Package mathfont v. 2.3 Implementation

Conrad Kosowsky
September 2023
kosowsky.latex@gmail.com

For easy, off-the-shelf use, type the following in your preamble and compile with XƎLaTeX or LuaLaTeX:

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

As of version 2.0, using LuaLaTeX is recommended.

Overview

The mathfont package adapts unicode text fonts for math mode. The package allows the user to specify a default unicode font for different classes of math symbols, and it provides tools to change the font locally for math alphabet characters. When typesetting with LuaTeX, mathfont adds resizable delimiters, big operators, and a MathConstants table to text fonts.

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

1 Implementation Basics

First and foremost, the package needs to declare itself.

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

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

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

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

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

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

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

```
\@tfor\@i:=\setfont
  \RuleThicknessFactor
  \IntegralItalicFactor
  \SurdVerticalFactor
  \SurdHorizontalFactor
  \CharmLine
  \CharmFile\do{%
    \edef\@tempa{\protected\def\expandafter\noexpand\@i}%
    \edef\@tempb{\noexpand\M@NoMathfontError
      \expandafter\noexpand\@i
      \noexpand\@gobble}%
    \expandafter\@tempa\expandafter{\@tempb}}
```

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

```
\@tfor\@i:=\newmathrm
  \newmathit
  \newmathbf
  \newmathbfit
  \newmathbold
  \newmathboldit
  \newmathsc
  \newmathscit
  \newmathbfsc
  \newmathbfscit\do{%
    \edef\@tempa{\protected\def\expandafter\noexpand\@i}%
    \edef\@tempb{\noexpand\M@NoMathfontError
      \expandafter\noexpand\@i
      \noexpand\@gobbletwo}%
    \expandafter\@tempa\expandafter{\@tempb}}
```

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

```
\@tfor\@i:=\mathfont\mathconstantsfont\do{%
  \edef\@tempa{\protected\def\expandafter\noexpand\@i}%
  \edef\@tempb{\noexpand\M@NoMathfontError
    \expandafter\noexpand\@i
    \noexpand\@ifnextchar[{\noexpand\expandafter
      \noexpand\@gobble\noexpand\@gobble@brackets}{\noexpand\@gobble}}
  \expandafter\@tempa\expandafter{\@tempb}}
```

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

```
\protected\def\newmathfontcommand{%
  \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.)

```
\ifdefined\directlua
  \M@XeTeXLuaTeXtrue
\fi
\ifdefined\XeTeXrevision
  \M@XeTeXLuaTeXtrue
\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 `\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.

```
\ifM@XeTeXLuaTeX\else
\begingroup
\catcode`\ =12\relax
```

```
\def\M@XeTeXLuaTeXError{\GenericError{}%
{\MessageBreak\MessageBreak
Package mathfont error:%
\MessageBreak\MessageBreak
************************\MessageBreak
*                       *\MessageBreak
*       UNABLE TO       *\MessageBreak
*     LOAD MATHFONT     *\MessageBreak
*                       *\MessageBreak
*     Missing XeTeX     *\MessageBreak
*       or LuaTeX       *\MessageBreak
*                       *\MessageBreak
************************\MessageBreak\@gobbletwo}%
{See the mathfont package documentation for explanation.}%
{I need XeTeX or LuaTeX to use mathfont. It\MessageBreak
looks like the current engine is something\MessageBreak
else, so I'm going to stop reading in the\MessageBreak
package file now. (You won't be able to use\MessageBreak
commands from mathfont in your document.) To\MessageBreak
load mathfont correctly, please retypeset your\MessageBreak
document with one of those two engines.^^J}}%
\expandafter\endgroup
\M@XeTeXLuaTeXError
  \AtEndOfPackage{\typeout{:: mathfont :: Failed to load\on@line.}}
  \expandafter\endinput % we should \endinput with a balanced conditional
\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 XƎTEX, issue a warning.

```
\ifdefined\directlua
  \M@adjust@fonttrue % if engine is LuaTeX, adjust font by default
  \IfFileExists{luaotfload.sty}
    {\M@Noluaotfloadfalse\RequirePackage{luaotfload}}{\M@Noluaotfloadtrue}
\else
  \AtEndOfPackage{\PackageWarningNoLine{mathfont}{%
    The current engine is XeTeX, but as\MessageBreak
    of mathfont version 2.0, LuaTeX is\MessageBreak
    recommended. Consider compiling with\MessageBreak
    LuaLaTeX. Certain features will not\MessageBreak
    work with XeTeX}}
\fi
```

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

```
\ifM@Noluaotfload % false by default; true if LuaTeX AND no luaotfload.sty
\begingroup
\catcode`\ =12\relax
\def\M@NoluaotfloadError{\GenericError{}%
{\MessageBreak\MessageBreak
Package mathfont error:%
\MessageBreak\MessageBreak
*************************\MessageBreak
*                       *\MessageBreak
*       UNABLE TO       *\MessageBreak
*     LOAD MATHFONT     *\MessageBreak
*                       *\MessageBreak
*    Cannot find the    *\MessageBreak
*  file luaotfload.sty  *\MessageBreak
*                       *\MessageBreak
*************************\MessageBreak\@gobbletwo}%
{You are likely seeing this message because you haven't^^J%
installed luaotfload. Check your TeX distribution for a^^J%
list of the packages on your system.^^J^^J%
See the mathfont documentation for further explanation.}%
{You're in trouble here. It looks like the current\MessageBreak
engine is LuaTeX, so I need the luaotfload package\MessageBreak
to make mathfont work properly. However, I can't\MessageBreak
find luaotfload, which likely means something is\MessageBreak
wrong with your TeX installation. I'm going to stop\MessageBreak
reading in the mathfont package file. (You won't be\MessageBreak
able to use commands from mathfont in your document.)\MessageBreak
To load mathfont work correctly, make sure you have\MessageBreak
installed luaotfload.sty in a directory searchable\MessageBreak
by TeX or compile with XeLaTeX.^^J}}%
\expandafter\endgroup
\M@NoluaotfloadError
  \AtEndOfPackage{\typeout{:: mathfont :: Failed to load\on@line.}}
  \expandafter\endinput % we should \endinput with a balanced conditional
\fi
```

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

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

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

```
\DeclareOption{packages}{%
```

```
  \M@Optiondeprecated{packages}
  {the macro \string\restoremathinternals}}
\DeclareOption{operators}{%
  \M@Optiondeprecated{operators}
  {the bigops keyword with \string\mathfont}}
\DeclareOption{no-operators}{%
  \M@Optiondeprecated{no-operators}
  {the bigops keyword with \string\mathfont}}
```

Easter egg!

```
\DeclareOption{easter-egg}{%
  \ifE@sterEggDecl@red\else
    \E@sterEggDecl@redtrue
    \def\EasterEggUpdate{\show\E@sterEggUpd@te}
    \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
        Okay, opening your Easter egg.^^J^^J}
      \EasterEggUpdate
    \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
        Uh oh. It looks like^^J%
        your Easter egg flew^^J%
        out the window. I don't^^J%
        suppose you know the^^J%
        best kind of bait to^^J%
        lure an egg?^^J^^J}
      \EasterEggUpdate
    \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
        Still wrangling. Try back later.^^J^^J}
    \AtBeginDocument{\bgroup
      \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J:%
        If we have zero eggs^^J%
        and zero bunnies, how^^J%
        many gnats does it take^^J%
        to change a lightbulb??^^J^^J}
      \EasterEggUpdate
      \egroup}
    \AtEndDocument{%
      \def\E@sterEggUpd@te{Easter Egg Status:^^J^^J%
        Happy, happy day! Happy,^^J%
        happy day! Clap your hands,^^J%
        and be glad your hovercraft^^J%
        isn't full of eels!^^J^^J}
        \EasterEggUpdate
      \let\E@sterEggUpd@te\relax
      \let\EasterEggUpdate\relax}
  \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.

