{\meaning#1}}
2706 \ifin@
```

Now redeclare the symbol.

```
2707 \Umathchardef#1+=\@@mathchar@type#2
2708     +\csname sym#3\endcsname+#4\relax
```

The next two lines implement Unicode input.

```
2709 \Umathcode #4+=\@@mathchar@type#2
2710     +\csname sym#3\endcsname+#4\relax
```

If `#1` does not code for a math symbol, we check whether it is already defined. If no, we define it, and if yes, we issue an error. Again, we implement Unicode input.

```
2711 \else
2712     \ifx#1\@undefined
2713         \Umathchardef#1+=\@@mathchar@type#2
2714             +\csname sym#3\endcsname+#4\relax
2715         \Umathcode #4+=\@@mathchar@type#2
2716             +\csname sym#3\endcsname+#4\relax
2717     \else
2718         \@latex@error{Command "\string#1" already defined}\@eha
2719     \fi
2720 \fi
```

Easy to deal with the case where `#1` is a single character.

```
2721 \else
2722     \Umathcode`#1+=\@@mathchar@type#2+\csname sym#3\endcsname
2723         +\relax
2724 \fi}
```

Similar deal for accents; `\M@acc@` is our version of `\DeclareMathAccent`. Newer versions of the L^AT_EX kernel define math accents as robust commands, so we have to incorporate a check for robustness as well. We let `\@tempswa` be true or false according to whether we can (re)define the control sequence `#1` as a math accent.

```
2725 \def\M@acc@#1#2#3#4{
2726     \begingroup
2727     \@tempswatrue
2728     \ifdefined#1
2729         \expandafter\in@\expanded{
2730             {\expandafter\@gobble\string\mathaccent}
```



```

2731      {\meaning#1}}
2732  \ifin@
2733  \else
2734    \begingroup
2735      \escapechar\m@ne
2736      \expandafter
2737      \endgroup
2738      \expandafter\in@\expanded{
2739        {\string\mathaccent}
2740        {\expandafter\meaning\csname\string#1\space\endcsname}}
2741  \ifin@
2742  \else
2743    \@tempswafalse
2744  \fi
2745  \fi
2746  \fi
2747  \expandafter
2748  \endgroup

```

Now (re)define the command or issue an error.

```

2749  \if@tempswa
2750    \protected\edef#1{\Umathaccent+\@mathchar@type#2
2751      +\csname sym#3\endcsname+#4\relax}
2752  \else
2753    \@latex@error{Command "\string#1" already defined}\@eha
2754  \fi}

```

Set upper-case Latin characters. We use an `\edef` for `\M@upper@id` 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 mathcodes to use encoding slots in the Math Alphanumeric Symbols block.

```

\M@upper@set 2755 \def\M@upper@set{
2756   \edef\M@upper@id{M\@tempc-\M@uppershape}
2757   \M@sym@{A}{\mathalpha}{\M@upper@id}{`A}
2758   \M@sym@{B}{\mathalpha}{\M@upper@id}{`B}
2759   \M@sym@{C}{\mathalpha}{\M@upper@id}{`C}
2760   \M@sym@{D}{\mathalpha}{\M@upper@id}{`D}
2761   \M@sym@{E}{\mathalpha}{\M@upper@id}{`E}
2762   \M@sym@{F}{\mathalpha}{\M@upper@id}{`F}
2763   \M@sym@{G}{\mathalpha}{\M@upper@id}{`G}
2764   \M@sym@{H}{\mathalpha}{\M@upper@id}{`H}
2765   \M@sym@{I}{\mathalpha}{\M@upper@id}{`I}

```

```

2766 \M@sym@{J}{\mathalpha}{\M@upper@id}{`J}
2767 \M@sym@{K}{\mathalpha}{\M@upper@id}{`K}
2768 \M@sym@{L}{\mathalpha}{\M@upper@id}{`L}
2769 \M@sym@{M}{\mathalpha}{\M@upper@id}{`M}
2770 \M@sym@{N}{\mathalpha}{\M@upper@id}{`N}
2771 \M@sym@{O}{\mathalpha}{\M@upper@id}{`O}
2772 \M@sym@{P}{\mathalpha}{\M@upper@id}{`P}
2773 \M@sym@{Q}{\mathalpha}{\M@upper@id}{`Q}
2774 \M@sym@{R}{\mathalpha}{\M@upper@id}{`R}
2775 \M@sym@{S}{\mathalpha}{\M@upper@id}{`S}
2776 \M@sym@{T}{\mathalpha}{\M@upper@id}{`T}
2777 \M@sym@{U}{\mathalpha}{\M@upper@id}{`U}
2778 \M@sym@{V}{\mathalpha}{\M@upper@id}{`V}
2779 \M@sym@{W}{\mathalpha}{\M@upper@id}{`W}
2780 \M@sym@{X}{\mathalpha}{\M@upper@id}{`X}
2781 \M@sym@{Y}{\mathalpha}{\M@upper@id}{`Y}
2782 \M@sym@{Z}{\mathalpha}{\M@upper@id}{`Z}}

```

Set lower-case Latin characters.

```

\M@lower@set 2783 \def\M@lower@set{
2784   \edef\M@lower@id{M\@tempc-\M@lower@shape}
2785   \M@sym@{a}{\mathalpha}{\M@lower@id}{`a}
2786   \M@sym@{b}{\mathalpha}{\M@lower@id}{`b}
2787   \M@sym@{c}{\mathalpha}{\M@lower@id}{`c}
2788   \M@sym@{d}{\mathalpha}{\M@lower@id}{`d}
2789   \M@sym@{e}{\mathalpha}{\M@lower@id}{`e}
2790   \M@sym@{f}{\mathalpha}{\M@lower@id}{`f}
2791   \M@sym@{g}{\mathalpha}{\M@lower@id}{`g}
2792   \M@sym@{h}{\mathalpha}{\M@lower@id}{`h}
2793   \M@sym@{i}{\mathalpha}{\M@lower@id}{`i}
2794   \M@sym@{j}{\mathalpha}{\M@lower@id}{`j}
2795   \M@sym@{k}{\mathalpha}{\M@lower@id}{`k}
2796   \M@sym@{l}{\mathalpha}{\M@lower@id}{`l}
2797   \M@sym@{m}{\mathalpha}{\M@lower@id}{`m}
2798   \M@sym@{n}{\mathalpha}{\M@lower@id}{`n}
2799   \M@sym@{o}{\mathalpha}{\M@lower@id}{`o}
2800   \M@sym@{p}{\mathalpha}{\M@lower@id}{`p}
2801   \M@sym@{q}{\mathalpha}{\M@lower@id}{`q}
2802   \M@sym@{r}{\mathalpha}{\M@lower@id}{`r}
2803   \M@sym@{s}{\mathalpha}{\M@lower@id}{`s}
2804   \M@sym@{t}{\mathalpha}{\M@lower@id}{`t}
2805   \M@sym@{u}{\mathalpha}{\M@lower@id}{`u}
2806   \M@sym@{v}{\mathalpha}{\M@lower@id}{`v}

```

```

2807 \M@sym@{w}{\mathalpha}{\M@lower@id}{`w}
2808 \M@sym@{x}{\mathalpha}{\M@lower@id}{`x}
2809 \M@sym@{y}{\mathalpha}{\M@lower@id}{`y}
2810 \M@sym@{z}{\mathalpha}{\M@lower@id}{`z}
2811 \M@sym@{\imath}{\mathalpha}{\M@lower@id}{131}
2812 \M@sym@{\jmath}{\mathalpha}{\M@lower@id}{237}
2813 \let\hbar\@undefined
2814 \M@sym@{\hbar}{\mathord}{\M@lower@id}{127}}

```

Set diacritics.

```

\M@diacritics@set 2815 \def\M@diacritics@set{
2816 \edef\M@diacritics@id{M\@tempc-\M@diacriticssshape}
2817 \M@acc@{\acute} {\mathalpha}{\M@diacritics@id}{B4}
2818 \M@acc@{\aacute} {\mathalpha}{\M@diacritics@id}{2DD}
2819 \M@acc@{\dot} {\mathalpha}{\M@diacritics@id}{2D9}
2820 \M@acc@{\ddot} {\mathalpha}{\M@diacritics@id}{A8}
2821 \M@acc@{\grave} {\mathalpha}{\M@diacritics@id}{60}
2822 \M@acc@{\breve} {\mathalpha}{\M@diacritics@id}{2D8}
2823 \M@acc@{\hat} {\mathalpha}{\M@diacritics@id}{2C6}
2824 \M@acc@{\check} {\mathalpha}{\M@diacritics@id}{2C7}
2825 \M@acc@{\bar} {\mathalpha}{\M@diacritics@id}{2C9}
2826 \M@acc@{\mathring}{\mathalpha}{\M@diacritics@id}{2DA}
2827 \M@acc@{\tilde} {\mathalpha}{\M@diacritics@id}{2DC}}

```

Set capital Greek characters.

```

\M@greekupper@set 2828 \def\M@greekupper@set{
2829 \edef\M@greekupper@id{M\@tempc-\M@greekuppersshape}
2830 \M@sym@{\Alpha} {\mathalpha}{\M@greekupper@id}{391}
2831 \M@sym@{\Beta} {\mathalpha}{\M@greekupper@id}{392}
2832 \M@sym@{\Gamma} {\mathalpha}{\M@greekupper@id}{393}
2833 \M@sym@{\Delta} {\mathalpha}{\M@greekupper@id}{394}
2834 \M@sym@{\Epsilon} {\mathalpha}{\M@greekupper@id}{395}
2835 \M@sym@{\Zeta} {\mathalpha}{\M@greekupper@id}{396}
2836 \M@sym@{\Eta} {\mathalpha}{\M@greekupper@id}{397}
2837 \M@sym@{\Theta} {\mathalpha}{\M@greekupper@id}{398}
2838 \M@sym@{\Iota} {\mathalpha}{\M@greekupper@id}{399}
2839 \M@sym@{\Kappa} {\mathalpha}{\M@greekupper@id}{39A}
2840 \M@sym@{\Lambda} {\mathalpha}{\M@greekupper@id}{39B}
2841 \M@sym@{\Mu} {\mathalpha}{\M@greekupper@id}{39C}
2842 \M@sym@{\Nu} {\mathalpha}{\M@greekupper@id}{39D}
2843 \M@sym@{\Xi} {\mathalpha}{\M@greekupper@id}{39E}
2844 \M@sym@{\Omicron} {\mathalpha}{\M@greekupper@id}{39F}
2845 \M@sym@{\Pi} {\mathalpha}{\M@greekupper@id}{3A0}

```

```

2846 \M@sym@{\Rho}      {\mathalpha}{\M@greekupper@id}{"3A1}
2847 \M@sym@{\Sigma}    {\mathalpha}{\M@greekupper@id}{"3A3}
2848 \M@sym@{\Tau}       {\mathalpha}{\M@greekupper@id}{"3A4}
2849 \M@sym@{\Upsilon}   {\mathalpha}{\M@greekupper@id}{"3A5}
2850 \M@sym@{\Phi}       {\mathalpha}{\M@greekupper@id}{"3A6}
2851 \M@sym@{\Chi}       {\mathalpha}{\M@greekupper@id}{"3A7}
2852 \M@sym@{\Psi}       {\mathalpha}{\M@greekupper@id}{"3A8}
2853 \M@sym@{\Omega}     {\mathalpha}{\M@greekupper@id}{"3A9}
2854 \M@sym@{\varTheta} {\mathalpha}{\M@greekupper@id}{"3F4}

```

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

```

2855 \ifM@adjust@font
2856   \ifM@symbols\else
2857     \M@sym@{\increment}
2858     {\mathord}{\M@greekupper@id}{"2206}
2859     \M@sym@{\nabla}
2860     {\mathord}{\M@greekupper@id}{"2207}
2861   \fi
2862 \else
2863   \ifM@symbols\else
2864     \M@sym@{\increment}
2865     {\mathord}{\M@greekupper@id}{"2206}
2866   \fi
2867   \ifM@extsymbols\else
2868     \M@sym@{\nabla}
2869     {\mathord}{\M@greekupper@id}{"2207}
2870   \fi
2871 \fi}

```

Set minuscule Greek characters.

```

\M@greeklower@set 2872 \def\M@greeklower@set{
2873   \edef\M@greeklower@id{\M@tempc-\M@greeklower@shape}
2874   \M@sym@{\alpha}    {\mathalpha}{\M@greeklower@id}{"3B1}
2875   \M@sym@{\beta}     {\mathalpha}{\M@greeklower@id}{"3B2}
2876   \M@sym@{\gamma}    {\mathalpha}{\M@greeklower@id}{"3B3}
2877   \M@sym@{\delta}    {\mathalpha}{\M@greeklower@id}{"3B4}
2878   \M@sym@{\epsilon}   {\mathalpha}{\M@greeklower@id}{"3B5}
2879   \M@sym@{\zeta}     {\mathalpha}{\M@greeklower@id}{"3B6}
2880   \M@sym@{\eta}      {\mathalpha}{\M@greeklower@id}{"3B7}
2881   \M@sym@{\theta}    {\mathalpha}{\M@greeklower@id}{"3B8}
2882   \M@sym@{\iota}     {\mathalpha}{\M@greeklower@id}{"3B9}
2883   \M@sym@{\kappa}    {\mathalpha}{\M@greeklower@id}{"3BA}

```

```

2884 \M@sym@{\lambda} {\mathalpha}{\M@greeklower@id}{\3BB}
2885 \M@sym@{\mu} {\mathalpha}{\M@greeklower@id}{\3BC}
2886 \M@sym@{\nu} {\mathalpha}{\M@greeklower@id}{\3BD}
2887 \M@sym@{\xi} {\mathalpha}{\M@greeklower@id}{\3BE}
2888 \M@sym@{\omicron} {\mathalpha}{\M@greeklower@id}{\3BF}
2889 \M@sym@{\pi} {\mathalpha}{\M@greeklower@id}{\3C0}
2890 \M@sym@{\rho} {\mathalpha}{\M@greeklower@id}{\3C1}
2891 \M@sym@{\sigma} {\mathalpha}{\M@greeklower@id}{\3C3}
2892 \M@sym@{\tau} {\mathalpha}{\M@greeklower@id}{\3C4}
2893 \M@sym@{\upsilon} {\mathalpha}{\M@greeklower@id}{\3C5}
2894 \M@sym@{\phi} {\mathalpha}{\M@greeklower@id}{\3C6}
2895 \M@sym@{\chi} {\mathalpha}{\M@greeklower@id}{\3C7}
2896 \M@sym@{\psi} {\mathalpha}{\M@greeklower@id}{\3C8}
2897 \M@sym@{\omega} {\mathalpha}{\M@greeklower@id}{\3C9}
2898 \M@sym@{\varbeta} {\mathalpha}{\M@greeklower@id}{\3D0}
2899 \M@sym@{\varepsilon} {\mathalpha}{\M@greeklower@id}{\3F5}
2900 \M@sym@{\varkappa} {\mathalpha}{\M@greeklower@id}{\3F0}
2901 \M@sym@{\vartheta} {\mathalpha}{\M@greeklower@id}{\3D1}
2902 \M@sym@{\varrho} {\mathalpha}{\M@greeklower@id}{\3F1}
2903 \M@sym@{\varsigma} {\mathalpha}{\M@greeklower@id}{\3C2}
2904 \M@sym@{\varphi} {\mathalpha}{\M@greeklower@id}{\3D5}}

```

Set capital ancient Greek characters.