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

```
\DeclareOption{fontspec-loader}{\M@loader\@ne}
```

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

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

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

```
\ifcase\M@loader
  \wlog{Package mathfont Info: Default font-loader was
    requested for font loading.}
\or
  \wlog{Package mathfont Info: Package fontspec was
    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.

```
  \@ifpackageloaded{fontspec}
    {\wlog{Package mathfont Info: Package fontspec detected.}
      \csname bool_if:NTF\expandafter\endcsname
        \csname g__fontspec_math_bool\endcsname
        {\wlog{Package mathfont Info: Setting
          \string\g__fontspec_math_bool to false.}
          \csname bool_set_false:N\expandafter\endcsname
          \csname g__fontspec_math_bool\endcsname}{\relax}}
```

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

```
    {\wlog{Package mathfont Info: Package fontspec not detected.}
      \IfFileExists{fontspec.sty}
        {\wlog{Package mathfont Info: File fontspec.sty was found.}
          \wlog{Package mathfont Info: Loading fontspec.}
          \RequirePackage[no-math]{fontspec}}
        {\PackageError{mathfont}{Missing package fontspec;\MessageBreak
          using default font-loader instead}
          {You requested fontspec as the font-loader for\MessageBreak
          mathfont. However, I can't find the fontspec\MessageBreak
          package file, so I'm going to use the default\MessageBreak
          font-loader instead. (This likely means that\MessageBreak
          something is wrong with your TeX installation.)\MessageBreak
          Check your TeX distribution for a list of the\MessageBreak
          packages installed on your system. To resolve\MessageBreak
          this error, make sure fontspec is installed in\MessageBreak
          a directory searchable by TeX or load mathfont\MessageBreak
          with the default-loader option.^^J}
          \M@loader\z@}}
\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 XeTeX/luaotfload syntax versus fontspec syntax. By default, mathfont loads fonts with Latin script, default language, TeX and common ligatures, and lining numbers.

```
\ifcase\M@loader
  \wlog{Package mathfont Info: Using default font-loader.}
  \AtEndOfPackage{\typeout{:: mathfont :: Using default font-loader.}}
  \def\M@otf@features{script=latin;language=DFLT;+tlig;+liga;+lnum}
\or
  \wlog{Package mathfont Info: Using fontspec as font-loader.}
  \AtEndOfPackage{\typeout{:: mathfont :: Using fontspec as font-loader.}}
  \def\M@otf@features{Script=Latin,
    Language=Default,
    Ligatures={TeX,Common},
    Numbers=Lining}
\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`.

```
\ifdefined\directlua
  \ifM@adjust@font
    \wlog{Package mathfont Info: Enabling Lua-based font adjustments.}
    \AtEndOfPackage{%
      \typeout{:: mathfont :: Lua-based font adjustments enabled.}}
  \else
    \wlog{Package mathfont Info: Disabling Lua-based font adjustments.}
    \AtEndOfPackage{%
      \typeout{:: mathfont :: Lua-based font adjustments disabled.}}
  \fi
\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.

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

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

2 Errors and Messaging

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

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

Table 1: Various Messages and Errors and Their Uses

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

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

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

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

Error messages associated with `\mathfont`.

```
\def\M@InvalidOptionError#1{\PackageError{mathfont}
  {Invalid^^Joption "#1" for \string\mathfont\on@line}
  {Hm. You used a keyword that isn't actually an optional\MessageBreak
  argument for \string\mathfont. Check
  that you spelled the keyword\MessageBreak
  correctly. Otherwise, I'm not sure what's wrong. Is this\MessageBreak
  option listed in the package documentation? In any event,\MessageBreak
  I'm going to ignore it.^^J}}
\def\M@InvalidSuboptionError#1{\PackageError{mathfont}
  {Invalid^^Jsuboption "#1" for \string\mathfont\on@line}
  {Hm. You used a keyword that isn't actually a suboption\MessageBreak
  for \string\mathfont. Check that you
  spelled the keyword correctly.\MessageBreak
  Otherwise, I'm not sure what's wrong. Is this suboption\MessageBreak
  listed in the package documentation? In any event, I'm\MessageBreak
  going to ignore it.^^J}}
\def\M@MissingOptionError{\PackageError{mathfont}
  {Missing^^Joption for \string\mathfont\on@line}
  {It looks like you included a , or = in\MessageBreak
  the optional argument of \string\mathfont\space but\MessageBreak
  didn't put anything before it.^^J}}
\def\M@MissingSuboptionError{\PackageError{mathfont}
  {Missing^^Jsuboption for \string\mathfont\on@line}
  {It looks like you included an = somewhere\MessageBreak
  in the optional argument of \string\mathfont\space but\MessageBreak
  didn't put the suboption after it. Either\MessageBreak
  that or you typed == instead of =.^^J}}
\def\M@InternalsRestoredError{\PackageError{mathfont}
  {Internal^^Jcommands restored}
  {This package slightly changes two LaTeX\MessageBreak
  internal commands, and you really shouldn't\MessageBreak
  be loading new math fonts without those\MessageBreak
  adjustments. What happened here is that you\MessageBreak
  used \string\mathfont\space in a situation where those\MessageBreak
  two commands retain their original defini-\MessageBreak
  tions. Presumably you used \string\mathfont\space after\MessageBreak
  calling the \string\restoremathinternals\space command.\MessageBreak
  I'm going to ignore this call to \string\mathfont.\MessageBreak
  Try typesetting this document with all\MessageBreak
  \string\mathfont\space commands placed before you call\MessageBreak
  \string\restoremathinternals.^^J}}
```

Error messages for `\mathconstantsfont`.

```
\def\M@BadMathConstantsFontError#1{\PackageError{mathfont}
  {Invalid font specifier\MessageBreak
  for \string\mathconstantsfont:\MessageBreak
  "#1"}
  {Your command was ignored--I can't parse your argument.\MessageBreak
  Please make sure to use text that you have previously\MessageBreak
  fed to \string\mathfont\space for the argument of
  \string\mathconstantsfont.^^J}}
\def\M@BadMathConstantsFontTypeError#1{\PackageError{mathfont}
  {Invalid\MessageBreak font specifier for
  \string\mathconstantsfont:\MessageBreak"#1"}
  {The optional argument of \string\mathconstantsfont\MessageBreak
  should be "upright" or "italic." Right now,\MessageBreak
  it's "#1."^^J}}
\def\M@MathConstantsNoFontAdjustWarning{\PackageWarning{mathfont}
  {Your \string\mathconstantsfont\space
  on line \the\inputlineno\MessageBreak
  is mainly for use in LuaTeX with font\MessageBreak
  adjustments enabled. In the current\MessageBreak
  situation, it is probably not doing\MessageBreak
  anything\@gobble}}
```

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

```
\def\M@MissingControlSequenceError#1#2{\PackageError{mathfont}
  {Missing control sequence\MessageBreak
  for\string#1\MessageBreak on input line \the\inputlineno}
  {Your command was ignored. Right now the\MessageBreak
  first argument of \string#1\space is "#2."\MessageBreak
  Please use a control sequence instead.^^J}}
\def\M@DoubleArgError#1#2{\PackageError{mathfont}
  {Multiple characters in\MessageBreak
  first argument of \string#1\MessageBreak
  on input line \the\inputlineno}
  {Your command was ignored. Right now the\MessageBreak
  first argument of \string#1\space is "#2,"\MessageBreak
  which is multiple characters. Please use\MessageBreak
  a single character instead.^^J}}
\def\M@HModeError#1{\PackageError{mathfont}
  {Missing \string$ inserted\MessageBreak
  on input line line \the\inputlineno}
  {I generated an error because
  you used \string#1\space outside of\MessageBreak
  math mode. I inserted a \string$
  before your \string#1, so we\MessageBreak
  should be all good now.^^J}}
```

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

```
\def\M@ForbiddenCharmLine#1{\PackageError{mathfont}
  {Forbidden charm info contains #1}
  {The argument of your \string\CharmLine\space
  macro on line \the\inputlineno\MessageBreak
  contains the character #1, which will mess me up\MessageBreak
  if I try to read it, so I'm ignoring this call\MessageBreak
  to \string\CharmLine. To resolve this error, make sure\MessageBreak
  your charm information contains only integers,\MessageBreak
  floats, asterisks, commas, and spaces.^^J}}
\def\M@ForbiddenCharmFile#1{\PackageError{mathfont}
  {Forbidden charm info contains #1}
  {One of the lines in your \string\CharmFile\space
  from line \the\inputlineno\MessageBreak
  contains the character #1, which will mess me up\MessageBreak
  if I try to read it, so I'm ignoring this line\MessageBreak
  from your file. To resolve this error, make sure\MessageBreak
  your charm information contains only integers,\MessageBreak
  floats, asterisks, commas, and spaces.^^J}}
\def\M@NoFontAdjustError#1{\PackageError{mathfont}
  {Your command \MessageBreak\string#1 is invalid\MessageBreak
  without Lua-based font adjustments}
  {You haven't enabled Lua-based font adjustments,\MessageBreak
  but the macro you called won't do anything without\MessageBreak
  them. I'm going to ignore your command for now. To\MessageBreak
  resolve this error, load mathfont with the package\MessageBreak
  option "adjust" or compile with LuaLaTeX.^^J}}
```

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

3 Default Settings

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

```
\let\@@set@mathchar\set@mathchar
\let\@@set@mathsymbol\set@mathsymbol
\let\@@set@mathaccent\set@mathaccent
\let\@@DeclareSymbolFont\DeclareSymbolFont
\@onlypreamble\@@set@mathchar
\@onlypreamble\@@set@mathsymbol
\@onlypreamble\@@set@mathaccent
\@onlypreamble\@@DeclareSymbolFont
\wlog{Package mathfont Info: Adapting \noexpand\set@mathchar for unicode.}
\wlog{Package mathfont Info: Adapting \noexpand\set@mathsymbol for unicode.}
\wlog{Package mathfont Info: Adapting \noexpand\set@mathaccent for unicode.}
\wlog{Package mathfont Info: Increasing upper bound on
  \noexpand\DeclareSymbolFont to 256.}
```

Kernel command to set math characters from keystrokes.

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

Kernel command to set math characters from control sequences.

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

Kernel command to set accents.

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

We increase the upper bound on the number of symbol fonts to be 256. LuaTeX and XeTeX allow up to 256 math families, but the LaTeX kernel keeps the old upper bound of 16 symbol fonts under these two engines. We patch `\DeclareSymbolFont` to change the `\count18<15` to `\count18<\e@mathgroup@top`, where `\e@mathgroup@top` is the number of math families and is 256 in XƎTEX and LuaTEX. Because macro patching is complicated, the next few lines may seem somewhat esoteric. Our approach is to get a sanitized definition with `\meaning` and `\strip@prefix`, implement the patch by expanding `\M@p@tch@decl@re`, and retokenize the whole thing. A simpler approach, such as calling `\M@p@tch@decl@re` directly on the expansion of `\DeclareSymbolFont`, won't work because of how TEX stores and expands parameter symbols inside macros.

As of November 2022, the LATEX kernel team the name of `\DeclareSymbolFont` to `\DeclareSymbolFont@m@dropped`, and now `\DeclareSymbolFont` is a wrapper around the renamed macro. This was done for error checking purposes to remove extra `m`'s from certain NFSS family names. This means that if `\DeclareSymbolFont@m@dropped` is defined, we should patch that macro, and otherwise, we should patch `\DeclareSymbolFont`.

```
\ifx\DeclareSymbolFont@m@dropped\@undefined
  \edef\@tempa{\expandafter\strip@prefix\meaning\DeclareSymbolFont}
  \def\@tempb{\def\DeclareSymbolFont##1##2##3##4##5}
\else
  \edef\@tempa{\expandafter\strip@prefix\meaning\DeclareSymbolFont@m@dropped}
  \def\@tempb{\def\DeclareSymbolFont@m@dropped##1##2##3##4##5}
\fi
\def\M@p@tch@decl@re#1<15#2\@nil{#1<\e@mathgroup@top#2}
\edef\M@DecSymDef{\expandafter\M@p@tch@decl@re\@tempa\@nil}
```

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

```
\ifdefined\directlua
  \expandafter\@tempb\expandafter{\scantextokens\expandafter{\M@DecSymDef}}
```

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

```
\else
  \begingroup
  \everyeof{\noexpand}
  \endlinechar\m@ne
```

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

```
  \edef\M@retokenize{\noexpand\scantokens{\noexpand\unexpanded{\M@DecSymDef}}}
  \edef\M@retokenize{\M@retokenize}
  \expandafter\endgroup
    \expandafter\@tempb\expandafter{\M@retokenize}
\fi
```

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

```
\newbox\surdbox
\newcount\M@count
\newcount\M@RuleThicknessFactor
\newcount\M@IntegralItalicFactor
\newcount\M@SurdVerticalFactor
\newcount\M@SurdHorizontalFactor
\newmuskip\radicandoffset
\newread\M@Charm
\newtoks\M@toks
\M@count\z@
\M@RuleThicknessFactor\@m
\M@IntegralItalicFactor=400\relax
\M@SurdHorizontalFactor\@m
\M@SurdVerticalFactor\@m
\radicandoffset=3mu\relax
```

Necessary booleans and default math font shapes.

```
\newif\ifM@upper
\newif\ifM@lower
\newif\ifM@diacritics
\newif\ifM@greekupper
\newif\ifM@greeklower
\newif\ifM@agreekupper
\newif\ifM@agreeklower
\newif\ifM@cyrillicupper
\newif\ifM@cyrilliclower
\newif\ifM@hebrew
\newif\ifM@digits
\newif\ifM@operator
\newif\ifM@symbols
\newif\ifM@extsymbols
\newif\ifM@delimiters
\newif\ifM@radical
\newif\ifM@arrows
\newif\ifM@bigops
\newif\ifM@extbigops
\newif\ifM@bb
\newif\ifM@cal
\newif\ifM@frak
\newif\ifM@bcal
```

```
\newif\ifM@bfrak
\newif\if@optionpresent
\newif\if@suboptionpresent
\newif\ifM@arg@good
\newif\ifM@Decl@reF@mily
\newif\ifM@fromCharmFile
```

Default shapes.

```
\def\M@uppershape{italic}          % latin upper
\def\M@lowershape{italic}          % latin lower
\def\M@diacriticsshape{upright}    % diacritics
\def\M@greekuppershape{upright}    % greek upper
\def\M@greeklowershape{italic}     % greek lower
\def\M@agreekuppershape{upright}   % ancient greek upper
\def\M@agreeklowershape{italic}    % ancient greek lower
\def\M@cyrillicuppershape{upright} % cyrillic upper
\def\M@cyrilliclowershape{italic}  % cyrillic lower
\def\M@hebrewshape{upright}        % hebrew
\def\M@digitsshape{upright}        % numerals
\def\M@operatorshape{upright}      % operator font
\def\M@delimitersshape{upright}    % delimiters
\def\M@radicalshape{upright}       % surd
\def\M@bigopsshape{upright}        % big operators
\def\M@extbigopsshape{upright}     % extended big operators
\def\M@symbolsshape{upright}       % basic symbols
\def\M@extsymbolsshape{upright}    % extended symbols
\def\M@arrowsshape{upright}        % arrows
\def\M@bbshape{upright}            % blackboard bold
\def\M@calshape{upright}           % caligraphic
\def\M@frakshape{upright}          % fraktur
\def\M@bcalshape{upright}          % bold caligraphic
\def\M@bfrakshape{upright}         % bold fraktur
```