```

\M@agreekupper@set 2905 \def\M@agreekupper@set{
2906   \edef\M@agreekupper@id{M\@tempc-\M@agreekupper@shape}
2907   \M@sym@{\Heta} {\mathalpha}{\M@agreekupper@id}{\370}
2908   \M@sym@{\Sampi} {\mathalpha}{\M@agreekupper@id}{\3E0}
2909   \M@sym@{\Digamma} {\mathalpha}{\M@agreekupper@id}{\3DC}
2910   \M@sym@{\Koppa} {\mathalpha}{\M@agreekupper@id}{\3D8}
2911   \M@sym@{\Stigma} {\mathalpha}{\M@agreekupper@id}{\3DA}
2912   \M@sym@{\Sho} {\mathalpha}{\M@agreekupper@id}{\3F7}
2913   \M@sym@{\San} {\mathalpha}{\M@agreekupper@id}{\3FA}
2914   \M@sym@{\varSampi} {\mathalpha}{\M@agreekupper@id}{\372}
2915   \M@sym@{\varDigamma} {\mathalpha}{\M@agreekupper@id}{\376}
2916   \M@sym@{\varKoppa} {\mathalpha}{\M@agreekupper@id}{\3DE}}

```

Set minuscule ancient Greek characters.

```

\M@agreeklower@set 2917 \def\M@agreeklower@set{
2918   \edef\M@agreeklower@id{M\@tempc-\M@agreeklower@shape}
2919   \M@sym@{\heta} {\mathalpha}{\M@agreeklower@id}{\371}
2920   \M@sym@{\sampi} {\mathalpha}{\M@agreeklower@id}{\3E1}
2921   \M@sym@{\digamma} {\mathalpha}{\M@agreeklower@id}{\3DD}
2922   \M@sym@{\koppa} {\mathalpha}{\M@agreeklower@id}{\3D9}

```

```

2923 \M@sym@{\stigma}    {\mathalpha}{\M@agreeklower@id}{"3DB}
2924 \M@sym@{\sho}      {\mathalpha}{\M@agreeklower@id}{"3F8}
2925 \M@sym@{\san}      {\mathalpha}{\M@agreeklower@id}{"3FB}
2926 \M@sym@{\varsampi} {\mathalpha}{\M@agreeklower@id}{"373}
2927 \M@sym@{\vardigamma}{\mathalpha}{\M@agreeklower@id}{"377}
2928 \M@sym@{\varkoppa} {\mathalpha}{\M@agreeklower@id}{"3DF}}

```

Set capital Cyrillic characters.

```

\M@cyrillicupper@s 2929 \def\M@cyrillicupper@set{
2930   \edef\M@cyrillicupper@id{M\@tempc-\M@cyrillicuppershape}
2931   \M@sym@{\cyrA}    {\mathalpha}{\M@cyrillicupper@id}{"410}
2932   \M@sym@{\cyrBe}   {\mathalpha}{\M@cyrillicupper@id}{"411}
2933   \M@sym@{\cyrVe}   {\mathalpha}{\M@cyrillicupper@id}{"412}
2934   \M@sym@{\cyrGhe}  {\mathalpha}{\M@cyrillicupper@id}{"413}
2935   \M@sym@{\cyrDe}   {\mathalpha}{\M@cyrillicupper@id}{"414}
2936   \M@sym@{\cyrIe}   {\mathalpha}{\M@cyrillicupper@id}{"415}
2937   \M@sym@{\cyrZhe}  {\mathalpha}{\M@cyrillicupper@id}{"416}
2938   \M@sym@{\cyrZe}   {\mathalpha}{\M@cyrillicupper@id}{"417}
2939   \M@sym@{\cyrI}    {\mathalpha}{\M@cyrillicupper@id}{"418}
2940   \M@sym@{\cyrKa}   {\mathalpha}{\M@cyrillicupper@id}{"41A}
2941   \M@sym@{\cyrEl}   {\mathalpha}{\M@cyrillicupper@id}{"41B}
2942   \M@sym@{\cyrEm}   {\mathalpha}{\M@cyrillicupper@id}{"41C}
2943   \M@sym@{\cyrEn}   {\mathalpha}{\M@cyrillicupper@id}{"41D}
2944   \M@sym@{\cyrO}    {\mathalpha}{\M@cyrillicupper@id}{"41E}
2945   \M@sym@{\cyrPe}   {\mathalpha}{\M@cyrillicupper@id}{"41F}
2946   \M@sym@{\cyrEr}   {\mathalpha}{\M@cyrillicupper@id}{"420}
2947   \M@sym@{\cyrEs}   {\mathalpha}{\M@cyrillicupper@id}{"421}
2948   \M@sym@{\cyrTe}   {\mathalpha}{\M@cyrillicupper@id}{"422}
2949   \M@sym@{\cyrU}    {\mathalpha}{\M@cyrillicupper@id}{"423}
2950   \M@sym@{\cyrEf}   {\mathalpha}{\M@cyrillicupper@id}{"424}
2951   \M@sym@{\cyrHa}   {\mathalpha}{\M@cyrillicupper@id}{"425}
2952   \M@sym@{\cyrTse}  {\mathalpha}{\M@cyrillicupper@id}{"426}
2953   \M@sym@{\cyrChe}  {\mathalpha}{\M@cyrillicupper@id}{"427}
2954   \M@sym@{\cyrSha}  {\mathalpha}{\M@cyrillicupper@id}{"428}
2955   \M@sym@{\cyrShcha}{\mathalpha}{\M@cyrillicupper@id}{"429}
2956   \M@sym@{\cyrHard} {\mathalpha}{\M@cyrillicupper@id}{"42A}
2957   \M@sym@{\cyrYeru} {\mathalpha}{\M@cyrillicupper@id}{"42B}
2958   \M@sym@{\cyrSoft} {\mathalpha}{\M@cyrillicupper@id}{"42C}
2959   \M@sym@{\cyrE}    {\mathalpha}{\M@cyrillicupper@id}{"42D}
2960   \M@sym@{\cyrYu}   {\mathalpha}{\M@cyrillicupper@id}{"42E}
2961   \M@sym@{\cyrYa}   {\mathalpha}{\M@cyrillicupper@id}{"42F}
2962   \M@sym@{\cyrvarI} {\mathalpha}{\M@cyrillicupper@id}{"419}}

```

Set minuscule Cyrillic characters.

```

\M@cyrilliclower@s 2963 \def\M@cyrilliclower@set{
2964   \edef\M@cyrilliclower@id{M\@tempc-\M@cyrilliclower@shape}
2965   \M@sym@{\cyra}    {\mathalpha}{\M@cyrilliclower@id}{430}
2966   \M@sym@{\cyrbe}   {\mathalpha}{\M@cyrilliclower@id}{431}
2967   \M@sym@{\cyurve}  {\mathalpha}{\M@cyrilliclower@id}{432}
2968   \M@sym@{\cyrghe}  {\mathalpha}{\M@cyrilliclower@id}{433}
2969   \M@sym@{\cyrde}   {\mathalpha}{\M@cyrilliclower@id}{434}
2970   \M@sym@{\cyrie}   {\mathalpha}{\M@cyrilliclower@id}{435}
2971   \M@sym@{\cyrzhe}  {\mathalpha}{\M@cyrilliclower@id}{436}
2972   \M@sym@{\cyrze}   {\mathalpha}{\M@cyrilliclower@id}{437}
2973   \M@sym@{\cyri}    {\mathalpha}{\M@cyrilliclower@id}{438}
2974   \M@sym@{\cyrka}   {\mathalpha}{\M@cyrilliclower@id}{43A}
2975   \M@sym@{\cyrel}   {\mathalpha}{\M@cyrilliclower@id}{43B}
2976   \M@sym@{\cyrem}   {\mathalpha}{\M@cyrilliclower@id}{43C}
2977   \M@sym@{\cyren}   {\mathalpha}{\M@cyrilliclower@id}{43D}
2978   \M@sym@{\cyro}    {\mathalpha}{\M@cyrilliclower@id}{43E}
2979   \M@sym@{\cyrpe}   {\mathalpha}{\M@cyrilliclower@id}{43F}
2980   \M@sym@{\cyrer}   {\mathalpha}{\M@cyrilliclower@id}{440}
2981   \M@sym@{\cyres}   {\mathalpha}{\M@cyrilliclower@id}{441}
2982   \M@sym@{\cyrte}   {\mathalpha}{\M@cyrilliclower@id}{442}
2983   \M@sym@{\cyru}    {\mathalpha}{\M@cyrilliclower@id}{443}
2984   \M@sym@{\cyref}   {\mathalpha}{\M@cyrilliclower@id}{444}
2985   \M@sym@{\cyrha}   {\mathalpha}{\M@cyrilliclower@id}{445}
2986   \M@sym@{\cyrts}   {\mathalpha}{\M@cyrilliclower@id}{446}
2987   \M@sym@{\cyrche}  {\mathalpha}{\M@cyrilliclower@id}{447}
2988   \M@sym@{\cyrsha}  {\mathalpha}{\M@cyrilliclower@id}{448}
2989   \M@sym@{\cyrshcha}{\mathalpha}{\M@cyrilliclower@id}{449}
2990   \M@sym@{\cyrhard} {\mathalpha}{\M@cyrilliclower@id}{44A}
2991   \M@sym@{\cyreru}  {\mathalpha}{\M@cyrilliclower@id}{44B}
2992   \M@sym@{\cyrsoft} {\mathalpha}{\M@cyrilliclower@id}{44C}
2993   \M@sym@{\cyre}    {\mathalpha}{\M@cyrilliclower@id}{44D}
2994   \M@sym@{\cyrYu}   {\mathalpha}{\M@cyrilliclower@id}{44E}
2995   \M@sym@{\cyrya}   {\mathalpha}{\M@cyrilliclower@id}{44F}
2996   \M@sym@{\cyrvari} {\mathalpha}{\M@cyrilliclower@id}{439}}

```

Set Hebrew characters.

```

\M@hebrew@set 2997 \def\M@hebrew@set{
2998   \edef\M@hebrew@id{M\@tempc-\M@hebrew@shape}
2999   \M@sym@{\aleph}   {\mathalpha}{\M@hebrew@id}{5D0}
3000   \M@sym@{\beth}    {\mathalpha}{\M@hebrew@id}{5D1}
3001   \M@sym@{\gimel}   {\mathalpha}{\M@hebrew@id}{5D2}
3002   \M@sym@{\daleth}  {\mathalpha}{\M@hebrew@id}{5D3}
3003   \M@sym@{\he}      {\mathalpha}{\M@hebrew@id}{5D4}

```

```

3004 \M@sym@{\vav}      {\mathalpha}{\M@hebrew@id}{\5D5}
3005 \M@sym@{\zayin}    {\mathalpha}{\M@hebrew@id}{\5D6}
3006 \M@sym@{\het}      {\mathalpha}{\M@hebrew@id}{\5D7}
3007 \M@sym@{\tet}      {\mathalpha}{\M@hebrew@id}{\5D8}
3008 \M@sym@{\yod}      {\mathalpha}{\M@hebrew@id}{\5D9}
3009 \M@sym@{\kaf}      {\mathalpha}{\M@hebrew@id}{\5DB}
3010 \M@sym@{\lamed}    {\mathalpha}{\M@hebrew@id}{\5DC}
3011 \M@sym@{\mem}      {\mathalpha}{\M@hebrew@id}{\5DE}
3012 \M@sym@{\nun}      {\mathalpha}{\M@hebrew@id}{\5E0}
3013 \M@sym@{\samekh}   {\mathalpha}{\M@hebrew@id}{\5E1}
3014 \M@sym@{\ayin}     {\mathalpha}{\M@hebrew@id}{\5E2}
3015 \M@sym@{\pe}       {\mathalpha}{\M@hebrew@id}{\5E4}
3016 \M@sym@{\tsadi}    {\mathalpha}{\M@hebrew@id}{\5E6}
3017 \M@sym@{\qof}      {\mathalpha}{\M@hebrew@id}{\5E7}
3018 \M@sym@{\resh}     {\mathalpha}{\M@hebrew@id}{\5E8}
3019 \M@sym@{\shin}     {\mathalpha}{\M@hebrew@id}{\5E9}
3020 \M@sym@{\tav}      {\mathalpha}{\M@hebrew@id}{\5EA}
3021 \M@sym@{\varkaf}   {\mathalpha}{\M@hebrew@id}{\5DA}
3022 \M@sym@{\varmem}    {\mathalpha}{\M@hebrew@id}{\5DD}
3023 \M@sym@{\varnun}   {\mathalpha}{\M@hebrew@id}{\5DF}
3024 \M@sym@{\varpe}    {\mathalpha}{\M@hebrew@id}{\5E3}
3025 \M@sym@{\vartsadi} {\mathalpha}{\M@hebrew@id}{\5E5}

```

Set digits.

```

\M@digits@set 3026 \def\M@digits@set{
3027   \edef\M@digits@id{M\@tempc-\M@digitsshape}
3028   \M@sym@{0}{\mathalpha}{\M@digits@id}{\`0}
3029   \M@sym@{1}{\mathalpha}{\M@digits@id}{\`1}
3030   \M@sym@{2}{\mathalpha}{\M@digits@id}{\`2}
3031   \M@sym@{3}{\mathalpha}{\M@digits@id}{\`3}
3032   \M@sym@{4}{\mathalpha}{\M@digits@id}{\`4}
3033   \M@sym@{5}{\mathalpha}{\M@digits@id}{\`5}
3034   \M@sym@{6}{\mathalpha}{\M@digits@id}{\`6}
3035   \M@sym@{7}{\mathalpha}{\M@digits@id}{\`7}
3036   \M@sym@{8}{\mathalpha}{\M@digits@id}{\`8}
3037   \M@sym@{9}{\mathalpha}{\M@digits@id}{\`9}}

```

Set new operator font. We change the `\fam` to the user's requested symbol font for math operators.

```

\M@operator@set 3038 \def\M@operator@set{
\operator@font 3039   \edef\operator@font{\mathgroup
3040     \csname symM\@tempc-\M@operatorshape\endcsname}}

```

Set delimiters.