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

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

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

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

4 Fontloader

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

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

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

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

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

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

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

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

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

```
\def\M@check@nfss@shapes#1{%
  \let\@tempb\@empty
  \let\@tempwarning\@gobble
  \@for\@i:=\mddefault/\shapedefault,%
    \mddefault/\itdefault,%
    \bfdefault/\shapedefault,%
    \bfdefault/\itdefault\do{%
      \expandafter\ifx\csname TU/#1/\@i\endcsname\relax
        \def\@tempwarning{\M@MissingNFSSShapesWarning{#1}}
        \edef\@tempb{\@tempb, \@i}
      \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`.

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

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

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

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

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

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

```
  \ifcsname TU+\@tempbase\endcsname % is font family already in nfss?
    \let\M@f@ntn@me\@tempbase
    \M@check@nfss@shapes\M@f@ntn@me
    \ifx\directlua\@undefined % if XeTeX
      \expandafter\edef\csname\M@f@ntn@me-base\endcsname{\M@f@ntn@me}
      \let\M@f@ntn@meb@se\M@f@ntn@me
    \else % if LuaTeX
```

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

```
      \ifcsname\M@f@ntn@me-base\endcsname % if base-mode version is known
        \edef\M@f@ntn@meb@se{\csname\M@f@ntn@me-base\endcsname}
        \M@check@nfss@shapes\M@f@ntn@meb@se
      \else
        \M@NoBaseModeDetectedWarning{\M@f@ntn@me}
        \edef\M@f@ntn@meb@se{\M@f@ntn@me}
        \expandafter\edef\csname\M@f@ntn@me-base\endcsname{\M@f@ntn@me}
      \fi
    \fi
```

Now save the font families for reference later.

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

```
      {\M@f@ntn@me}
    \expandafter\edef\csname M@fontfamily@base@\@tempbase\endcsname
      {\M@f@ntn@meb@se}
  \else
```

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

```
    \edef@nospace\@tempa{\@tempa}
    \ifcsname M@fontfamily@\@tempa\endcsname
      \edef\M@f@ntn@me{\csname M@fontfamily@\@tempa\endcsname}
      \edef\M@f@ntn@meb@se{\csname M@fontfamily@base@\@tempa\endcsname}
    \else
```

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

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

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

```
        \ifdefined\directlua
          \edef\M@f@ntn@meb@se{\M@f@ntn@me-base}
          \wlog{Package mathfont Info: Adding \@tempbase with mode=base
            to the nfss!}
          \wlog{Family name: \M@f@ntn@meb@se}
          \DeclareFontFamily{TU}{\M@f@ntn@meb@se}{}
          \M@fill@nfss@shapes
            {\M@f@ntn@meb@se}{\M@otf@features}{\@tempfeatures;mode=base}
        \else
          \edef\M@f@ntn@meb@se{\M@f@ntn@me}
        \fi
      \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`.

```
        \wlog{Package mathfont Info: Passing \@tempbase\space to fontspec
          for handling!}
        \csname fontspec_set_family:Nnn\endcsname\M@f@ntn@me
          {\M@otf@features,\@tempfeatures}{\@tempbase}
        \ifdefined\directlua
          \wlog{Package mathfont Info: Passing \@tempbase\space with
            Renderer=Base to fontspec for handling!}
          \csname fontspec_set_family:Nnn\endcsname\M@f@ntn@meb@se
            {\M@otf@features,\@tempfeatures,Renderer=Base}{\@tempbase}
        \else
            \edef\M@f@ntn@meb@se{\M@f@ntn@me}
        \fi
      \fi
```

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

```
      \expandafter\edef\csname\M@f@ntn@me-base\endcsname{\M@f@ntn@meb@se}
      \expandafter\edef\csname M@fontfamily@\@tempa\endcsname
        {\M@f@ntn@me}
      \expandafter\edef\csname M@fontfamily@base@\@tempa\endcsname
        {\M@f@ntn@meb@se}
    \fi
  \fi}
```

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

```
\@onlypreamble\M@declare@shape
\@onlypreamble\M@fill@nfss@shapes
\@onlypreamble\M@newfont
```

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

5 Parse Input

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

```
\def\M@check@option@valid#1{%
  \let\@temperror\M@InvalidOptionError % error by default
  \@for\@j:=\M@keys\do{%
    \ifx\@j#1
      \let\@temperror\@gobble % eliminate error
      \@optionpresenttrue     % set switch to true
    \fi}
  \def\@j{empty} % if option is "empty," we do nothing
  \ifx\@j#1
    \let\@temperror\@gobble
    \@optionpresentfalse
  \fi
  \@temperror{#1}}
```

Do the same thing for the suboption.

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

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

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

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

```
  \ifx\@temp@opt\@empty
    \M@MissingOptionError
  \else
```

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

```
    \M@check@option@valid\@temp@opt
    \bgroup\def\@tempa{=}
    \ifx\@temp@sub\@tempa
      \egroup % first branch \egroup
      \M@MissingSuboptionError
    \else
      \egroup % second branch \egroup
```

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

```
      \ifx\@temp@sub\@empty
      \else
        \edef\@temp@sub{\expandafter\M@strip@equals\@temp@sub}
        \M@check@suboption@valid\@temp@sub % check that suboption is valid
      \fi
    \fi
```

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

```
    \bgroup\def\@tempa{roman}
    \ifx\@temp@sub\@tempa
      \egroup % first branch \egroup
      \def\@temp@sub{upright}
    \else
      \egroup % second branch \egroup
    \fi
  \fi}
```

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

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

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

6 Default Font Changes

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

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

The internal font-changing command.

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

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

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

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

```
    \M@newfont{#2}
    \ifcsname M@fontid@\M@f@ntn@me\endcsname\else % need new \M@count value?
      \expandafter\edef\csname M@fontid@\M@f@ntn@me\endcsname{\the
        \M@count}
      \expandafter\edef\csname M@fontid@\M@f@ntn@meb@se\endcsname{\the
        \M@count}
      \advance\M@count\@ne
    \fi
    \edef\@tempa{\csname M@fontid@\M@f@ntn@me\endcsname}
```

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

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

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

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

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

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

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

```
          \if@suboptionpresent
          \expandafter\edef\csname M@\@temp@opt shape\endcsname{\@temp@sub}
        \fi
        \edef\@tempc{\the\M@toks\space
          (\csname M@\@temp@opt shape\endcsname)}
        \M@toks\expandafter{\@tempc}
        \edef\@tempc{\csname M@\@temp@opt shape\endcsname}
```

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

```
        \def\@tempb{upright}
        \ifx\@tempb\@tempc
          \let\@tempb\shapedefault
        \else
          \let\@tempb\itdefault
        \fi
```

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

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

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

And now the magic happens!

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

Display concluding messages for the user.

```
  \edef\@tempa{\the\M@toks}
  \ifx\@tempa\@empty
    \wlog{The \string\mathfont\space command on line
      \the\inputlineno\space did not change the font for any characters!}
  \else
    \wlog{}
    \typeout{:: mathfont :: Using font \@tempbase\space
      on line \the\inputlineno.}
      \wlog{Character classes changed:\the\M@toks}
    \fi
  \fi}
\@onlypreamble\mathfont
\@onlypreamble\m@thf@nt
\@onlypreamble\@mathfont
```

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

```
\protected\def\setfont#1{%
  \mathfont{#1}
  \mathconstantsfont{#1}
  \setmathfontcommands{#1}
  \let\rmdefault\M@f@ntn@me}
\@onlypreamble\setfont
```

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

```
\let\M@SetMathConstants\relax
\protected\def\mathconstantsfont{%
  \@ifnextchar[{\@mathconstantsfont}{\@mathconstantsfont[upright]}}
\def\@mathconstantsfont[#1]#2{%
  \edef@nospace\@tempa{#2}
  \edef\@tempa{\csname M@fontfamily@base@\@tempa\endcsname}
  \expandafter\ifx\@tempa\relax
    \M@BadMathConstantsFontError{#2}
  \else
```

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

```
    \def\@tempb{#1}
    \def\@tempc{upright}
    \ifx\@tempb\@tempc
      \let\m@th@const@nts@font@sh@pe\shapedefault
    \else
      \def\@tempc{italic}
      \ifx\@tempb\@tempc
        \let\m@th@const@nts@font@sh@pe\itdefault
      \else
        \M@BadMathConstantsFontTypeError{#1}
      \fi
    \fi
    \ifcsname symM#1\csname M@fontid@\@tempa\endcsname\endcsname\else
      \DeclareSymbolFont{M#1\csname M@fontid@\@tempa\endcsname}
        {TU}{\@tempa}{\mddefault}{\m@th@const@nts@font@sh@pe}
    \fi
    \let\m@th@const@nts@font\@tempa
```

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

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

```
    \protected\def\M@SetMathConstants{%
      \begingroup
      \escapechar\m@ne
      \expandafter\getanddefine@fonts
        \csname symM#1\csname M@fontid@\m@th@const@nts@font\endcsname
          \expandafter
        \endcsname % expands to \symMupright<id>
        \csname TU/\m@th@const@nts@font
                 /\seriesdefault
                 /\m@th@const@nts@font@sh@pe
        \endcsname % expands to \TU/<nfss family name>/m/<shape>
      \globaldefs\@ne
      \expandafter\@gobbletwo\math@fonts % gobble to avoid infinite loop
      \endgroup}
    \fi
  \ifM@adjust@font\else
    \M@MathConstantsNoFontAdjustWarning
  \fi}
\def\math@fonts{\aftergroup\M@SetMathConstants}
\@onlypreamble\mathconstantsfont
```

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

7 Local Font Changes

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

```
\def\M@check@csarg#1#2{%
  \expandafter\ifx\expandafter\@nnil\@gobble#2\@nnil % good
    \ifcat\relax\noexpand#2 % good
      \M@arg@goodtrue
    \else % if #2 not a control sequence
      \M@MissingControlSequenceError#1{#2}
      \M@arg@goodfalse
    \fi
  \else % if #2 is multiple tokens
    \M@DoubleArgError#1{#2}
    \M@arg@goodfalse
  \fi}
```

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

```
\protected\def\newmathfontcommand#1#2#3#4{%
  \M@check@csarg\newmathfontcommand{#1}
  \ifM@arg@good
    \M@newfont{#2}
    \M@NewFontCommandInfo{#1}{\@tempbase}{\M@f@ntn@meb@se}{#3}{#4}
    \DeclareMathAlphabet{#1}{TU}{\M@f@ntn@meb@se}{#3}{#4}
  \fi}
\@onlypreamble\newmathfontcommand
```

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

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

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

```
\def\M@define@newmath@cmd#1#2#3{%
  \protected\def#1##1##2{%
    \M@check@csarg{#1}{##1}
    \newmathfontcommand{##1}{##2}{#2}{#3}}}
\def\M@default@newmath@cmds{%
  \newmathrm{\mddefault}{\shapedefault},%
  \newmathit{\mddefault}{\itdefault},%
  \newmathbf{\bfdefault}{\shapedefault},%
  \newmathbfit{\bfdefault}{\itdefault},%
  \newmathsc{\mddefault}{\scdefault},%
  \newmathscit{\mddefault}{\scdefault\itdefault},%
  \newmathbfsc{\bfdefault}{\scdefault},%
  \newmathbfscit{\bfdefault}{\scdefault\itdefault}}
\@for\@i:=\M@default@newmath@cmds\do{\expandafter\M@define@newmath@cmd\@i}
\@onlypreamble\newmathrm
\@onlypreamble\newmathit
\@onlypreamble\newmathbf
\@onlypreamble\newmathbfit
\@onlypreamble\newmathsc
\@onlypreamble\newmathscit
\@onlypreamble\newmathbfsc
\@onlypreamble\newmathbfscit
\@onlypreamble\M@define@newmath@cmd
\let\M@default@newmath@cmds\relax
```

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

```
\protected\def\setmathfontcommands#1{%
  \newmathrm\mathrm{#1}
  \newmathit\mathit{#1}
  \newmathbf\mathbf{#1}
  \newmathbfit\mathbfit{#1}
  \newmathsc\mathsc{#1}
  \newmathscit\mathscit{#1}
  \newmathbfsc\mathbfsc{#1}
  \newmathbfscit\mathbfscit{#1}}
\@onlypreamble\setmathfontcommands
```

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

```
\protected\def\newmathbold{%
  \M@DeprecatedWarning\newmathbold\newmathbf\newmathbf}
\protected\def\newmathboldit{%
  \M@DeprecatedWarning\newmathboldit\newmathbfit\newmathbfit}
```

8 Miscellaneous Material

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

```
\ifM@adjust@font
  \def\M@check@int#1{%
  \M@arg@goodfalse
  \begingroup
  \edef\@tempa{\number0#1}
  \edef\@tempa{\detokenize\expandafter{\@tempa}}
  \@expandtwoargs\in@{"}{\@tempa}
```

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

```
  \ifin@ % is " in #1?
    \endgroup % first branch \endgroup
  \else
    \@expandtwoargs\in@{\@backslashchar}{\@tempa}
    \ifin@ % is backslash in #1?
      \endgroup % second branch \endgroup
    \else
      \directlua{
      local num = tonumber("\@tempa")
      local bool = 0 % keep track if \@tempa is (int >= 0)
      if num then % if number?
        if num == num - (num \@percentchar 1) then % if integer?
          if num >= 0 then % if nonnegative?
            bool = 1
          end
        end
      end
      tex.print("\@backslashchar\@backslashchar endgroup")
      if bool == 1 then
        tex.print("\@backslashchar\@backslashchar csname M@arg@goodtrue%
        \@backslashchar\@backslashchar endcsname")
      end}
    \fi
  \fi}
```

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

```
  \let\@tempa~
  \let~\noexpand
  \count@\escapechar
  \escapechar\m@ne
  \@tfor\@i:=\RuleThicknessFactor\IntegralItalicFactor
```

```
    \SurdHorizontalFactor\SurdVerticalFactor\do{%
      \protected\expandafter\edef\@i#1{%
        ~\M@check@int{#1}%
        ~\ifM@arg@good
          ~\global
            \expandafter~\csname M@\expandafter\string\@i\endcsname=#1\relax
        ~\else
          ~\M@BadIntegerError\expandafter~\@i{#1}%
        ~\fi}}
  \let~\@tempa
  \escapechar\count@
```

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

```
\else
  \@tfor\@i:=\RuleThicknessFactor\IntegralItalicFactor\SurdHorizontalFactor
    \SurdVerticalFactor\CharmLine\CharmFile
      \do{%
        \protected\expandafter\edef\@i{\noexpand\M@NoFontAdjustError
          \expandafter\noexpand\@i
          \noexpand\@gobble}}
\fi
```

These commands should appear in the preamble only.

```
\@onlypreamble\RuleThicknessFactor
\@onlypreamble\IntegralItalicFactor
\@onlypreamble\SurdHorizontalFactor
\@onlypreamble\SurdVerticalFactor
\@onlypreamble\CharmLine
\@onlypreamble\CharmFile
```