```

3041 \ifM@adjust@font
\M@delimiters@set 3042 \def\M@delimiters@set{
3043   \edef\M@delimiters@id{M\@tempc-\M@delimitersshape}
3044   \edef\M@delimiters@num{
3045     \csname sym\M@delimiters@id\endcsname}
3046   \M@sym@{({}          {\mathopen} {\M@delimiters@id}{"28}
3047   \M@sym@{)}          {\mathclose}{\M@delimiters@id}{"29}
3048   \M@sym@{[{}         {\mathopen} {\M@delimiters@id}{"5B}
3049   \M@sym@{]}          {\mathclose}{\M@delimiters@id}{"5D}
3050   \M@sym@{\leftbrace} {\mathopen} {\M@delimiters@id}{"7B}
3051   \M@sym@{\rightbrace}{\mathclose}{\M@delimiters@id}{"7D}

```

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

```

3052   \Udelcode"28+\M@delimiters@num+"28\relax % (
3053   \Udelcode"29+\M@delimiters@num+"29\relax % )
3054   \Udelcode"2F+\M@delimiters@num+"2F\relax % /
3055   \Udelcode"5B+\M@delimiters@num+"5B\relax % [
3056   \Udelcode"5D+\M@delimiters@num+"5D\relax % ]
3057   \Udelcode"7C+\M@delimiters@num+"7C\relax % |
3058   \ifM@symbols\else
3059     \M@sym@{|}{\mathord}{\M@delimiters@id}{"7C}
3060   \fi
3061   \let\vert=|

```

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

```

3062   \protected\def\backslash{
3063     \ifmmode\mathbackslash\else\textbackslash\fi}
3064   \protected\edef\mathbackslash{
3065     \Udelimiter+2+\M@delimiters@num+92\relax}
3066   \protected\edef\lbrace{
3067     \Udelimiter+4+\M@delimiters@num+123\relax}
3068   \protected\edef\rbrace{
3069     \Udelimiter+5+\M@delimiters@num+125\relax}
3070   \protected\edef\lguil{
3071     \Udelimiter+4+\M@delimiters@num+8249\relax}
3072   \protected\edef\rguil{
3073     \Udelimiter+5+\M@delimiters@num+8250\relax}
3074   \protected\edef\llguil{
3075     \Udelimiter+4+\M@delimiters@num+171\relax}
3076   \protected\edef\rrguil{
3077     \Udelimiter+5+\M@delimiters@num+187\relax}
3078   \protected\edef\fakelangle{

```

```

3079     \Udelimiter+4+\M@delimiters@num
3080     +\directlua{tex.print(mathfont.fakel)}\relax}
3081 \protected\edef\fakerangle{
3082     \Udelimiter+5+\M@delimiters@num
3083     +\directlua{tex.print(mathfont.faker)}\relax}
3084 \protected\edef\fakellangle{
3085     \Udelimiter+4+\M@delimiters@num
3086     +\directlua{tex.print(mathfont.fakell)}\relax}
3087 \protected\edef\fakerrangle{
3088     \Udelimiter+5+\M@delimiters@num
3089     +\directlua{tex.print(mathfont.fakerr)}\relax}}
3090 \else

```

```

\M@delimiters@set 3091 \def\M@delimiters@set{
3092     \edef\M@delimiters@id{M\@tempc-\M@delimiters@shape}
3093     \M@sym@{(}          {\mathopen} {\M@delimiters@id}{"28}
3094     \M@sym@{)}          {\mathclose}{\M@delimiters@id}{"29}
3095     \M@sym@{[}          {\mathopen} {\M@delimiters@id}{"5B}
3096     \M@sym@{]}          {\mathclose}{\M@delimiters@id}{"5D}
3097     \M@sym@{\lguil}     {\mathopen} {\M@delimiters@id}{"2039}
3098     \M@sym@{\rguil}     {\mathclose}{\M@delimiters@id}{"203A}
3099     \M@sym@{\llguil}    {\mathopen} {\M@delimiters@id}{"AB}
3100     \M@sym@{\rrguil}    {\mathclose}{\M@delimiters@id}{"BB}
3101     \M@sym@{\leftbrace} {\mathopen} {\M@delimiters@id}{"7B}
3102     \M@sym@{\rightbrace}{\mathclose}{\M@delimiters@id}{"7D}}
3103 \fi

```

Radicals. When we define `\surd` to typeset U+221A, `\M@sym@` sets the `\Umathcode` of $\sqrt{}$ to be a surd symbol. However, if we modified the font, we know the surd character in the requested font can successfully make a square root expression, so we override the definition from `\M@sym@` to turn $\sqrt{}$ to an active character in math mode.

```

3104 \ifM@adjust@font
\M@radical@set 3105 \def\M@radical@set{
3106     \edef\M@radical@id{M\@tempc-\M@radical@shape}
\surd 3107     \let\surd\@undefined
3108     \M@sym@{\surd}{\mathord}{\M@radical@id}{"221A}

```

Now set the `\mathcode` of $\sqrt{}$ to 8000. This is probably me being paranoid, but I wanted to stick to ascii characters in the sty file. We use `\directlua` to print the surd character instead. This also has the advantage of printing an active character, so we don't have to scan any tokens.

```

3109     \expandafter\protected\expandafter\def\directlua{
3110         tex.cprint(13, utf8.char(0x221A))}

```

```

3111      {\ifmmode\expandafter\sqrt\else\Uchar"221A\relax\fi}
3112      \mathcode"221A="8000\relax
\@sqrts@gn 3113      \edef\@sqrts@gn##1{\Uradical+\number
3114      \csname sym\M@radical@id\endcsname+"221A\relax{##1}}

```

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

```

\r@t 3115      \def\r@t##1##2{
3116      \setbox\z@\hbox{\$m@th##1\sqrtsign{##2}$}
3117      \setbox\surdbox\hbox{\$m@th##1\@sqrts@gn{
3118      \hbox{\vphantom{\$m@th##1##2$}}}$}
3119      \dimen@ht\surdbox
3120      \advance\dimen@dp\surdbox
3121      \dimen@=0.6\dimen@
3122      \advance\dimen@-\dp\surdbox
3123      \ifdim\wd\rootbox<0.6\wd\surdbox
3124      \kern0.6\wd\surdbox
3125      \else
3126      \kern\wd\rootbox
3127      \fi
3128      \raise\dimen@\hbox{\llap{\copy\rootbox}}
3129      \kern-0.6\wd\surdbox
3130      \box\z@}
\sqrtsign 3131      \protected\def\sqrtsign##1{
3132      \@sqrts@gn{\mkern\radicandoffset##1}}
3133 \else
\M@radical@set 3134      \def\M@radical@set{
3135      \edef\M@radical@id{M\@tempc-\M@radicalshape}
\surd 3136      \let\surd\@undefined
3137      \M@sym@{\surd}{\mathord}{\M@radical@id}{\text{"221A}}
3138 \fi

```

Big operators.

```

\M@bigops@set 3139 \def\M@bigops@set{
3140 \edef\M@bigops@id{M\@tempc-\M@bigopsshape}
3141 \let\sum\@undefined
3142 \let\prod\@undefined
3143 \M@sym@{\sum} {\mathop}{\M@bigops@id}{\text{"2211}}
3144 \M@sym@{\prod} {\mathop}{\M@bigops@id}{\text{"220F}}
3145 \M@sym@{\intop}{\mathop}{\M@bigops@id}{\text{"222B}}

```

Extended big operators.

```

\M@extbigops@set 3146 \def\M@extbigops@set{
3147   \edef\M@extbigops@id{M\@tempc-\M@extbigopsshape}
3148   \let\coprod\@undefined
3149   \let\bigvee\@undefined
3150   \let\bigwedge\@undefined
3151   \let\bigcup\@undefined
3152   \let\bigcap\@undefined
3153   \let\bigoplus\@undefined
3154   \let\bigotimes\@undefined
3155   \let\bigodot\@undefined
3156   \let\bigsqcup\@undefined
3157   \M@sym@\coprod    {\mathop}{\M@extbigops@id}{\string"2210}
3158   \M@sym@\bigvee    {\mathop}{\M@extbigops@id}{\string"22C1}
3159   \M@sym@\bigwedge  {\mathop}{\M@extbigops@id}{\string"22C0}
3160   \M@sym@\bigcup    {\mathop}{\M@extbigops@id}{\string"22C3}
3161   \M@sym@\bigcap    {\mathop}{\M@extbigops@id}{\string"22C2}
3162   \M@sym@\iintop    {\mathop}{\M@extbigops@id}{\string"222C}
3163   \M@sym@\iiintop   {\mathop}{\M@extbigops@id}{\string"222D}
3164   \M@sym@\ointop    {\mathop}{\M@extbigops@id}{\string"222E}
3165   \M@sym@\oiintop   {\mathop}{\M@extbigops@id}{\string"222F}
3166   \M@sym@\oiintop   {\mathop}{\M@extbigops@id}{\string"2230}
3167   \M@sym@\bigoplus  {\mathop}{\M@extbigops@id}{\string"2A01}
3168   \M@sym@\bigotimes {\mathop}{\M@extbigops@id}{\string"2A02}
3169   \M@sym@\bigodot   {\mathop}{\M@extbigops@id}{\string"2A00}
3170   \M@sym@\bigsqcap  {\mathop}{\M@extbigops@id}{\string"2A05}
3171   \M@sym@\bigsqcup  {\mathop}{\M@extbigops@id}{\string"2A06}
3172   \protected\def\iint{\iintop\nolimits}
3173   \protected\def\iiint{\iiintop\nolimits}
3174   \protected\def\oint{\ointop\nolimits}
3175   \protected\def\oiint{\oiintop\nolimits}
3176   \protected\def\oiint{\oiintop\nolimits}}

```

Set symbols.

```

\M@symbols@set 3177 \def\M@symbols@set{
3178   \edef\M@symbols@id{M\@tempc-\M@symbolsshape}
3179   \let\colon\@undefined
3180   \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.

```

3181 \ifM@arrows\else
3182   \Umathcharnumdef\@relbar=\Umathcodenum`\-
3183   \Umathcharnumdef\@Relbar=\Umathcodenum`\=
3184   \protected\def\relbar{\mathrel
3185     {\mathpalette\mathsm@sh\@relbar}}
3186   \protected\def\Relbar{\@Relbar}

```

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

```

3187 \ifpackageloaded{amsmath}
3188   {\relax}{
3189     \let\@@relbar\relbar
3190     \let\@@Relbar\Relbar
3191     \AtBeginDocument{\ifM@arrows\else
3192       \ifpackageloaded{amsmath}{
3193         \let\relbar\@@relbar
3194         \let\Relbar\@@Relbar}
3195       {\relax}
3196     \fi}}
3197 \fi

```

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 in case that package gets loaded ahead of `mathfont`, we should make sure to clear that definition. It's important to declare the big operators before the normal versions of these characters so that `\M@sym@` defines the correct `\Umathcode` for them.

```

3198 \ifM@adjust@font
3199   \let\bigtimes\@undefined
3200   \M@sym@{\bigat}      {\mathop}{\M@symbols@id}{\ "40}
3201   \M@sym@{\bighash}    {\mathop}{\M@symbols@id}{\ "23}
3202   \M@sym@{\bigdollar}  {\mathop}{\M@symbols@id}{\ "24}
3203   \M@sym@{\bigpercent} {\mathop}{\M@symbols@id}{\ "25}
3204   \M@sym@{\bigand}     {\mathop}{\M@symbols@id}{\ "26}
3205   \M@sym@{\bigplus}    {\mathop}{\M@symbols@id}{\ "2B}
3206   \M@sym@{\bigp}       {\mathop}{\M@symbols@id}{\ "21}
3207   \M@sym@{\bigq}       {\mathop}{\M@symbols@id}{\ "3F}

```

```

3208 \M@sym@{\bigS}      {\mathop}{\M@symbols@id}{A7}
3209 \M@sym@{\bigtimes}  {\mathop}{\M@symbols@id}{D7}
3210 \M@sym@{\bigdiv}    {\mathop}{\M@symbols@id}{F7}

```

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

```

3211 \M@sym@{\nabla}     {\mathord}{\M@symbols@id}{2207}
3212 \fi

```

The rest of the symbols.

```

3213 \M@sym@{.}          {\mathord}{\M@symbols@id}{2E}
3214 \M@sym@{@}           {\mathord}{\M@symbols@id}{40}
3215 \M@sym@{' }          {\mathord}{\M@symbols@id}{2032}
3216 \M@sym@{\prime}      {\mathord}{\M@symbols@id}{2032}
3217 \M@sym@{" }          {\mathord}{\M@symbols@id}{2033}
3218 \M@sym@{\mathhash}   {\mathord}{\M@symbols@id}{23}
3219 \M@sym@{\mathdollar} {\mathord}{\M@symbols@id}{24}
3220 \M@sym@{\mathpercent} {\mathord}{\M@symbols@id}{25}
3221 \M@sym@{\mathand}     {\mathord}{\M@symbols@id}{26}
3222 \M@sym@{\mathparagraph}{\mathord}{\M@symbols@id}{B6}
3223 \M@sym@{\mathsection} {\mathord}{\M@symbols@id}{A7}
3224 \let\mathsterling\@undefined
3225 \M@sym@{\mathsterling} {\mathord}{\M@symbols@id}{A3}
3226 \M@sym@{\neg}         {\mathord}{\M@symbols@id}{AC}
3227 \M@sym@{\mid}         {\mathrel}{\M@symbols@id}{7C}
3228 \M@sym@{|}            {\mathord}{\M@symbols@id}{7C}
3229 \M@sym@{\infty}       {\mathord}{\M@symbols@id}{221E}
3230 \M@sym@{\partial}      {\mathord}{\M@symbols@id}{2202}
3231 \M@sym@{\degree}      {\mathord}{\M@symbols@id}{B0}
3232 \M@sym@{\increment}   {\mathord}{\M@symbols@id}{2206}
3233 \M@sym@{+}            {\mathbin}{\M@symbols@id}{2B}
3234 \M@sym@{-}            {\mathbin}{\M@symbols@id}{2212}
3235 \M@sym@{*}            {\mathbin}{\M@symbols@id}{2A}
3236 \M@sym@{\times}       {\mathbin}{\M@symbols@id}{D7}
3237 \M@sym@{/}            {\mathord}{\M@symbols@id}{2F}
3238 \M@sym@{\fractionslash}{\mathord}{\M@symbols@id}{2215}
3239 \M@sym@{\div}          {\mathbin}{\M@symbols@id}{F7}
3240 \M@sym@{\pm}           {\mathbin}{\M@symbols@id}{B1}
3241 \M@sym@{\bullet}       {\mathbin}{\M@symbols@id}{2022}
3242 \M@sym@{\dagger}       {\mathbin}{\M@symbols@id}{2020}
3243 \M@sym@{\ddagger}      {\mathbin}{\M@symbols@id}{2021}
3244 \M@sym@{\cdot}         {\mathbin}{\M@symbols@id}{2219}
3245 \M@sym@{\setminus}     {\mathbin}{\M@symbols@id}{5C}

```

```

3246 \M@sym@{=} {\mathrel} {\M@symbols@id}{"3D}
3247 \M@sym@{<} {\mathrel} {\M@symbols@id}{"3C}
3248 \M@sym@{>} {\mathrel} {\M@symbols@id}{"3E}
3249 \M@sym@{\leq} {\mathrel} {\M@symbols@id}{"2264}
3250 \M@sym@{\geq} {\mathrel} {\M@symbols@id}{"2265}
3251 \M@sym@{\sim} {\mathrel} {\M@symbols@id}{"7E}
3252 \M@sym@{\approx} {\mathrel} {\M@symbols@id}{"2248}
3253 \M@sym@{\equiv} {\mathrel} {\M@symbols@id}{"2261}
3254 \M@sym@{\parallel} {\mathrel} {\M@symbols@id}{"2016}
3255 \M@sym@{\colon} {\mathpunct} {\M@symbols@id}{"3A}
3256 \M@sym@{:} {\mathrel} {\M@symbols@id}{"3A}
3257 \M@sym@{?} {\mathclose} {\M@symbols@id}{"3F}
3258 \M@sym@{!} {\mathclose} {\M@symbols@id}{"21}
3259 \M@sym@{\comma} {\mathord} {\M@symbols@id}{"2C}
3260 \M@sym@{,} {\mathpunct} {\M@symbols@id}{"2C}
3261 \M@sym@{;} {\mathpunct} {\M@symbols@id}{"3B}
3262 \M@sym@{\mathellipsis} {\mathinner} {\M@symbols@id}{"2026}

```

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`.

```

3263 \protected\def\#{\ifmmode\mathhash\else\char"23\relax\fi}
3264 \protected\def\%{\ifmmode\mathpercent\else\char"25\relax\fi}
3265 \protected\def\&{\ifmmode\mathand\else\char"26\relax\fi}
3266 \ifM@extsymbols\else
3267   \protected\def\simeq{
3268     \mathrel{\mathpalette\stack@flatrel{-}\sim}}
3269   \protected\def\cong{
3270     \mathrel{\mathpalette\stack@flatrel{=}\sim}}
3271 \fi
\models 3272 \protected\def\models{\mathrel{|\}\joinrel\mathrel{=}}

```

New definition for `\not`. We define it to accept a `#1` argument, which we store in an `\hbox` in the appropriate style. Then we typeset a `/` halfway across the distance of the `\hbox` and the `\hbox` itself. This approach manually positions the `/` halfway across the `#1` subformula instead of using a character that appears to the right of a slim bounding box as in traditional \TeX . In case any users want to access the old `\not` definition, we save it as `\negslash`.

```

\negslash 3273 \let\negslash\not
3274 \protected\def\not##1{\mathrel{\mathchoice

```

```

3275 {\setbox\@tempboxa\hbox{\$ \displaystyle##1\m@th$}
3276 \hbox{\hb@xt@\wd\@tempboxa{\hss$\displaystyle/\m@th$\hss}
3277 \llap{\box\@tempboxa}}}
3278 {\setbox\@tempboxa\hbox{\$ \textstyle##1\m@th$}
3279 \hbox{\hb@xt@\wd\@tempboxa{\hss$\textstyle/\m@th$\hss}
3280 \llap{\box\@tempboxa}}}
3281 {\setbox\@tempboxa\hbox{\$ \scriptstyle##1\m@th$}
3282 \hbox{\hb@xt@\wd\@tempboxa{\hss$\scriptstyle/\m@th$\hss}
3283 \llap{\box\@tempboxa}}}
3284 {\setbox\@tempboxa\hbox{\$ \scriptscriptstyle##1\m@th$}
3285 \hbox{\hb@xt@\wd\@tempboxa{\hss$\scriptscriptstyle/
3286 \m@th$\hss}
3287 \llap{\box\@tempboxa}}}}}}

```

Set extended symbols.

```

\M@extsymbols@set 3288 \def\M@extsymbols@set{
3289 \edef\M@extsymbols@id{M\@tempc-\M@extsymbolsshape}
3290 \let\angle\@undefined
3291 \let\simeq\@undefined
3292 \let\squsubset\@undefined
3293 \let\sqsupset\@undefined
3294 \let\bowtie\@undefined
3295 \let\doteq\@undefined
3296 \let\neq\@undefined
3297 \M@sym@\wp} {\mathord}{\M@extsymbols@id}{"2118}
3298 \M@sym@\ell} {\mathord}{\M@extsymbols@id}{"2113}
3299 \M@sym@\forall} {\mathord}{\M@extsymbols@id}{"2200}
3300 \M@sym@\exists} {\mathord}{\M@extsymbols@id}{"2203}
3301 \M@sym@\emptyset} {\mathord}{\M@extsymbols@id}{"2205}
3302 \M@sym@\in} {\mathord}{\M@extsymbols@id}{"2208}
3303 \M@sym@\ni} {\mathord}{\M@extsymbols@id}{"220B}
3304 \M@sym@\mp} {\mathord}{\M@extsymbols@id}{"2213}
3305 \M@sym@\angle} {\mathord}{\M@extsymbols@id}{"2220}
3306 \M@sym@\top} {\mathord}{\M@extsymbols@id}{"22A4}
3307 \M@sym@\bot} {\mathord}{\M@extsymbols@id}{"22A5}
3308 \M@sym@\vdash} {\mathord}{\M@extsymbols@id}{"22A2}
3309 \M@sym@\dashv} {\mathord}{\M@extsymbols@id}{"22A3}
3310 \M@sym@\flat} {\mathord}{\M@extsymbols@id}{"266D}
3311 \M@sym@\natural} {\mathord}{\M@extsymbols@id}{"266E}
3312 \M@sym@\sharp} {\mathord}{\M@extsymbols@id}{"266F}
3313 \M@sym@\fflat} {\mathord}{\M@extsymbols@id}{"1D12B}
3314 \M@sym@\ssharp} {\mathord}{\M@extsymbols@id}{"1D12A}
3315 \M@sym@\bclubsuit} {\mathord}{\M@extsymbols@id}{"2663}

```


3316	<code>\M@sym@{\bdiamondsuit}</code>	<code>{\mathord}{\M@extsymbols@id}{ "2666}</code>
3317	<code>\M@sym@{\bheartsuit}</code>	<code>{\mathord}{\M@extsymbols@id}{ "2665}</code>
3318	<code>\M@sym@{\bspadesuit}</code>	<code>{\mathord}{\M@extsymbols@id}{ "2660}</code>
3319	<code>\M@sym@{\wclubsuit}</code>	<code>{\mathord}{\M@extsymbols@id}{ "2667}</code>
3320	<code>\M@sym@{\wdiamondsuit}</code>	<code>{\mathord}{\M@extsymbols@id}{ "2662}</code>
3321	<code>\M@sym@{\wheartsuit}</code>	<code>{\mathord}{\M@extsymbols@id}{ "2661}</code>
3322	<code>\M@sym@{\wspadesuit}</code>	<code>{\mathord}{\M@extsymbols@id}{ "2664}</code>
3323	<code>\let\spadesuit\bspadesuit</code>	
3324	<code>\let\heartsuit\wheartsuit</code>	
3325	<code>\let\diamondsuit\wdiamondsuit</code>	
3326	<code>\let\clubsuit\bclubsuit</code>	
3327	<code>\M@sym@{\wedge}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "2227}</code>
3328	<code>\M@sym@{\vee}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "2228}</code>
3329	<code>\M@sym@{\cap}</code>	<code>{\mathord}{\M@extsymbols@id}{ "2229}</code>
3330	<code>\M@sym@{\cup}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "222A}</code>
3331	<code>\M@sym@{\sqcap}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "2293}</code>
3332	<code>\M@sym@{\sqcup}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "2294}</code>
3333	<code>\M@sym@{\amalg}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "2A3F}</code>
3334	<code>\M@sym@{\wr}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "2240}</code>
3335	<code>\M@sym@{\ast}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "2217}</code>
3336	<code>\M@sym@{\star}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "22C6}</code>
3337	<code>\M@sym@{\diamond}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "22C4}</code>
3338	<code>\M@sym@{\varcdot}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "22C5}</code>
3339	<code>\M@sym@{\varsetminus}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "2216}</code>
3340	<code>\M@sym@{\oplus}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "2295}</code>
3341	<code>\M@sym@{\otimes}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "2297}</code>
3342	<code>\M@sym@{\ominus}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "2296}</code>
3343	<code>\M@sym@{\odiv}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "2A38}</code>
3344	<code>\M@sym@{\oslash}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "2298}</code>
3345	<code>\M@sym@{\odot}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "2299}</code>
3346	<code>\M@sym@{\sqplus}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "229E}</code>
3347	<code>\M@sym@{\sqtimes}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "22A0}</code>
3348	<code>\M@sym@{\sqminus}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "229F}</code>
3349	<code>\M@sym@{\sqdot}</code>	<code>{\mathbin}{\M@extsymbols@id}{ "22A1}</code>
3350	<code>\M@sym@{\in}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2208}</code>
3351	<code>\M@sym@{\ni}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "220B}</code>
3352	<code>\M@sym@{\subset}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2282}</code>
3353	<code>\M@sym@{\supset}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2283}</code>
3354	<code>\M@sym@{\subteq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2286}</code>
3355	<code>\M@sym@{\supseteq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2287}</code>
3356	<code>\M@sym@{\sqsubset}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "228F}</code>
3357	<code>\M@sym@{\sqsupset}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2290}</code>

3358	<code>\M@sym@\sqsubseteq</code>	<code>{\mathrel}{\M@extsymbols@id}{"2291}</code>
3359	<code>\M@sym@\sqsupseteq</code>	<code>{\mathrel}{\M@extsymbols@id}{"2292}</code>
3360	<code>\M@sym@\triangleleft</code>	<code>{\mathrel}{\M@extsymbols@id}{"22B2}</code>
3361	<code>\M@sym@\triangleright</code>	<code>{\mathrel}{\M@extsymbols@id}{"22B3}</code>
3362	<code>\M@sym@\trianglelefteq</code>	<code>{\mathrel}{\M@extsymbols@id}{"22B4}</code>
3363	<code>\M@sym@\trianglerighteq</code>	<code>{\mathrel}{\M@extsymbols@id}{"22B5}</code>
3364	<code>\M@sym@\propto</code>	<code>{\mathrel}{\M@extsymbols@id}{"221D}</code>
3365	<code>\M@sym@\bowtie</code>	<code>{\mathrel}{\M@extsymbols@id}{"22C8}</code>
3366	<code>\M@sym@\hourglass</code>	<code>{\mathrel}{\M@extsymbols@id}{"29D6}</code>
3367	<code>\M@sym@\therefore</code>	<code>{\mathrel}{\M@extsymbols@id}{"2234}</code>
3368	<code>\M@sym@\because</code>	<code>{\mathrel}{\M@extsymbols@id}{"2235}</code>
3369	<code>\M@sym@\ratio</code>	<code>{\mathrel}{\M@extsymbols@id}{"2236}</code>
3370	<code>\M@sym@\proportion</code>	<code>{\mathrel}{\M@extsymbols@id}{"2237}</code>
3371	<code>\M@sym@\ll</code>	<code>{\mathrel}{\M@extsymbols@id}{"226A}</code>
3372	<code>\M@sym@\gg</code>	<code>{\mathrel}{\M@extsymbols@id}{"226B}</code>
3373	<code>\M@sym@\lll</code>	<code>{\mathrel}{\M@extsymbols@id}{"22D8}</code>
3374	<code>\M@sym@\ggg</code>	<code>{\mathrel}{\M@extsymbols@id}{"22D9}</code>
3375	<code>\M@sym@\leqq</code>	<code>{\mathrel}{\M@extsymbols@id}{"2266}</code>
3376	<code>\M@sym@\geqq</code>	<code>{\mathrel}{\M@extsymbols@id}{"2267}</code>
3377	<code>\M@sym@\lapprox</code>	<code>{\mathrel}{\M@extsymbols@id}{"2A85}</code>
3378	<code>\M@sym@\gapprox</code>	<code>{\mathrel}{\M@extsymbols@id}{"2A86}</code>
3379	<code>\M@sym@\simeq</code>	<code>{\mathrel}{\M@extsymbols@id}{"2243}</code>
3380	<code>\M@sym@\eqsim</code>	<code>{\mathrel}{\M@extsymbols@id}{"2242}</code>
3381	<code>\M@sym@\simeqq</code>	<code>{\mathrel}{\M@extsymbols@id}{"2245}</code>
3382	<code>\let\cong\simeqq</code>	
3383	<code>\M@sym@\approxeq</code>	<code>{\mathrel}{\M@extsymbols@id}{"224A}</code>
3384	<code>\M@sym@\sssim</code>	<code>{\mathrel}{\M@extsymbols@id}{"224B}</code>
3385	<code>\M@sym@\seq</code>	<code>{\mathrel}{\M@extsymbols@id}{"224C}</code>
3386	<code>\M@sym@\doteq</code>	<code>{\mathrel}{\M@extsymbols@id}{"2250}</code>
3387	<code>\M@sym@\coloneq</code>	<code>{\mathrel}{\M@extsymbols@id}{"2254}</code>
3388	<code>\M@sym@\eqcolon</code>	<code>{\mathrel}{\M@extsymbols@id}{"2255}</code>
3389	<code>\M@sym@\ringeq</code>	<code>{\mathrel}{\M@extsymbols@id}{"2257}</code>
3390	<code>\M@sym@\arceq</code>	<code>{\mathrel}{\M@extsymbols@id}{"2258}</code>
3391	<code>\M@sym@\wedgieq</code>	<code>{\mathrel}{\M@extsymbols@id}{"2259}</code>
3392	<code>\M@sym@\veeeq</code>	<code>{\mathrel}{\M@extsymbols@id}{"225A}</code>
3393	<code>\M@sym@\stareq</code>	<code>{\mathrel}{\M@extsymbols@id}{"225B}</code>
3394	<code>\M@sym@\triangleeq</code>	<code>{\mathrel}{\M@extsymbols@id}{"225C}</code>
3395	<code>\M@sym@\defeq</code>	<code>{\mathrel}{\M@extsymbols@id}{"225D}</code>
3396	<code>\M@sym@\qeq</code>	<code>{\mathrel}{\M@extsymbols@id}{"225F}</code>
3397	<code>\M@sym@\lsim</code>	<code>{\mathrel}{\M@extsymbols@id}{"2272}</code>
3398	<code>\M@sym@\gsim</code>	<code>{\mathrel}{\M@extsymbols@id}{"2273}</code>
3399	<code>\M@sym@\prec</code>	<code>{\mathrel}{\M@extsymbols@id}{"227A}</code>

3400	<code>\M@sym@{\succ}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "227B}</code>
3401	<code>\M@sym@{\preceq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "227C}</code>
3402	<code>\M@sym@{\succeq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "227D}</code>
3403	<code>\M@sym@{\preceqq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2AB3}</code>
3404	<code>\M@sym@{\succeqq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2AB4}</code>
3405	<code>\M@sym@{\precsim}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "227E}</code>
3406	<code>\M@sym@{\succsim}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "227F}</code>
3407	<code>\M@sym@{\precapprox}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2AB7}</code>
3408	<code>\M@sym@{\succapprox}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2AB8}</code>
3409	<code>\M@sym@{\precprec}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2ABB}</code>
3410	<code>\M@sym@{\succsucc}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2ABC}</code>
3411	<code>\M@sym@{\asymp}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "224D}</code>
3412	<code>\M@sym@{\nin}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2209}</code>
3413	<code>\M@sym@{\nni}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "220C}</code>
3414	<code>\M@sym@{\nsubset}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2284}</code>
3415	<code>\M@sym@{\nsupset}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2285}</code>
3416	<code>\M@sym@{\nsubseteq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2288}</code>
3417	<code>\M@sym@{\nsupseteq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2289}</code>
3418	<code>\M@sym@{\subsetneq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "228A}</code>
3419	<code>\M@sym@{\supsetneq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "228B}</code>
3420	<code>\M@sym@{\nsubseteqq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "22E2}</code>
3421	<code>\M@sym@{\nsupseteqq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "22E3}</code>
3422	<code>\M@sym@{\sqsubseteqneq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "22E4}</code>
3423	<code>\M@sym@{\sqsupsetneq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "22E5}</code>
3424	<code>\M@sym@{\neq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2260}</code>
3425	<code>\M@sym@{\nleq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "226E}</code>
3426	<code>\M@sym@{\nleqq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2270}</code>
3427	<code>\M@sym@{\ngeq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2271}</code>
3428	<code>\M@sym@{\lneq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2A87}</code>
3429	<code>\M@sym@{\gneq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2A88}</code>
3430	<code>\M@sym@{\lneqq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2268}</code>
3431	<code>\M@sym@{\gneqq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2269}</code>
3432	<code>\M@sym@{\ntriangleleft}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "22EA}</code>
3433	<code>\M@sym@{\ntriangleright}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "22EB}</code>
3434	<code>\M@sym@{\ntrianglelefteq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "22EC}</code>
3435	<code>\M@sym@{\ntrianglerighteq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "22ED}</code>
3436	<code>\M@sym@{\nsim}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2241}</code>
3437	<code>\M@sym@{\napprox}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2249}</code>
3438	<code>\M@sym@{\nsimeq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2244}</code>
3439	<code>\M@sym@{\nsimeqq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2247}</code>
3440	<code>\M@sym@{\simneqq}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2246}</code>
3441	<code>\M@sym@{\nlsim}</code>	<code>{\mathrel}{\M@extsymbols@id}{ "2274}</code>

```

3442 \M@sym@\ngsim}          {\mathrel}{\M@extsymbols@id}{"2275}
3443 \M@sym@\lnsim}          {\mathrel}{\M@extsymbols@id}{"22E6}
3444 \M@sym@\gnsim}          {\mathrel}{\M@extsymbols@id}{"22E7}
3445 \M@sym@\lnapprox}       {\mathrel}{\M@extsymbols@id}{"2A89}
3446 \M@sym@\gnapprox}       {\mathrel}{\M@extsymbols@id}{"2A8A}
3447 \M@sym@\nprec}          {\mathrel}{\M@extsymbols@id}{"2280}
3448 \M@sym@\nsucc}          {\mathrel}{\M@extsymbols@id}{"2281}
3449 \M@sym@\npreceq}        {\mathrel}{\M@extsymbols@id}{"22E0}
3450 \M@sym@\nsucceq}        {\mathrel}{\M@extsymbols@id}{"22E1}
3451 \M@sym@\precneq}        {\mathrel}{\M@extsymbols@id}{"2AB1}
3452 \M@sym@\succneq}        {\mathrel}{\M@extsymbols@id}{"2AB2}
3453 \M@sym@\precneqq}       {\mathrel}{\M@extsymbols@id}{"2AB5}
3454 \M@sym@\succneqq}       {\mathrel}{\M@extsymbols@id}{"2AB6}
3455 \M@sym@\precnsim}       {\mathrel}{\M@extsymbols@id}{"22E8}
3456 \M@sym@\succnsim}       {\mathrel}{\M@extsymbols@id}{"22E9}
3457 \M@sym@\precnapprox}    {\mathrel}{\M@extsymbols@id}{"2AB9}
3458 \M@sym@\succnapprox}    {\mathrel}{\M@extsymbols@id}{"2ABA}
3459 \M@sym@\nequiv}        {\mathrel}{\M@extsymbols@id}{"2262}