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

```
\def\restoremathinternals{%
  \ifx\set@mathchar\@@set@mathchar
  \else
    \wlog{Package mathfont Info: Restoring \string\set@mathchar.}
    \wlog{Package mathfont Info: Restoring \string\set@mathsymbol.}
    \wlog{Package mathfont Info: Restoring \string\set@mathaccent.}
    \wlog{Package mathfont Info: Restoring \string\DeclareSymbolFont.}
    \let\set@mathchar\@@set@mathchar
    \let\set@mathsymbol\@@set@mathsymbol
    \let\set@mathaccent\@@set@mathaccent
    \let\DeclareSymbolFont\@@DeclareSymbolFont
  \fi}
```

Three macros used in defining `\simeq` and `\cong`. The construction is clunky and needs the intermediate macro `\st@ck@fl@trel` because `\mathchoice` is a bit of an odd macro. Instead of expanding to different replacement text depending on the math style, it fully typesets each of its four arguments and then takes the one corresponding to the correct style. A cleaner implementation would use `\mathstyle` from LuaTEX—perhaps in a future version.

```
  \protected\gdef\clap#1{\hb@xt@\z@{\hss#1\hss}}
  \protected\def\stack@flatrel#1#2{\expandafter
    \st@ck@fl@trel\expandafter#1\@firstofone#2}
  \protected\gdef\st@ck@fl@trel#1#2#3{%
    {\setbox0\hbox{$\m@th#1#2$}% contains \mathrel symbol
    \setbox1\hbox{$\m@th#1#3$}% gets raised over \box0
    \if\wd0>\wd1\relax
      \hb@xt@\wd0{%
        \hfil
        \clap{\raise0.7\ht0\box1}%
        \clap{\box0}\hfil}%
    \else
      \hb@xt@\wd1{%
        \hfil
        \clap{\raise0.7\ht0\box1}%
        \clap{\box0}\hfil}%
    \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.

```
\ifnum\tracinglostchars<\tw@
  \tracinglostchars\tw@
\fi
```

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

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

Now print the messages.

```
\AtBeginDocument{%
  \def\@tempa{%    <-- everything should be 14 characters long
    upper:\@spaces\@spaces,%
    lower:\@spaces\@spaces,%
    diacritics:\space\space\space,%
    greekupper:\space\space\space,%
    greeklower:\space\space\space,%
    agreekupper:\space\space,%
    agreeklower:\space\space,%
    cyrillicupper:,%
    cyrilliclower:,%
    hebrew:\@spaces\space\space\space,%
    digits:\@spaces\space\space\space,%
    operator:\@spaces\space,%
    delimiters:\space\space\space,%
    radical:\@spaces\space\space,%
    bigops:\@spaces\space\space\space,%
    extbigops:\@spaces,%
    symbols:\@spaces\space\space,%
    extsymbols:\space\space\space,%
    arrows:\@spaces\space\space\space,%
    bb:\@spaces\@spaces\space\space\space,%
    cal:\@spaces\@spaces\space\space,%
    frak:\@spaces\@spaces\space,%
    bcal:\@spaces\@spaces\space,%
    bfrak:\@spaces\@spaces}
  \wlog{^^JPackage mathfont Info: List of changes made in the preamble.}
  \@for\@i:=\@tempa\do{%
    \expandafter\keyword@info@begindocument\@i\@nil}
  \wlog{}}
```

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

```
\ifdefined\XeTeXrevision
  \ifM@font@loaded
    \AtEndOfPackage{%
    \PackageWarningNoLine{mathfont}
    {XeTeX detected. It looks like you\MessageBreak
    specified a font when you loaded\MessageBreak
    mathfont. If you run into problems\MessageBreak
    with a font whose name is multiple\MessageBreak
    words, try compiling with LuaLaTeX\MessageBreak
    or call \string\setfont\space or \string\mathfont\MessageBreak
    manually}}
  \fi
\fi
```

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

```
\ifM@font@loaded
  \AtEndOfPackage{\setfont\M@font@load}
\fi
```

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

```
\@onlypreamble\M@upper@set
\@onlypreamble\M@lower@set
\@onlypreamble\M@diacritics@set
```

```
\@onlypreamble\M@greekupper@set
\@onlypreamble\M@greeklower@set
\@onlypreamble\M@agreekupper@set
\@onlypreamble\M@agreeklower@set
\@onlypreamble\M@cyrillicupper@set
\@onlypreamble\M@cyrilliclower@set
\@onlypreamble\M@hebrew@set
\@onlypreamble\M@digits@set
\@onlypreamble\M@operator@set
\@onlypreamble\M@delimiters@set
\@onlypreamble\M@radical@set
\@onlypreamble\M@bigops@set
\@onlypreamble\M@extbigops@set
\@onlypreamble\M@symbols@set
\@onlypreamble\M@extsymbols@set
\@onlypreamble\M@arrows@set
\@onlypreamble\M@bb@set
\@onlypreamble\M@cal@set
\@onlypreamble\M@frak@set
\@onlypreamble\M@bcal@set
\@onlypreamble\M@bfrak@set
```

9 Adjust Fonts: Setup

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

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

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

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

Table 2: Fields of Character Subtables in `mathfont`

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

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

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

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

Table 3: Functions in `mathfont`

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

```
\ifM@adjust@font
\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.

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

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

```
\protected\def\CharmLine#1{%
  \begingroup
  \edef\@tempa{#1}
  \edef\@tempa{\detokenize\expandafter{\@tempa}}
  \@expandtwoargs\in@{"}{\@tempa}
```

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

```
  \ifin@ % is " in #1?
    \ifM@fromCharmFile
      \M@ForbiddenCharmFile{"}
    \else
      \M@ForbiddenCharmLine{"}
    \fi
  \else
    \@expandtwoargs\in@{\@backslashchar}{\@tempa}
    \ifin@ % is backslash in #1?
      \ifM@fromCharmFile
        \M@ForbiddenCharmFile{\@backslashchar}
      \else
        \M@ForbiddenCharmLine{\@backslashchar}
      \fi
    \else
```

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

```
      \directlua{mathfont:add_to_charm("\@tempa")}
    \fi
  \fi
  \endgroup}
```

The argument of `\CharmFile` should be a valid filename, and we open it in `\M@Charm`. The `\M@fromCharmFiletrue` command sets the boolean for an open charm file to true. This command and the corresponding false command are global because of how the kernel defines `\newif`. We can't check `\ifeof\M@Charm` because during processing of the last line from `\M@Charm`, we are at the end of the file even though it is still open.

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

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

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

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

```
  \@next
  \immediate\closein\M@Charm
  \M@fromCharmFilefalse
  \endgroup}
```

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

```
\directlua{
mathfont = {}
```

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

```
function mathfont:new_type_a(index, next, data)
  self[index] = {}
  self[index].type = "a"
  self[index].next = next
  self[index].left_stretch = data[1] / 1000
  self[index].right_stretch = data[2] / 1000
  self[index].top_accent_stretch = data[3] / 1000
  self[index].bot_accent_stretch = data[4] / 1000
end
```

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

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

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

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

```
  local v = \string# next
  self[index] = {}
  self[index].type = "e"
  self[index].smash = smash
  self[index].next = next
  self[index].total_variants = v
  self[index].data = {}
```

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

```
  for i = 1, v, 1 do
    self[index].data[i] = {}
    self[index].data[i].x = data[2*i-1] / 1000
    self[index].data[i].y = data[2*i] / 1000
  end
  self[index].top_accent_stretch = data[2*v+1] / 1000
  self[index].bot_accent_stretch = data[2*v+2] / 1000
end
```

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

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

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

```
function mathfont.parse_charm(charm_input)
  local split_string = {}
  local charm_string = charm_input
  local temp_string = ""
  local i = 1
```

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

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

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

```
  local sep = first_space
  if first_comma < first_space then
    sep = first_comma
  end
```

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

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

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

```
  if temp_string \noexpand~= "" then
    if tonumber(temp_string) then % if a number, append number
      split_string[i] = tonumber(temp_string)
      i = i+1
    elseif temp_string == "*" then % if asterisk, append asterisk
      split_string[i] = temp_string
      i = i+1
    else % if neither, raise error
      error(\M@number@ssert)
    end
  end
end
```