```

The `math-operator` package renames `\Re` and `\Im` to `\varRe` and `\varIm`. To make `mathfont` compatible with that package, we test whether these macros contain `mathchar` in their definitions before redefining. First `\Re`.

```

3460 \expanded{\noexpand\in@{\expandafter\@gobble\string\mathchar}
3461   {\meaning\Re}}
3462 \ifin@
3463   \M@sym@\Re}{\mathord}{\M@extsymbols@id}{"211C}
3464 \else
3465   \expanded{\noexpand\in@{\expandafter\@gobble\string\mathchar}
3466     {\meaning\varRe}}
3467   \ifin@
3468     \M@sym@\varRe}{\mathord}{\M@extsymbols@id}{"211C}
3469   \fi
3470 \fi

```

And `\Im`.

```

3471 \expanded{\noexpand\in@{\expandafter\@gobble\string\mathchar}
3472   {\meaning\Im}}
3473 \ifin@
3474   \M@sym@\Im}{\mathord}{\M@extsymbols@id}{"2111}
3475 \else
3476   \expanded{\noexpand\in@{\expandafter\@gobble\string\mathchar}
3477     {\meaning\varIm}}
3478   \ifin@

```

```

3479      \M@sym@{\varIm}{\mathord}{\M@extsymbols@id}{2111}
3480      \fi
3481      \fi

```

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

```

3482      \let\textng\ng
3483      \M@sym@{\mathng}{\mathrel}{\M@extsymbols@id}{226F}
3484      \protected\def\ng{\ifmmode\mathng\else\textng\fi}

```

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

```

3485      \ifM@adjust@font\else
3486      \M@sym@{\nabla}{\mathord}{\M@extsymbols@id}{2207}
3487      \fi}

```

Set arrows.

```

\M@arrows@set 3488 \def\M@arrows@set{
3489   \edef\M@arrows@id{M\@tempc-\M@arrowsshape}
3490   \let\uparrow\@undefined
3491   \let\Uparrow\@undefined
3492   \let\downarrow\@undefined
3493   \let\Downarrow\@undefined
3494   \let\updownarrow\@undefined
3495   \let\Updownarrow\@undefined
3496   \let\longrightarrow\@undefined
3497   \let\longleftarrow\@undefined
3498   \let\longleftarrow\@undefined
3499   \let\hookrightarrow\@undefined
3500   \let\hookleftarrow\@undefined
3501   \let\Longrightarrow\@undefined
3502   \let\Longleftarrow\@undefined
3503   \let\Longleftarrow\@undefined
3504   \let\rightleftharpoons\@undefined
3505   \M@sym@{\rightarrow}{\mathrel}{\M@arrows@id}{2192}
3506   \let\to\rightarrow
3507   \M@sym@{\nrightarrow}{\mathrel}{\M@arrows@id}{219B}
3508   \M@sym@{\Rrightarrow}{\mathrel}{\M@arrows@id}{21D2}
3509   \M@sym@{\nRrightarrow}{\mathrel}{\M@arrows@id}{21CF}
3510   \M@sym@{\Rrightarrow}{\mathrel}{\M@arrows@id}{21DB}
3511   \M@sym@{\longrightarrow}{\mathrel}{\M@arrows@id}{27F6}
3512   \M@sym@{\Longrightarrow}{\mathrel}{\M@arrows@id}{27F9}
3513   \M@sym@{\rightbararrow}{\mathrel}{\M@arrows@id}{21A6}
3514   \let\mapsto\rightbararrow
3515   \M@sym@{\Rrightarrow}{\mathrel}{\M@arrows@id}{2907}

```

3516	<code>\M@sym@{\longrightbararrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "27FC}</code>
3517	<code>\let\longmapsto\longrightbararrow</code>	
3518	<code>\M@sym@{\Longrightbararrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "27FE}</code>
3519	<code>\M@sym@{\hookrightarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21AA}</code>
3520	<code>\M@sym@{\rightrightarrows}</code>	<code>{\mathrel}{\M@arrows@id}{ "21E2}</code>
3521	<code>\M@sym@{\rightharpoonup}</code>	<code>{\mathrel}{\M@arrows@id}{ "21C0}</code>
3522	<code>\M@sym@{\rightharpoondown}</code>	<code>{\mathrel}{\M@arrows@id}{ "21C1}</code>
3523	<code>\M@sym@{\rightarrowtail}</code>	<code>{\mathrel}{\M@arrows@id}{ "21A3}</code>
3524	<code>\M@sym@{\rightarrowtail}</code>	<code>{\mathrel}{\M@arrows@id}{ "27F4}</code>
3525	<code>\M@sym@{\rightarrowtail}</code>	<code>{\mathrel}{\M@arrows@id}{ "219D}</code>
3526	<code>\M@sym@{\rightsquigarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21DD}</code>
3527	<code>\M@sym@{\longrightsquigarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "27FF}</code>
3528	<code>\M@sym@{\looparrowright}</code>	<code>{\mathrel}{\M@arrows@id}{ "21AC}</code>
3529	<code>\M@sym@{\curvearrowright}</code>	<code>{\mathrel}{\M@arrows@id}{ "293B}</code>
3530	<code>\M@sym@{\circlearrowright}</code>	<code>{\mathrel}{\M@arrows@id}{ "21BB}</code>
3531	<code>\M@sym@{\twoheadrightarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21A0}</code>
3532	<code>\M@sym@{\rightarrowtail}</code>	<code>{\mathrel}{\M@arrows@id}{ "21E5}</code>
3533	<code>\M@sym@{\rightarrowtail}</code>	<code>{\mathrel}{\M@arrows@id}{ "21E8}</code>
3534	<code>\M@sym@{\rightarrowtail}</code>	<code>{\mathrel}{\M@arrows@id}{ "21C9}</code>
3535	<code>\M@sym@{\rightarrowtail}</code>	<code>{\mathrel}{\M@arrows@id}{ "21F6}</code>
3536	<code>\M@sym@{\leftarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "2190}</code>
3537	<code>\let\from\leftarrow</code>	
3538	<code>\M@sym@{\nleftarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "219A}</code>
3539	<code>\M@sym@{\Leftarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21D0}</code>
3540	<code>\M@sym@{\nLeftarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21CD}</code>
3541	<code>\M@sym@{\lleftarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21DA}</code>
3542	<code>\M@sym@{\longleftarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "27F5}</code>
3543	<code>\M@sym@{\Longleftarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "27F8}</code>
3544	<code>\M@sym@{\leftbararrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21A4}</code>
3545	<code>\let\mapsfrom\leftbararrow</code>	
3546	<code>\M@sym@{\Leftbararrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "2906}</code>
3547	<code>\M@sym@{\longleftbararrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "27FB}</code>
3548	<code>\let\longmapsfrom\longleftbararrow</code>	
3549	<code>\M@sym@{\Longleftbararrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "27FD}</code>
3550	<code>\M@sym@{\hookleftarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21A9}</code>
3551	<code>\M@sym@{\leftrightharrows}</code>	<code>{\mathrel}{\M@arrows@id}{ "21E0}</code>
3552	<code>\M@sym@{\leftharpoonup}</code>	<code>{\mathrel}{\M@arrows@id}{ "21BC}</code>
3553	<code>\M@sym@{\leftharpoondown}</code>	<code>{\mathrel}{\M@arrows@id}{ "21BD}</code>
3554	<code>\M@sym@{\leftarrowtail}</code>	<code>{\mathrel}{\M@arrows@id}{ "21A2}</code>
3555	<code>\M@sym@{\leftoplusarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "2B32}</code>
3556	<code>\M@sym@{\leftwavyarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "219C}</code>
3557	<code>\M@sym@{\leftsquigarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21DC}</code>

3558	<code>\M@sym@{\longleftsquigarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "2B33}</code>
3559	<code>\M@sym@{\looparrowleft}</code>	<code>{\mathrel}{\M@arrows@id}{ "21AB}</code>
3560	<code>\M@sym@{\curvearrowleft}</code>	<code>{\mathrel}{\M@arrows@id}{ "293A}</code>
3561	<code>\M@sym@{\circlearrowleft}</code>	<code>{\mathrel}{\M@arrows@id}{ "21BA}</code>
3562	<code>\M@sym@{\twoheadleftarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "219E}</code>
3563	<code>\M@sym@{\leftarrowto}</code>	<code>{\mathrel}{\M@arrows@id}{ "21E4}</code>
3564	<code>\M@sym@{\leftwhitearrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21E6}</code>
3565	<code>\M@sym@{\leftleftarrows}</code>	<code>{\mathrel}{\M@arrows@id}{ "21C7}</code>
3566	<code>\M@sym@{\leftleftleftarrows}</code>	<code>{\mathrel}{\M@arrows@id}{ "2B31}</code>
3567	<code>\M@sym@{\leftrightarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "2194}</code>
3568	<code>\M@sym@{\Leftrightarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21D4}</code>
3569	<code>\M@sym@{\nLeftrightarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21CE}</code>
3570	<code>\M@sym@{\longletrightarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "27F7}</code>
3571	<code>\M@sym@{\Longletrightarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "27FA}</code>
3572	<code>\M@sym@{\leftrightharpoonup}</code>	<code>{\mathrel}{\M@arrows@id}{ "21AD}</code>
3573	<code>\M@sym@{\leftrightharpoons}</code>	<code>{\mathrel}{\M@arrows@id}{ "21C6}</code>
3574	<code>\M@sym@{\leftrightharpoons}</code>	<code>{\mathrel}{\M@arrows@id}{ "21CB}</code>
3575	<code>\M@sym@{\leftrightharpoonup}</code>	<code>{\mathrel}{\M@arrows@id}{ "21B9}</code>
3576	<code>\M@sym@{\rightleftarrows}</code>	<code>{\mathrel}{\M@arrows@id}{ "21C4}</code>
3577	<code>\M@sym@{\rightleftharpoons}</code>	<code>{\mathrel}{\M@arrows@id}{ "21CC}</code>
3578	<code>\M@sym@{\uparrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "2191}</code>
3579	<code>\M@sym@{\Uparrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21D1}</code>
3580	<code>\M@sym@{\Uparrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "290A}</code>
3581	<code>\M@sym@{\upbararrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21A5}</code>
3582	<code>\M@sym@{\updasharrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21E1}</code>
3583	<code>\M@sym@{\upharpoonleft}</code>	<code>{\mathrel}{\M@arrows@id}{ "21BF}</code>
3584	<code>\M@sym@{\upharpoonright}</code>	<code>{\mathrel}{\M@arrows@id}{ "21BE}</code>
3585	<code>\M@sym@{\twoheaduparrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "219F}</code>
3586	<code>\M@sym@{\uparrowto}</code>	<code>{\mathrel}{\M@arrows@id}{ "2912}</code>
3587	<code>\M@sym@{\upwhitearrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21E7}</code>
3588	<code>\M@sym@{\upwhitebararrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21EA}</code>
3589	<code>\M@sym@{\upuparrows}</code>	<code>{\mathrel}{\M@arrows@id}{ "21C8}</code>
3590	<code>\M@sym@{\downarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "2193}</code>
3591	<code>\M@sym@{\Downarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21D3}</code>
3592	<code>\M@sym@{\Ddownarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "290B}</code>
3593	<code>\M@sym@{\downbararrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21A7}</code>
3594	<code>\M@sym@{\downdasharrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21E3}</code>
3595	<code>\M@sym@{\zigzagarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21AF}</code>
3596	<code>\let\lightningboltarrow\zigzagarrow</code>	
3597	<code>\M@sym@{\downharpoonleft}</code>	<code>{\mathrel}{\M@arrows@id}{ "21C3}</code>
3598	<code>\M@sym@{\downharpoonright}</code>	<code>{\mathrel}{\M@arrows@id}{ "21C2}</code>
3599	<code>\M@sym@{\twoheaddownarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21A1}</code>

3600	<code>\M@sym@{\downarrowto bar}</code>	<code>{\mathrel}{\M@arrows@id}{ "2913}</code>
3601	<code>\M@sym@{\downwhite arrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21E9}</code>
3602	<code>\M@sym@{\down downarrows}</code>	<code>{\mathrel}{\M@arrows@id}{ "21CA}</code>
3603	<code>\M@sym@{\up downarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "2195}</code>
3604	<code>\M@sym@{\Up downarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21D5}</code>
3605	<code>\M@sym@{\up downarrows}</code>	<code>{\mathrel}{\M@arrows@id}{ "21C5}</code>
3606	<code>\M@sym@{\down uparrows}</code>	<code>{\mathrel}{\M@arrows@id}{ "21F5}</code>
3607	<code>\M@sym@{\up downharpoons}</code>	<code>{\mathrel}{\M@arrows@id}{ "296E}</code>
3608	<code>\M@sym@{\down upharpoons}</code>	<code>{\mathrel}{\M@arrows@id}{ "296F}</code>
3609	<code>\M@sym@{\nearrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "2197}</code>
3610	<code>\M@sym@{\Nearrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21D7}</code>
3611	<code>\M@sym@{\nwarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "2196}</code>
3612	<code>\M@sym@{\Nwarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21D6}</code>
3613	<code>\M@sym@{\searrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "2198}</code>
3614	<code>\M@sym@{\Searrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21D8}</code>
3615	<code>\M@sym@{\swarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "2199}</code>
3616	<code>\M@sym@{\Swarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "21D9}</code>
3617	<code>\M@sym@{\nwsearrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "2921}</code>
3618	<code>\M@sym@{\neswarrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "2922}</code>
3619	<code>\M@sym@{\lrcirclearrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "27F2}</code>
3620	<code>\M@sym@{\rcirclearrow}</code>	<code>{\mathrel}{\M@arrows@id}{ "27F3}</code>

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_YT_EX and LuaT_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.

```

3621 \let\@relbar\@undefined
3622 \let\@Relbar\@undefined
3623 \M@sym@{\@relbar}{\mathbin}{\M@arrows@id}{ "2212}
3624 \M@sym@{\@Relbar}{\mathrel}{\M@arrows@id}{ "3D}
3625 \protected\def\relbar{\mathrel{\mathpalette\mathsm@sh\@relbar}}
3626 \protected\def\Relbar{\@Relbar}

```

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

```

3627 \ifpackageloaded{amsmath}
3628   {\relax}{
3629     \let\@@relbar\relbar
3630     \let\@@Relbar\Relbar

```



```

3631      \AtBeginDocument{\@ifpackageloaded{amsmath}{
3632          \let\relbar\@@relbar
3633          \let\Relbar\@@Relbar}
3634          {\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 3635 \def\M@bb@set{
  \mathbb 3636 \protected\def\mathbb##1{\relax
    3637   \ifmmode\else
    3638     \M@HModeError\mathbb
    3639     $
    3640     \fi
    3641     \begingroup
    3642       \M@bb@mathcodes
    3643       ##1
    3644     \endgroup}
\M@bb@num 3645 \edef\M@bb@num{\number
  3646   \csname symM\@tempc-\M@bbshape\endcsname}
\M@bb@mathcodes 3647 \protected\edef\M@bb@mathcodes{
  3648   \Umathcode`A=0+\M@bb@num"1D538\relax
  3649   \Umathcode`B=0+\M@bb@num"1D539\relax
  3650   \Umathcode`C=0+\M@bb@num"2102\relax
  3651   \Umathcode`D=0+\M@bb@num"1D53B\relax
  3652   \Umathcode`E=0+\M@bb@num"1D53C\relax
  3653   \Umathcode`F=0+\M@bb@num"1D53D\relax
  3654   \Umathcode`G=0+\M@bb@num"1D53E\relax
  3655   \Umathcode`H=0+\M@bb@num"210D\relax
  3656   \Umathcode`I=0+\M@bb@num"1D540\relax
  3657   \Umathcode`J=0+\M@bb@num"1D541\relax
  3658   \Umathcode`K=0+\M@bb@num"1D542\relax
  3659   \Umathcode`L=0+\M@bb@num"1D543\relax
  3660   \Umathcode`M=0+\M@bb@num"1D544\relax
  3661   \Umathcode`N=0+\M@bb@num"2115\relax
  3662   \Umathcode`O=0+\M@bb@num"1D546\relax

```

```

3663 \Umathcode`P=0+\M@bb@num"2119\relax
3664 \Umathcode`Q=0+\M@bb@num"211A\relax
3665 \Umathcode`R=0+\M@bb@num"211D\relax
3666 \Umathcode`S=0+\M@bb@num"1D54A\relax
3667 \Umathcode`T=0+\M@bb@num"1D54B\relax
3668 \Umathcode`U=0+\M@bb@num"1D54C\relax
3669 \Umathcode`V=0+\M@bb@num"1D54D\relax
3670 \Umathcode`W=0+\M@bb@num"1D54E\relax
3671 \Umathcode`X=0+\M@bb@num"1D54F\relax
3672 \Umathcode`Y=0+\M@bb@num"1D550\relax
3673 \Umathcode`Z=0+\M@bb@num"2124\relax
3674 \Umathcode`a=0+\M@bb@num"1D552\relax
3675 \Umathcode`b=0+\M@bb@num"1D553\relax
3676 \Umathcode`c=0+\M@bb@num"1D554\relax
3677 \Umathcode`d=0+\M@bb@num"1D555\relax
3678 \Umathcode`e=0+\M@bb@num"1D556\relax
3679 \Umathcode`f=0+\M@bb@num"1D557\relax
3680 \Umathcode`g=0+\M@bb@num"1D558\relax
3681 \Umathcode`h=0+\M@bb@num"1D559\relax
3682 \Umathcode`i=0+\M@bb@num"1D55A\relax
3683 \Umathcode`j=0+\M@bb@num"1D55B\relax
3684 \Umathcode`k=0+\M@bb@num"1D55C\relax
3685 \Umathcode`l=0+\M@bb@num"1D55D\relax
3686 \Umathcode`m=0+\M@bb@num"1D55E\relax
3687 \Umathcode`n=0+\M@bb@num"1D55F\relax
3688 \Umathcode`o=0+\M@bb@num"1D560\relax
3689 \Umathcode`p=0+\M@bb@num"1D561\relax
3690 \Umathcode`q=0+\M@bb@num"1D562\relax
3691 \Umathcode`r=0+\M@bb@num"1D563\relax
3692 \Umathcode`s=0+\M@bb@num"1D564\relax
3693 \Umathcode`t=0+\M@bb@num"1D565\relax
3694 \Umathcode`u=0+\M@bb@num"1D566\relax
3695 \Umathcode`v=0+\M@bb@num"1D567\relax
3696 \Umathcode`w=0+\M@bb@num"1D568\relax
3697 \Umathcode`x=0+\M@bb@num"1D569\relax
3698 \Umathcode`y=0+\M@bb@num"1D56A\relax
3699 \Umathcode`z=0+\M@bb@num"1D56B\relax
3700 \Umathcode`0=0+\M@bb@num"1D7D8\relax
3701 \Umathcode`1=0+\M@bb@num"1D7D9\relax
3702 \Umathcode`2=0+\M@bb@num"1D7DA\relax
3703 \Umathcode`3=0+\M@bb@num"1D7DB\relax
3704 \Umathcode`4=0+\M@bb@num"1D7DC\relax

```

```

3705    \Umathcode`5=0+\M@bb@num"1D7DD\relax
3706    \Umathcode`6=0+\M@bb@num"1D7DE\relax
3707    \Umathcode`7=0+\M@bb@num"1D7DF\relax
3708    \Umathcode`8=0+\M@bb@num"1D7E0\relax
3709    \Umathcode`9=0+\M@bb@num"1D7E1\relax}}

```

Set caligraphic letters.

```

\M@cal@set 3710 \def\M@cal@set{
  \mathcal 3711 \protected\def\mathcal##1{\relax
    3712 \ifmmode\else
    3713 \M@HModeError\mathcal
    3714 $
    3715 \fi
    3716 \begingroup
    3717 \M@cal@mathcodes
    3718 ##1
    3719 \endgroup}
\M@cal@num 3720 \edef\M@cal@num{\number
  3721 \csname symM\@tempc-\M@calshape\endcsname}
\M@cal@mathcodes 3722 \protected\edef\M@cal@mathcodes{
  3723 \Umathcode`A=0+\M@cal@num"1D49C\relax
  3724 \Umathcode`B=0+\M@cal@num"212C\relax
  3725 \Umathcode`C=0+\M@cal@num"1D49E\relax
  3726 \Umathcode`D=0+\M@cal@num"1D49F\relax
  3727 \Umathcode`E=0+\M@cal@num"2130\relax
  3728 \Umathcode`F=0+\M@cal@num"2131\relax
  3729 \Umathcode`G=0+\M@cal@num"1D4A2\relax
  3730 \Umathcode`H=0+\M@cal@num"210B\relax
  3731 \Umathcode`I=0+\M@cal@num"2110\relax
  3732 \Umathcode`J=0+\M@cal@num"1D4A5\relax
  3733 \Umathcode`K=0+\M@cal@num"1D4A6\relax
  3734 \Umathcode`L=0+\M@cal@num"2112\relax
  3735 \Umathcode`M=0+\M@cal@num"2133\relax
  3736 \Umathcode`N=0+\M@cal@num"1D4A9\relax
  3737 \Umathcode`O=0+\M@cal@num"1D4AA\relax
  3738 \Umathcode`P=0+\M@cal@num"1D4AB\relax
  3739 \Umathcode`Q=0+\M@cal@num"1D4AC\relax
  3740 \Umathcode`R=0+\M@cal@num"211B\relax
  3741 \Umathcode`S=0+\M@cal@num"1D4AE\relax
  3742 \Umathcode`T=0+\M@cal@num"1D4AF\relax
  3743 \Umathcode`U=0+\M@cal@num"1D4B0\relax
  3744 \Umathcode`V=0+\M@cal@num"1D4B1\relax
  3745 \Umathcode`W=0+\M@cal@num"1D4B2\relax

```

```

3746 \Umathcode`X=0+\M@cal@num"1D4B3\relax
3747 \Umathcode`Y=0+\M@cal@num"1D4B4\relax
3748 \Umathcode`Z=0+\M@cal@num"1D4B5\relax
3749 \Umathcode`a=0+\M@cal@num"1D4B6\relax
3750 \Umathcode`b=0+\M@cal@num"1D4B7\relax
3751 \Umathcode`c=0+\M@cal@num"1D4B8\relax
3752 \Umathcode`d=0+\M@cal@num"1D4B9\relax
3753 \Umathcode`e=0+\M@cal@num"212F\relax
3754 \Umathcode`f=0+\M@cal@num"1D4BB\relax
3755 \Umathcode`g=0+\M@cal@num"210A\relax
3756 \Umathcode`h=0+\M@cal@num"1D4BD\relax
3757 \Umathcode`i=0+\M@cal@num"1D4BE\relax
3758 \Umathcode`j=0+\M@cal@num"1D4BF\relax
3759 \Umathcode`k=0+\M@cal@num"1D4C0\relax
3760 \Umathcode`l=0+\M@cal@num"1D4C1\relax
3761 \Umathcode`m=0+\M@cal@num"1D4C2\relax
3762 \Umathcode`n=0+\M@cal@num"1D4C3\relax
3763 \Umathcode`o=0+\M@cal@num"2134\relax
3764 \Umathcode`p=0+\M@cal@num"1D4C5\relax
3765 \Umathcode`q=0+\M@cal@num"1D4C6\relax
3766 \Umathcode`r=0+\M@cal@num"1D4C7\relax
3767 \Umathcode`s=0+\M@cal@num"1D4C8\relax
3768 \Umathcode`t=0+\M@cal@num"1D4C9\relax
3769 \Umathcode`u=0+\M@cal@num"1D4CA\relax
3770 \Umathcode`v=0+\M@cal@num"1D4CB\relax
3771 \Umathcode`w=0+\M@cal@num"1D4CC\relax
3772 \Umathcode`x=0+\M@cal@num"1D4CD\relax
3773 \Umathcode`y=0+\M@cal@num"1D4CE\relax
3774 \Umathcode`z=0+\M@cal@num"1D4CF\relax}}

```

Set fraktur letters.

```

\M@frak@set 3775 \def\M@frak@set{
  \mathfrak 3776 \protected\def\mathfrak##1{\relax
    3777 \ifmmode\else
    3778 \M@HModeError\mathfrak
    3779 $
    3780 \fi
    3781 \begingroup
    3782 \M@frak@mathcodes
    3783 ##1
    3784 \endgroup}
\M@frak@num 3785 \edef\M@frak@num{\number
    3786 \csname symM\@tempc-\M@frakshape\endcsname}

```

```

\M@frak@mathcodes 3787 \protected\edef\M@frak@mathcodes{
3788   \Umathcode`A=0+\M@frak@num"1D504\relax
3789   \Umathcode`B=0+\M@frak@num"1D505\relax
3790   \Umathcode`C=0+\M@frak@num"212D\relax
3791   \Umathcode`D=0+\M@frak@num"1D507\relax
3792   \Umathcode`E=0+\M@frak@num"1D508\relax
3793   \Umathcode`F=0+\M@frak@num"1D509\relax
3794   \Umathcode`G=0+\M@frak@num"1D50A\relax
3795   \Umathcode`H=0+\M@frak@num"210C\relax
3796   \Umathcode`I=0+\M@frak@num"2111\relax
3797   \Umathcode`J=0+\M@frak@num"1D50D\relax
3798   \Umathcode`K=0+\M@frak@num"1D50E\relax
3799   \Umathcode`L=0+\M@frak@num"1D50F\relax
3800   \Umathcode`M=0+\M@frak@num"1D510\relax
3801   \Umathcode`N=0+\M@frak@num"1D511\relax
3802   \Umathcode`O=0+\M@frak@num"1D512\relax
3803   \Umathcode`P=0+\M@frak@num"1D513\relax
3804   \Umathcode`Q=0+\M@frak@num"1D514\relax
3805   \Umathcode`R=0+\M@frak@num"211C\relax
3806   \Umathcode`S=0+\M@frak@num"1D516\relax
3807   \Umathcode`T=0+\M@frak@num"1D517\relax
3808   \Umathcode`U=0+\M@frak@num"1D518\relax
3809   \Umathcode`V=0+\M@frak@num"1D519\relax
3810   \Umathcode`W=0+\M@frak@num"1D51A\relax
3811   \Umathcode`X=0+\M@frak@num"1D51B\relax
3812   \Umathcode`Y=0+\M@frak@num"1D51C\relax
3813   \Umathcode`Z=0+\M@frak@num"2128\relax
3814   \Umathcode`a=0+\M@frak@num"1D51E\relax
3815   \Umathcode`b=0+\M@frak@num"1D51F\relax
3816   \Umathcode`c=0+\M@frak@num"1D520\relax
3817   \Umathcode`d=0+\M@frak@num"1D521\relax
3818   \Umathcode`e=0+\M@frak@num"1D522\relax
3819   \Umathcode`f=0+\M@frak@num"1D523\relax
3820   \Umathcode`g=0+\M@frak@num"1D524\relax
3821   \Umathcode`h=0+\M@frak@num"1D525\relax
3822   \Umathcode`i=0+\M@frak@num"1D526\relax
3823   \Umathcode`j=0+\M@frak@num"1D527\relax
3824   \Umathcode`k=0+\M@frak@num"1D528\relax
3825   \Umathcode`l=0+\M@frak@num"1D529\relax
3826   \Umathcode`m=0+\M@frak@num"1D52A\relax
3827   \Umathcode`n=0+\M@frak@num"1D52B\relax
3828   \Umathcode`o=0+\M@frak@num"1D52C\relax

```

```

3829 \Umathcode`p=0+\M@frak@num"1D52D\relax
3830 \Umathcode`q=0+\M@frak@num"1D52E\relax
3831 \Umathcode`r=0+\M@frak@num"1D52F\relax
3832 \Umathcode`s=0+\M@frak@num"1D530\relax
3833 \Umathcode`t=0+\M@frak@num"1D531\relax
3834 \Umathcode`u=0+\M@frak@num"1D532\relax
3835 \Umathcode`v=0+\M@frak@num"1D533\relax
3836 \Umathcode`w=0+\M@frak@num"1D534\relax
3837 \Umathcode`x=0+\M@frak@num"1D535\relax
3838 \Umathcode`y=0+\M@frak@num"1D536\relax
3839 \Umathcode`z=0+\M@frak@num"1D537\relax}}

```

Set bold caligraphic letters.

```

\M@bcal@set 3840 \def\M@bcal@set{
  \mathbcal 3841 \protected\def\mathbcal##1{\relax
    3842 \ifmmode\else
    3843 \M@HModeError\mathbcal
    3844 $
    3845 \fi
    3846 \begingroup
    3847 \M@bcal@mathcodes
    3848 ##1
    3849 \endgroup}
\M@bcal@num 3850 \edef\M@bcal@num{\number
  3851 \csname symM\@tempc-\M@bcalshape\endcsname}
\M@bcal@mathcodes 3852 \protected\edef\M@bcal@mathcodes{
  3853 \Umathcode`A=0+\M@bcal@num"1D4D0\relax
  3854 \Umathcode`B=0+\M@bcal@num"1D4D1\relax
  3855 \Umathcode`C=0+\M@bcal@num"1D4D2\relax
  3856 \Umathcode`D=0+\M@bcal@num"1D4D3\relax
  3857 \Umathcode`E=0+\M@bcal@num"1D4D4\relax
  3858 \Umathcode`F=0+\M@bcal@num"1D4D5\relax
  3859 \Umathcode`G=0+\M@bcal@num"1D4D6\relax
  3860 \Umathcode`H=0+\M@bcal@num"1D4D7\relax
  3861 \Umathcode`I=0+\M@bcal@num"1D4D8\relax
  3862 \Umathcode`J=0+\M@bcal@num"1D4D9\relax
  3863 \Umathcode`K=0+\M@bcal@num"1D4DA\relax
  3864 \Umathcode`L=0+\M@bcal@num"1D4DB\relax
  3865 \Umathcode`M=0+\M@bcal@num"1D4DC\relax
  3866 \Umathcode`N=0+\M@bcal@num"1D4DD\relax
  3867 \Umathcode`O=0+\M@bcal@num"1D4DE\relax
  3868 \Umathcode`P=0+\M@bcal@num"1D4DF\relax
  3869 \Umathcode`Q=0+\M@bcal@num"1D4E0\relax