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

```
temp_string = charm_string
if temp_string \noexpand~= "" then
  if tonumber(temp_string) then % if a number, append number
    split_string[i] = tonumber(temp_string)
  elseif temp_string == "*" then % if asterisk, append asterisk
    split_string[i] = temp_string
  else % if neither, raise error
    error(\M@number@ssert)
  end
end
```

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

```
  local index = split_string[1]
  if index == "*" then
    error(\M@index@ssert)
  end
```

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

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

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

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

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

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

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

```
  if self[index].type == "a" then
    local entries_needed = 5
    assert(number_of_entries >= entries_needed, \M@entries@ssert)
    if charm_metrics[2] \noexpand~= "*" then
      self[index].left_stretch = charm_metrics[2] / 1000
    end
    if charm_metrics[3] \noexpand~= "*" then
      self[index].right_stretch = charm_metrics[3] / 1000
    end
    if charm_metrics[4] \noexpand~= "*" then
      self[index].top_accent_stretch = charm_metrics[4] / 1000
    end
    if charm_metrics[5] \noexpand~= "*" then
      self[index].bot_accent_stretch = charm_metrics[5] / 1000
    end
```

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

```
  elseif self[index].type == "e" then
    local tot_variants = self[index].total_variants
    local entries_needed = 2 * tot_variants + 3
    assert(number_of_entries >= entries_needed, \M@entries@ssert)
    for i = 1, tot_variants, 1 do
      if charm_metrics[2*i] \noexpand~= "*" then
        self[index].data[i].x = charm_metrics[2*i] / 1000
      end
      if charm_metrics[2*i+1] \noexpand~= "*" then
        self[index].data[i].y = charm_metrics[2*i+1] / 1000
      end
    end
```

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

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

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

```
  elseif self[index].type == "u" then
    local entries_needed = 3
    assert(number_of_entries >= entries_needed, \M@entries@ssert)
    if charm_metrics[2] \noexpand~= "*" then
      self[index].top_accent_stretch = charm_metrics[2] / 1000
    end
    if charm_metrics[3] \noexpand~= "*" then
      self[index].bot_accent_stretch = charm_metrics[3] / 1000
    end
  end
end
```

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

```
function mathfont.make_hex_value(integer)
  if integer == 0 then
    return "0000"
  end
  if integer == 1 then
    return "0001"
  end
  local hex_digits = "0123456789ABCDEF" % for reference
  local hex_string = ""
  local curr_val = integer
  local remainder = 0
```

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

```
  local i = 0
  while 16^i <= curr_val do
    i = i+1
  end
```

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

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

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

```
  if \string# hex_string < 4 then
    for i = \string# hex_string, 4, 1 do
      hex_string = "0" .. hex_string
    end
  end
  return hex_string
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.

```
function mathfont.glyph_info(char)
  local glyph_width = char.width or 0
  local glyph_height = char.height or 0
  local glyph_depth = char.depth or 0
  local glyph_italic = char.italic or 0
  return glyph_width, glyph_height, glyph_depth, glyph_italic
end
```

Table 4: Callbacks Created by mathfont

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

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

```
function mathfont:smash_glyph(index, fontdata)
  local smash_table = {}
  smash_table.width = 0
  smash_table.height = 0
  smash_table.depth = 0
  smash_table.commands = {{"char", index}}
  return smash_table
end
```

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

```
function mathfont.empty(arg)
end
```

10 Adjust Fonts: Changes

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

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

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

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

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

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

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

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

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

Incorporate the italic correction into the character width.

```
  width = width + italic
```

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.

```
  local offset = width * left_stretch
  a_table.width = width * (1 + left_stretch + right_stretch)
  a_table.height = height
  a_table.depth = depth
  a_table.italic = italic
  a_table.unicode = index
```

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

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

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

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

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

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

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

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

Add the commands to the table.

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

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

```
a_table.mathkern = {}
a_table.mathkern.top_right = {{height = 0, kern = -italic}}
a_table.mathkern.bottom_right = {{height = 0, kern = -italic}}
a_table.mathkern.top_left = {{height = 0, kern = 0}}
```

```
  a_table.mathkern.bottom_left = {{height = 0, kern = 0}}
  return a_table
end
```

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

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

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

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

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

```
    local horizontal_scale = tex.getcount("M@SurdHorizontalFactor") / 1000
    local vertical_scale = tex.getcount("M@SurdVerticalFactor") / 1000
    char.width = horizontal_scale * char.width
    char.height = vertical_scale * height
  end
```

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

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

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

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

The function for type `e` characters returns a table with different structure because we need to create multiple characters at once. Specifically, the function returns a table with one entry for each larger variant that we want to add to the font. Many of the variables are the same as in `:make_type_a`. We store the base character subtable in `char` and the font's `slant` parameter in `slant`. The `tounicode` stores the hexadecimal unicode value of the base character for reference later, and `smash_index` is the index of the unicode slot that we are using to hold the smashed version of the base character.

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

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

```
  for i = 1, charm_data.total_variants, 1 do
    local h_stretch = charm_data.data[i].x
    local v_stretch = charm_data.data[i].y
    local new_width = width * h_stretch
    local new_height = height * v_stretch
    local new_depth = depth * v_stretch
    local new_italic = italic * h_stretch
```

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

```
    e_table[i] = {}
    e_table[i].width = new_width
    e_table[i].height = new_height
    e_table[i].depth = new_depth
    e_table[i].italic = new_italic
```

Add the unicode information.

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

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

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

Add the commands.

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

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

```
    if i < charm_data.total_variants then
      e_table[i].next = charm_data.next[i+1]
    end
  end
  return e_table
end
```

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

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

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

```
  local base_top_accent = 0.5 * new_width + 0.5 * slant * height
  local base_bot_accent = 0.5 * new_width - 0.5 * slant * height
  local top_accent_shift = charm_data.top_accent_stretch * new_width
  local bot_accent_shift = charm_data.bot_accent_stretch * new_width
  u_table.top_accent = base_top_accent + top_accent_shift
  u_table.bot_accent = base_bot_accent + bot_accent_shift
```

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

```
  u_table.mathkern = {}
  u_table.mathkern.top_right = {{height = 0, kern = -italic}}
  u_table.mathkern.bottom_right = {{height = 0, kern = -italic}}
  u_table.mathkern.top_left = {{height = 0, kern = 0}}
  u_table.mathkern.bottom_left = {{height = 0, kern = 0}}
  return u_table
end
```

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

```
function mathfont.make_fake_angle(index, smash, fontdata)
  local temp = {}
  local lparen = fontdata.characters[40] or {}
  local lparen_height = lparen.height or 0
  local lparen_depth = lparen.depth or 0
  local glyph = fontdata.characters[index] or {}
  local glyph_height = glyph.height or 0
  local base_height = 0.9 * glyph_height
  local factor = 0
  if glyph_height \noexpand~= 0 then
    factor = (lparen_height + lparen_depth) / base_height
  end
  local shift = 0.1 * glyph_height * factor + lparen_depth
  temp.height = lparen_height
  temp.depth = lparen_depth
  temp.width = glyph.width or 0
  temp.italic = glyph.italic or 0
  temp.top_accent = glyph.top_accent or 0.5 * temp.width
  temp.bot_accent = glyph.bot_accent or 0.5 * temp.width
  temp.commands = {
    {"down", shift},
    {"pdf", "origin", string.format("q 1 0 0 \@percentchar s 0 0 cm", factor)},
    {"char", smash},
    {"pdf", "origin", "Q"},
    {"down", -shift}}
  return temp
end
```

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

```
function mathfont.set_nomath_true(fontdata)
  fontdata.nomath = false
  fontdata.oldmath = false
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.