```

```

3870 \Umathcode`R=0+\M@bcal@num"1D4E1\relax
3871 \Umathcode`S=0+\M@bcal@num"1D4E2\relax
3872 \Umathcode`T=0+\M@bcal@num"1D4E3\relax
3873 \Umathcode`U=0+\M@bcal@num"1D4E4\relax
3874 \Umathcode`V=0+\M@bcal@num"1D4E5\relax
3875 \Umathcode`W=0+\M@bcal@num"1D4E6\relax
3876 \Umathcode`X=0+\M@bcal@num"1D4E7\relax
3877 \Umathcode`Y=0+\M@bcal@num"1D4E8\relax
3878 \Umathcode`Z=0+\M@bcal@num"1D4E9\relax
3879 \Umathcode`a=0+\M@bcal@num"1D4EA\relax
3880 \Umathcode`b=0+\M@bcal@num"1D4EB\relax
3881 \Umathcode`c=0+\M@bcal@num"1D4EC\relax
3882 \Umathcode`d=0+\M@bcal@num"1D4ED\relax
3883 \Umathcode`e=0+\M@bcal@num"1D4EE\relax
3884 \Umathcode`f=0+\M@bcal@num"1D4EF\relax
3885 \Umathcode`g=0+\M@bcal@num"1D4F0\relax
3886 \Umathcode`h=0+\M@bcal@num"1D4F1\relax
3887 \Umathcode`i=0+\M@bcal@num"1D4F2\relax
3888 \Umathcode`j=0+\M@bcal@num"1D4F3\relax
3889 \Umathcode`k=0+\M@bcal@num"1D4F4\relax
3890 \Umathcode`l=0+\M@bcal@num"1D4F5\relax
3891 \Umathcode`m=0+\M@bcal@num"1D4F6\relax
3892 \Umathcode`n=0+\M@bcal@num"1D4F7\relax
3893 \Umathcode`o=0+\M@bcal@num"1D4F8\relax
3894 \Umathcode`p=0+\M@bcal@num"1D4F9\relax
3895 \Umathcode`q=0+\M@bcal@num"1D4FA\relax
3896 \Umathcode`r=0+\M@bcal@num"1D4FB\relax
3897 \Umathcode`s=0+\M@bcal@num"1D4FC\relax
3898 \Umathcode`t=0+\M@bcal@num"1D4FD\relax
3899 \Umathcode`u=0+\M@bcal@num"1D4FE\relax
3900 \Umathcode`v=0+\M@bcal@num"1D4FF\relax
3901 \Umathcode`w=0+\M@bcal@num"1D500\relax
3902 \Umathcode`x=0+\M@bcal@num"1D501\relax
3903 \Umathcode`y=0+\M@bcal@num"1D502\relax
3904 \Umathcode`z=0+\M@bcal@num"1D503\relax}}

```

Set bold fraktur letters.

```

\M@bfrak@set 3905 \def\M@bfrak@set{
\mathbfrak 3906 \protected\def\mathbfrak##1{\relax
3907 \ifmmode\else
3908 \M@HModeError\mathbfrak
3909 $
3910 \fi

```

```

3911     \begingroup
3912     \M@bfrak@mathcodes
3913     ##1
3914     \endgroup}
\M@bfrak@num 3915 \edef\M@bfrak@num{\number
3916     \csname symM\@tempc-\M@bfrakshape\endcsname}
\M@bfrak@mathcodes 3917 \protected\edef\M@bfrak@mathcodes{
3918     \Umathcode`A=0+\M@bfrak@num"1D56C\relax
3919     \Umathcode`B=0+\M@bfrak@num"1D56D\relax
3920     \Umathcode`C=0+\M@bfrak@num"1D56E\relax
3921     \Umathcode`D=0+\M@bfrak@num"1D56F\relax
3922     \Umathcode`E=0+\M@bfrak@num"1D570\relax
3923     \Umathcode`F=0+\M@bfrak@num"1D571\relax
3924     \Umathcode`G=0+\M@bfrak@num"1D572\relax
3925     \Umathcode`H=0+\M@bfrak@num"1D573\relax
3926     \Umathcode`I=0+\M@bfrak@num"1D574\relax
3927     \Umathcode`J=0+\M@bfrak@num"1D575\relax
3928     \Umathcode`K=0+\M@bfrak@num"1D576\relax
3929     \Umathcode`L=0+\M@bfrak@num"1D577\relax
3930     \Umathcode`M=0+\M@bfrak@num"1D578\relax
3931     \Umathcode`N=0+\M@bfrak@num"1D579\relax
3932     \Umathcode`O=0+\M@bfrak@num"1D57A\relax
3933     \Umathcode`P=0+\M@bfrak@num"1D57B\relax
3934     \Umathcode`Q=0+\M@bfrak@num"1D57C\relax
3935     \Umathcode`R=0+\M@bfrak@num"1D57D\relax
3936     \Umathcode`S=0+\M@bfrak@num"1D57E\relax
3937     \Umathcode`T=0+\M@bfrak@num"1D57F\relax
3938     \Umathcode`U=0+\M@bfrak@num"1D580\relax
3939     \Umathcode`V=0+\M@bfrak@num"1D581\relax
3940     \Umathcode`W=0+\M@bfrak@num"1D582\relax
3941     \Umathcode`X=0+\M@bfrak@num"1D583\relax
3942     \Umathcode`Y=0+\M@bfrak@num"1D584\relax
3943     \Umathcode`Z=0+\M@bfrak@num"1D585\relax
3944     \Umathcode`a=0+\M@bfrak@num"1D586\relax
3945     \Umathcode`b=0+\M@bfrak@num"1D587\relax
3946     \Umathcode`c=0+\M@bfrak@num"1D588\relax
3947     \Umathcode`d=0+\M@bfrak@num"1D589\relax
3948     \Umathcode`e=0+\M@bfrak@num"1D58A\relax
3949     \Umathcode`f=0+\M@bfrak@num"1D58B\relax
3950     \Umathcode`g=0+\M@bfrak@num"1D58C\relax
3951     \Umathcode`h=0+\M@bfrak@num"1D58D\relax
3952     \Umathcode`i=0+\M@bfrak@num"1D58E\relax

```



```

3953    \Umathcode`j=0+\M@bfrak@num"1D58F\relax
3954    \Umathcode`k=0+\M@bfrak@num"1D590\relax
3955    \Umathcode`l=0+\M@bfrak@num"1D591\relax
3956    \Umathcode`m=0+\M@bfrak@num"1D592\relax
3957    \Umathcode`n=0+\M@bfrak@num"1D593\relax
3958    \Umathcode`o=0+\M@bfrak@num"1D594\relax
3959    \Umathcode`p=0+\M@bfrak@num"1D595\relax
3960    \Umathcode`q=0+\M@bfrak@num"1D596\relax
3961    \Umathcode`r=0+\M@bfrak@num"1D597\relax
3962    \Umathcode`s=0+\M@bfrak@num"1D598\relax
3963    \Umathcode`t=0+\M@bfrak@num"1D599\relax
3964    \Umathcode`u=0+\M@bfrak@num"1D59A\relax
3965    \Umathcode`v=0+\M@bfrak@num"1D59B\relax
3966    \Umathcode`w=0+\M@bfrak@num"1D59C\relax
3967    \Umathcode`x=0+\M@bfrak@num"1D59D\relax
3968    \Umathcode`y=0+\M@bfrak@num"1D59E\relax
3969    \Umathcode`z=0+\M@bfrak@num"1D59F\relax}}

```

Reset \endlinechar.

```
3970 \endlinechar\count@
```

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 in unspecified mode (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 (fixed the `\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
- 2.4a** June 2025
- bug fix involving nil value and `the_font`
 - changed underscores in file names to hyphens
- 3.0** January 2026
- `\mathfont` and `\setfont` no longer restricted to preamble
 - effects of `\mathfont` are local instead of global
 - `\mathfont` now overrides previous calls to `\mathfont`
 - added `\mathfontshapes`
 - added `\mainfont`
 - changed `\setfont` to `\documentfont` (but kept old name for backwards compatibility)
 - changed `\setmathfontcommands` to `\mathfontcommands` and added documentation in the user guide

- added support for arbitrary NFSS series/shape codes in `\mathfont`
- separated macros for default shapes from macros for shapes used in `\mathfont`
- formalized language of keywords and shape identifiers
- font-loader is better about using argument with spaces removed for checking NFSS font family
- font-loader now uses `\@tempshape` and `\@tempseries` when parsing argument
- moved assignment of `\M@count` values into `\M@newfont`
- changed `\CharmLine` to `\charmline` (but kept old name for backwards compatibility)
- `\charmline` now uses `token.scan_string()` for scanning and expanding its argument
- changed `\CharmFile` to `\charmfile` (but kept old name for backwards compatibility)
- added `\charminfo`
- added `\charmtype`
- renamed counts for `\SurdHorizontalFactor`, etc. to be at user level
- formalized language of user-friendly versions of LuaTeX-only commands in the user guide
- corrected axis height
- font adjustments now happen only if `mode=base`
- no more virtual Latin letters in extra encoding slots (package now modifies Latin letters in their regular encoding slots for base-mode fonts)
- added `bold` and `bolditalic` shape identifiers
- no more type `u` (became type `a`)
- added separate charm information for upright and italic fonts (specified using `/` character)
- reworked default charm values
- possible to force type `a` and type `e` characters using `?` and `!` in `\charmline`
- added extra field in type `e` for italic correction
- clarified role of `bot_accent`, which does nothing, in the user guide
- cleaned up Lua code
- added `finishing_touches` callback
- bug fix involving the handling of italic correction in Lua font adjustments
- bug fix involving characters in operator font not displaying
- corrected UTF-16BE information added to character subtables in the font
- new definition for `\not`
- added `\negslash`
- added `script=math` to OpenType features for fonts if intended for math (so math fonts load with built-in math features)
- added support for Unicode input
- switched `\epsilon` and `\varepsilon` (so `\epsilon` is Unicode epsilon and `\varepsilon` is Unicode lunate epsilon)
- surd is now an active character

- in math mode (`\mathcode"8000`)
- cleaned up error messages
- improved messaging
- `\AtBeginDocument`
- better error checking in
- `\M@check@int`

3.0a February 2026

- removed spurious space in
- `\mainfont`
- for base-mode version of fonts,
- replaced `script=math` with
- `-nomathparam`
- added `script=latn` to default
- font features

Index

Entries refer to lines in the code. Bold means a definition.

Symbols	
<code>\#</code>	3263
<code>\%</code>	1614, 3264
<code>\&</code>	3265
<code>*</code>	319, 334, 338, 342
<code>\-</code>	3182
<code>\/</code>	320, 335, 339, 343
<code>\<</code>	624
<code>\=</code>	321, 348–350, 1532, 3183
<code>\@@Relbar</code> ..	3190, 3194, 3630, 3633
<code>\@@relbar</code> ..	3189, 3193, 3629, 3632
<code>\@Relbar</code>	3183, 3186, 3622, 3624, 3626
<code>\@documentfont</code>	1087, 1088
<code>\@mainfont</code>	968, 969 , 1089
<code>\@mathconstantsfont</code> ..	1105, 1106
<code>\@mathfontshapes</code>	1152, 1154 , 1191
<code>\@percentchar</code>	1728,
	1862, 1951, 1959, 1960, 2045, 2158
<code>\@relbar</code>	3182, 3185, 3621, 3623, 3625
<code>\@sqrts@gn</code>	3113 , 3117, 3132
<code>\~</code>	1531
<code>_</code>	142, 185
A	
<code>\aacute</code>	2818
<code>\acute</code>	2817
<code>\addtocharm@</code>	1615, 1620, 1665
<code>\agreeklowerdefault</code>	536
<code>\agreekupperdefault</code>	535
<code>\aleph</code>	2999
<code>\Alpha</code>	2560, 2830
<code>\amalg</code>	3333
<code>\angle</code>	3290, 3305
<code>\approx</code>	3252
<code>\approxeq</code>	3383
<code>\arceq</code>	3390
<code>\arrowsdefault</code>	548
<code>\asymp</code>	3411
<code>\ayin</code>	3014
B	
<code>\bar</code>	2825
<code>\bbdefault</code>	549
<code>\bcaldefault</code>	552
<code>\bclubsuit</code>	3315, 3326
<code>\bdiamondsuit</code>	3316
<code>\because</code>	3368
<code>\Beta</code>	2561, 2831
<code>\beta</code>	2586, 2875
<code>\beth</code>	3000
<code>\bfrakdefault</code>	553
<code>\bheartsuit</code>	3317
<code>\bigand</code>	3204
<code>\bigat</code>	3200
<code>\bigcap</code>	2648, 3152, 3161
<code>\bigcup</code>	2647, 3151, 3160
<code>\bigdiv</code>	3210
<code>\bigdollar</code>	3202
<code>\bighash</code>	3201
<code>\bigodot</code>	2651, 3155, 3169
<code>\bigoplus</code>	2649, 3153, 3167
<code>\bigopsdefault</code>	544
<code>\bigotimes</code>	2650, 3154, 3168
<code>\bigp</code>	3206
<code>\bigpercent</code>	3203
<code>\bigplus</code>	3205
<code>\bigq</code>	3207
<code>\bigS</code>	3208
<code>\bigsqcap</code>	2652, 3170
<code>\bigsqcup</code>	2653, 3156, 3171
<code>\bigtimes</code>	3199, 3209
<code>\bigvee</code>	2645, 3149, 3158
<code>\bigwedge</code>	2646, 3150, 3159
<code>\bot</code>	3307
<code>\bowtie</code>	3294, 3365
<code>\breve</code>	2822

\bspadesuit 3318, 3323
 \bullet 3241

C

\caldefault **550**
 \cdot 3244
 \CharmFile .. 91, 1356, 1373, **1685**
 \charmfile
 89, 729, 1356, 1371, **1666**, 1685
 \CharmInfo .. 92, 1356, **1688**, 1705
 \charminfo
 .. 110, 1361, **1361**, **1686**, 1692
 \charminfo@ 1616, 1623, 1686
 \CharmLine .. 90, 1356, 1372, **1684**
 \charmline 88, 1346,
 1355, 1370, **1665**, 1675, 1684
 \CharmType .. 93, 1356, **1707**, 1716
 \charmtype
 .. 114, 1365, **1365**, **1687**, 1714
 \charmtype@ 1617, 1656, 1687
 \check 2824
 \Chi 2581, 2851
 \chi 2606, 2895
 \circlearrowleft 3561
 \circlearrowright 3530
 \clubsuit 3326
 \colon 3179, 3255
 \coloneq 3387
 \comma 3259
 \cong 3269, 3382
 \coprod 2644, 3148, 3157
 \curvearrowleft 3560
 \curvearrowright 3529
 \cyrilliclowerdefault **538**
 \cyrillicupperdefault **537**

D

\dagger 3242
 \daleth 3002
 \dashv 3309
 \ddagger 3243
 \ddot 2820
 \Downarrow 3592

\DeclareFontFamily 809, 815
 \DeclareFontShape 749
 \DeclareMathAlphabet 1228
 \defeq 3395
 \degree 3231
 \delimitersdefault **542**
 \Delta 2563, 2833
 \delta 2588, 2877
 \diacriticsdefault **532**
 \diamond 3337
 \diamondsuit 3325
 \Digamma 2909
 \digamma 2921
 \digitsdefault **540**
 \div 2641, 3239
 \documentfont
 118, **1087**, 1101, 1102, 1520, 1526
 \dot 2819
 \doteq 3295, 3386
 \Downarrow 3493, 3591
 \downarrow 3492, 3590
 \downarrowto bar 3600
 \downbararrow 3593
 \downdasharrow 3594
 \downdownarrows 3602
 \downharpoonleft 3597
 \downharpoonright 3598
 \downuparrows 3606
 \downupharpoons 3608
 \downwhitearrow 3601

E

\ell 3298
 \emptyset 3301
 \Epsilon 2564, 2834
 \epsilon 2589, 2878
 \eqcolon 3388
 \eqsim 3380
 \equiv 3253
 \Eta 2566, 2836
 \exists 3300
 \extbigopsdefault **545**
 \extsymbolsdefault **547**

F

`\fakelangle` 2201, 2627, 3078
`\fakellangle` 2205, 2629, 3084
`\fakerrangle` 2203, 2628, 3081
`\fakerrangle` 2207, 2630, 3087
`\familydefault` **994**
`\fflat` 3313
`\flat` 3310
`\fontfamily` 1001
`\forall` 3299
`\fractionslash` 3238
`\frakdefault` **551**
`\from` 3537

G

`\Gamma` 2562, 2832
`\gamma` 2587, 2876
`\gapprox` 3378
`\geq` 3250
`\geqq` 3376
`\ggg` 3374
`\gimel` 3001
`\gnapprox` 3446
`\gneq` 3429
`\gneqq` 3431
`\gnsim` 3444
`\grave` 2821
`\greeklowerdefault` **534**
`\greekupperdefault` **533**
`\gsim` 3398

H

`\hat` 2823
`\hbar` 2813, 2814
`\heartsuit` 3324
`\hebrewdefault` **539**
`\het` 3006
`\Heta` 2907
`\heta` 2919
`\hookleftarrow` 3500, 3550
`\hookrightarrow` 3499, 3519
`\hourglass` 3366
`\hsurdfactor` 493, 499, 1332

I

`\ifM@adjust@font` 67,
 423, 435, 561, 1147, 1268, 1528,
 2698, 2855, 3041, 3104, 3198, 3485
`\ifM@arg@good` 528, 1225,
 1324, 1331, 1338, 1345, 1690, 1709
`\ifM@font@loaded` .. 68, 1513, 1525
`\ifM@mode@` 529, 1027, 1164
`\ifM@Noluaotfload` 66, 183
`\ifM@XeTeXLuaTeX` 65, 140
`\iiint` 2657, 3173
`\iiintop` 3163, 3173
`\iint` 2656, 3172
`\iintop` 3162, 3172
`\Im` 3472, 3474
`\imath` 2558, 2811
`\increment` 2857, 2864, 3232
`\infty` 3229
`\IntegralItalicFactor`
 85, **1343**, 1351, 1354, 1375
`\intop` 2655, 3145
`\Iota` 2568, 2838
`\iota` 2593, 2882

J

`\jmath` 2559, 2812

K

`\kaf` 3009
`\Kappa` 2569, 2839
`\kappa` 2594, 2883
`\kern` 3124, 3126, 3129
`\Koppa` 2910
`\koppa` 2922

L

`\Lambda` 2570, 2840
`\lambda` 2595, 2884
`\lamed` 3010
`\lapprox` 3377
`\lbrace` 3066
`\lcirclearrow` 3619
`\Leftarrow` 3539

- \M@check@nfss@shapes **734**, 779, 784
- \M@check@opt **864**, 881
- \M@check@specials **1204**, 1229
- \M@cyrillic@lower@set .. 1075, **2963**
- \M@cyrillic@lower@shape 2964
- \M@cyrillic@upper@set .. 1075, **2929**
- \M@cyrillic@upper@shape 2930
- \M@declare@shape ... **746**, 760, 862
- \M@DecSymDef **486**, 488
- \M@default@newmath@cmds
..... **1237**, 1246, **1257**
- \M@default@keys **559**, 562, **562**, 1004
- \M@define@newmath@cmd
..... **1233**, 1247, 1256
- \M@delimiters@set 1075, **3042**, **3091**
- \M@delimiters@shape ... 3043, 3092
- \M@diacritics@set 1075, **2815**
- \M@diacritic@shape 2816
- \M@digits@set 1075, **3026**
- \M@digit@shape 3027
- \M@empty@assert 1799
- \M@entries@assert 1903, 1919
- \M@entries@warning ... 1905, 1921
- \M@extbigops@set 1075, **3146**
- \M@extbigop@shape 3147
- \M@extsymbols@set 1075, **3288**
- \M@extsymbol@shape 3289
- \M@families **566**, 593,
595, **595**, 597, **597**, 1476, 1479
- \M@families@begin **1459**, 1480
- \M@FamilyTypeError **626**, 986
- \M@fill@nfss@shapes
..... **751**, 810, 816, 863
- \M@FontChangeInfo **598**, 1067
- \M@FontFamilyInfo .. **599**, 808, 814
- \M@fontinfo@begin **1432**, 1497
- \M@fontinfo@begin@
..... 1436, **1447**, 1457, 1458
- \M@FontShapesError
..... **670**, 804, 1032, 1050, 1117
- \M@frak@mathcodes 3782, **3787**
- \M@frak@num **3785**, 3788–3839
- \M@frak@set 1075, **3775**
- \M@frak@shape 3786
- \M@greek@lower@set 1075, **2872**
- \M@greek@lower@shape 2873
- \M@greek@upper@set 1075, **2828**
- \M@greek@upper@shape 2829
- \M@hebrew@set 1075, **2997**
- \M@hebrew@shape 2998
- \M@HModeError
..... **699**, 3638, 3713, 3778, 3843, 3908
- \M@index@assert 1821, 1823
- \M@InvalidOptionError .. **634**, 869
- \M@InvalidSuboptionError **642**, 951
- \M@keys **554**, 870, 1496
- \M@loader
70, 366, 367, 373, 406, 408, 806
- \M@localfonts@begin .. **1457**, 1506
- \M@lower@set 1075, **2783**
- \M@lower@shape 2784
- \M@LuaTeXOnlyWarning .. **682**, 1149
- \M@missing@assert 1803
- \M@MissingCSError **690**, 1197, 1201
- \M@MissingOptionError .. **651**, 867
- \M@MissingSuboptionError **660**, 888
- \M@mode@false 908, 911
- \M@mode@true 905
- \M@newfont
..... **765**, 971, 1007, 1107, 1156, 1226
- \M@NewFontCommandInfo .. **602**, 1227
- \M@NFSSShapesWarning ... **604**, 742
- \M@NoBaseModeError **612**, 786
- \M@NoCharmFileError .. **725**, 1683
- \M@NoFontAdjustError
..... **708**, 1359, 1361, 1365
- \M@NoLuaotfloadError ... **186**, 214
- \M@NoMathfontError
.. **72**, 95, 107, 110, 114, 124, 131
- \M@num@localfonts
..... 491, 579, 1399, 1408, 1502
- \M@number@assert 1776
- \M@operator@set 1075, **3038**
- \M@operator@shape 3040

- \M@otf@features
... **412**, **417**, 810, 816, 825, 831
 - \M@p@tch@decl@re **485**, 486
 - \M@parse@option **878**, 1012
 - \M@parse@sub
.... 894, **897**, 1017, 1109, 1162
 - \M@radical@set .. 1075, **3105**, **3134**
 - \M@radicalshape 3106, 3135
 - \M@SetMathConstants
..... **1103**, 1129, **1130**, 1151
 - \M@split@colon **761**, 766
 - \M@split@slash 915, **959**
 - \M@split@star 901, **962**
 - \M@strip@colon **764**, 768
 - \M@strip@equals **877**, 893
 - \M@strip@star **965**
 - \M@SymbolFontInfo
 600, 1034, 1052, 1119, 1167, 1179
 - \M@symbolfonts@begin .. **1458**, 1490
 - \M@symbols@set 1075, **3177**
 - \M@symbolsshape 3178
 - \m@th@const@nts@f@mily . **1110**,
 1113, 1119, 1126, 1135, 1139
 - \m@th@const@nts@series
..... **1111**, 1136, 1140
 - \m@th@const@nts@sh@pe
..... **1112**, 1136, 1141
 - \M@upper@set 1075, **2755**
 - \M@uppershape 2756
 - \M@virtual@ssert 2014
 - \M@XeTeXLuaTeXError ... **143**, 165
 - \mainfont .. 120, 629, 631, **968**, 970
 - \mapsfrom 3545
 - \mapsto 3514
 - \math@fonts 1144, **1151**
 - \mathand 3221, 3265
 - \mathbackslash 3063, 3064
 - \mathbb 1205, **3636**, 3638
 - \mathbcal 1205, **3841**, 3843
 - \mathbf 1261
 - \mathbfit 1262
 - \mathbffrac 1205, **3906**, 3908
 - \mathbfsc 1265
 - \mathbfscit 1266
 - \mathcal 1205, **3711**, 3713
 - \mathchoice 3274
 - \mathconstantsfont
..... 122, 1092, **1104**, 1149
 - \mathdollar 3219
 - \mathellipsis 3180, 3262
 - \mathfontcommands 1096, **1258**, 1267
 - \mathfontshapes 121, 681, **1152**, 1190
 - \mathfrac 1205, **3776**, 3778
 - \mathhash 3218, 3263
 - \mathit 1260
 - \mathng 3483, 3484
 - \mathnolimitsmode 1529
 - \mathparagraph 3222
 - \mathpercent 3220, 3264
 - \mathring 2826
 - \mathrm 1259
 - \mathsc 1263
 - \mathscit 1264
 - \mathsection 3223
 - \mathsterling 3224, 3225
 - \mem 3011
 - \mid 3227
 - \mkern 3132
 - \models **3272**
- N**
- \nabla 2859, 2868, 3211, 3486
 - \approx 3437
 - \natural 3311
 - \Nearrow 3610
 - \nearrow 3609
 - \neg 3226
 - \negslash **3273**
 - \neq 3296, 3424
 - \nequiv 3459
 - \neswarrow 3618
 - \newmathbf .. 100, 1240, 1250, 1261
 - \newmathbfit . 101, 1241, 1251, 1262
 - \newmathbfsc . 104, 1244, 1254, 1265

Q

`\qeq` 3396
`\qof` 3017

R

`\r@@t` **3115**
`\radicaldefault` **543**
`\radicandoffset` ... 495, 501, 3132
`\ratio` 3369
`\rbrace` 3068
`\rcirclearrow` 3620
`\Relbar` 3186,
3190, 3194, 3626, 3630, 3633
`\relbar` 3184,
3189, 3193, 3625, 3629, 3632
`\resh` 3018
`\rguil` 2198, 2624, 3072, 3098
`\Rho` 2576, 2846
`\rho` 2601, 2890
`\Rightarrow` 3508
`\rightarrow` 3505, 3506
`\rightarrowtail` 3523
`\rightarrowto` 3532
`\Rightbararrow` 3515
`\rightbararrow` 3513, 3514
`\rightbrace` 3051, 3102
`\righdasharrow` 3520
`\rightharpoon` 3522
`\rightharpoonup` 3521
`\rightleftarrows` 3576
`\rightleftharpoons` ... 3504, 3577
`\rightoplusarrow` 3524
`\righttrightrightarrows` 3534
`\righttrightrightarrows` ... 3535
`\rightsquigarrow` 3526
`\rightwvearrow` 3525
`\rightwhitearrow` 3533
`\ringeq` 3389
`\rootbox` 3123, 3126, 3128
`\rrguil` 2200, 2626, 3076, 3100
`\Rrightarrow` 3510
`\RuleThicknessFactor`
.... 84, **1322**, 1327, 1354, 1374

`\rulethicknessfactor` 492, 498, 1325

S

`\samekh` 3013
`\Sampi` 2908
`\sampi` 2920
`\San` 2913
`\san` 2925
`\Searrow` 3614
`\searrow` 3613
`\selectfont` 1001
`\seq` 3385
`\setfont` 83, **1098**, 1100
`\setminus` 3245
`\sharp` 3312
`\shin` 3019
`\Sho` 2912
`\Sigma` 2577, 2847
`\sigma` 2602, 2891
`\sim` 3251, 3268, 3270
`\simeq` 3267, 3291, 3379
`\simeqq` 3381, 3382
`\simneqq` 3440
`\spadesuit` 3323
`\sqcap` 3331
`\sqcup` 3332
`\sqdot` 3349
`\sqminus` 3348
`\sqplus` 3346
`\sqrt` 3111
`\sqrtsign` 3116, **1311**
`\sqsubset` 3292, 3356
`\sqsubseteq` 3358
`\sqsubsetneq` 3422
`\sqsupset` 3293, 3357
`\sqsupseteq` 3359
`\sqsupsetneq` 3423
`\sqtimes` 3347
`\ssharp` 3314
`\sssim` 3384
`\stack@fl@trel` 1380, **1381**
`\stack@flatrel` .. **1379**, 3268, 3270

<code>\star</code>	3336	<code>\tilde</code>	2827
<code>\stareq</code>	3393	<code>\times</code>	2640, 3236
<code>\Stigma</code>	2911	<code>\tracinglostchars</code>	1395, 1396
<code>\stigma</code>	2923	<code>\triangleeq</code>	3394
<code>\subset</code>	3352	<code>\triangleleft</code>	3360
<code>\subseteq</code>	3354	<code>\trianglelefteq</code>	3362
<code>\subsetneq</code>	3418	<code>\triangleright</code>	3361
<code>\succ</code>	3400	<code>\trianglerighteq</code>	3363
<code>\succapprox</code>	3408	<code>\tsadi</code>	3016
<code>\succeq</code>	3402	<code>\twoheaddownarrow</code>	3599
<code>\succeqq</code>	3404	<code>\twoheadleftarrow</code>	3562
<code>\succnapprox</code>	3458	<code>\twoheadrightarrow</code>	3531
<code>\succneq</code>	3452	<code>\twoheaduparrow</code>	3585
<code>\succneqq</code>	3454		
<code>\succnsim</code>	3456	U	
<code>\succsim</code>	3406	<code>\Uparrow</code>	3491, 3579
<code>\succsucc</code>	3410	<code>\uparrow</code>	3490, 3578
<code>\sum</code>	2643, 3141, 3143	<code>\uparrowto</code>	3586
<code>\supset</code>	3353	<code>\upbararrow</code>	3581
<code>\supseteq</code>	3355	<code>\updasharrow</code>	3582
<code>\supsetneq</code>	3419	<code>\Updownarrow</code>	3495, 3604
<code>\surd</code> . 2654, 3107 , 3108, 3136 , 3137		<code>\updownarrow</code>	3494, 3603
<code>\surdbox</code>	489,	<code>\updownarrows</code>	3605
3117, 3119, 3120, 3122–3124, 3129		<code>\updownharpoons</code>	3607
<code>\SurdHorizontalFactor</code>		<code>\upharpoonleft</code>	3583
.... 87, 1329 , 1334, 1355, 1376		<code>\upharpoonright</code>	3584
<code>\SurdVerticalFactor</code>		<code>\upperdefault</code>	530
.... 86, 1336 , 1341, 1355, 1377		<code>\Upsilon</code>	2579, 2849
<code>\Swarrow</code>	3616	<code>\upsilon</code>	2604, 2893
<code>\swarrow</code>	3615	<code>\upuparrows</code>	3589
<code>\symbolsdefault</code>	546	<code>\upwhitearrow</code>	3587
		<code>\upwhitebararrow</code>	3588
T		<code>\Uradical</code>	3113
<code>\Tau</code>	2578, 2848	<code>\Uparrow</code>	3580
<code>\tau</code>	2603, 2892		
<code>\tav</code>	3020	V	
<code>\tet</code>	3007	<code>\varbeta</code>	2609, 2898
<code>\textbackslash</code>	3063	<code>\varcdot</code>	3338
<code>\textng</code>	3482, 3484	<code>\varDigamma</code>	2915
<code>\therefore</code>	3367	<code>\vardigamma</code>	2927
<code>\Theta</code>	2567, 2837	<code>\varepsilon</code>	2610, 2899
<code>\theta</code>	2592, 2881	<code>\varIm</code>	3477, 3479
		<code>\varkaf</code>	3021