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

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

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

Now add the characters to the font.

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

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

```
if not chars[8711] then
  chars[8710] = chars[8710] or {}
  chars[1044508] = mathfont:smash_glyph(8710, fontdata)
  chars[8711] = {}
  chars[8711].width = chars[8710].width or 0
  chars[8711].height = chars[8710].height or 0
  chars[8711].depth = chars[8710].depth or 0
  chars[8711].italic = chars[8710].italic or 0
  chars[8711].top_accent = chars[8710].top_accent or 0.5 * chars[8711].width
  chars[8711].bot_accent = chars[8710].bot_accent or 0.5 * chars[8711].width
  chars[8711].unicode = 8711
  chars[8711].tounicode = mathfont.make_hex_value(8711)
  chars[8711].commands = {
    {"down", -chars[8711].height},
    {"pdf", "origin", "q 1 0 0 -1 0 0 cm"},
    {"char", 1044508},
    {"pdf", "origin", "Q"},
    {"down", chars[8711].height}}
end
```

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

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

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

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

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

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

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

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

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

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

```
        chars[info.next[i]] = variants_table[i]
      end
```

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

```
    elseif info.type == "u" then
      chars[index] = mathfont:make_u_table(index, info, fontdata)
    end
  end
end
end
```

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

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

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

```
  local size = fontdata.size or 0
  local ex = fontdata.parameters.x_height or 0
  local em = fontdata.parameters.quad or 0
  local A_height = 0
  local y_depth = 0
  if fontdata.characters[65] then
    A_height = fontdata.characters[65].height or 0 % A
  end
  if fontdata.characters[121] then
    y_depth = fontdata.characters[121].depth or 0 % y
  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.

```
  local axis = 0
  local rule_thickness = 0
```

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

```
  local dim = "FractionRuleThickness"
  if not fontdata.MathConstants[dim] then
    local scale_factor = tex.getcount("M@RuleThicknessFactor") / 1000
    rule_thickness = (size / 18) * scale_factor
    fontdata.MathConstants[dim] = rule_thickness
  else
    rule_thickness = fontdata.MathConstants[dim]
  end
```

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

```
  local dim = "AxisHeight"
  if fontdata.MathConstants[dim] then
    axis = fontdata.MathConstants[dim]
  else
    if fontdata.characters[45] then
      axis = fontdata.characters[45].height - 0.5 * rule_thickness
    else
      axis = 0.8 * ex
    end
    fontdata.MathConstants[dim] = axis
  end
```

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

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

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

Our general approach for `\displaystyle` fractions is to place the baseline of the numerator numerator at a distance above the fraction rule of 1.5 times the rule height plus descender depth plus a small extra space. The minimum clearance will be the rule height, so we expect the numerator to strictly exceed the minimum clearance in most situations. Doing so produces consistent baselines of numerators and gives our value for σ_8, the attempted height of the numerator in `\displaystyle` fractions. For smaller styles, we use a single rule height as clearance, so we add $0.5 *$ `rule_thickness` $+$ `y_depth` scaled down by 0.7 to the rule thickness. The minimum clearance for numerator and denominator are separate OpenType parameters, and we set them later. The extra 0.1 A-height in the attempted clearance relative to the minimum clearance appears because we measure attempted clearance from the axis, whereas we measure minimum clearance from the top or bottom of the fraction rule.

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

```
local sigma_11 = (-axis + 1.5  * rule_thickness + 0.5 * y_depth +
  1.1 * A_height)
local sigma_12 = (-axis + 1.35 * rule_thickness + 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.

```
local sigma_13 = 0.6 * A_height     % attempted superscript height
local sigma_15 = 0.5 * A_height     % attempted superscript for \cramped
local sigma_16 = 1.1 * y_depth      % attempted subscript lower
if sigma_16 < 0.2 * A_height then
  sigma_16 = 0.2 * A_height
end
local sigma_17 = sigma_16           % sigma_16 when superscript present
local sigma_18 = 0.5 * A_height     % superscript lower for boxed subformula
local sigma_19 = 0.1 * A_height     % subscript lower for boxed subformula
```

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

```
local dim = "DisplayOperatorMinHeight"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = 1.8 * A_height
end
local dim = "FractionDelimiterDisplayStyleSize"
```

```
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = 2 * size
end
local dim = "FractionDelimiterSize"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = 1.3 * size
end
local dim = "FractionDenominatorDisplayStyleShiftDown"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = sigma_11
end
local dim = "FractionDenominatorShiftDown"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = sigma_12
end
```

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

```
local dim = "FractionDenominatorDisplayStyleGapMin"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = rule_thickness + 0.5 * y_depth
end
local dim = "FractionDenominatorGapMin"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = rule_thickness + 0.35 * y_depth
end
local dim = "FractionNumeratorDisplayStyleShiftUp"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = sigma_8
end
local dim = "FractionNumeratorShiftUp"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = sigma_9
end
local dim = "FractionNumeratorDisplayStyleGapMin"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = rule_thickness
end
local dim = "FractionNumeratorGapMin"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = rule_thickness
end
```

The SkewedFractionHorizontalGap and SkewedFractionVerticalGap take the values that LuaTEXwould set for a traditional TEX font.

```
local dim = "SkewedFractionHorizontalGap"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = 0.5 * em
end
local dim = "SkewedFractionVerticalGap"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = ex
end
```

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

```
local dim = "UpperLimitBaselineRiseMin"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = xi_11
end
local dim = "UpperLimitGapMin"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = xi_9
end
local dim = "LowerLimitBaselineDropMin"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = xi_12
end
local dim = "LowerLimitGapMin"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = xi_10
end
```

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

```
local dim = "StretchStackGapBelowMin"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = xi_10
end
local dim = "StretchStackTopShiftUp"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = xi_11
end
local dim = "StretchStackGapAboveMin"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = xi_9
end
local dim = "StretchStackBottomShiftDown"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = xi_12
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.

```
local dim = "OverbarExtraAscender"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = rule_thickness
end
local dim = "OverbarRuleThickness"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = rule_thickness
end
local dim = "OverbarVerticalGap"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = 2 * rule_thickness
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.

```
local dim = "RadicalExtraAscender"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = rule_thickness
end
local dim = "RadicalRuleThickness"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = rule_thickness
end
local dim = "RadicalDisplayStyleVerticalGap"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = 2 * rule_thickness
end
local dim = "RadicalVerticalGap"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = rule_thickness
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.

```
local dim = "RadicalKernBeforeDegree"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = (5/18) * em
end
local dim = "RadicalKernAfterDegree"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = (10/18) * em
end
local dim = "RadicalDegreeBottomRaisePercent"
```

```
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = 60
end
```

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

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

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

```
local dim = "StackBottomDisplayStyleShiftDown"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = sigma_11
end
local dim = "StackBottomShiftDown"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = sigma_12
end
local dim = "StackTopDisplayStyleShiftUp"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = sigma_8
end
local dim = "StackTopShiftUp"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = sigma_10
end
```

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

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

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

```
local dim = "StackDisplayStyleGapMin"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = 5 * rule_thickness + 0.5 * y_depth
end
local dim = "StackGapMin"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = 3 * rule_thickness + 0.35 * y_depth
end
```

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

```
local dim = "SubscriptShiftDown"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = sigma_16
```

```
end
local dim = "SubscriptBaselineDropMin"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = sigma_19
end
local dim = "SubscriptShiftDownWithSuperscript"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = sigma_17
end
```

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

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

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

```
local dim = "SubSuperscriptGapMin"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = rule_thickness
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.

```
local dim = "SuperscriptBottomMin"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = sigma_15 - 0.7 * y_depth
end
local dim = "SuperscriptBaselineDropMax"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = sigma_18
end
local dim = "SuperscriptShiftUp"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = sigma_13
end
local dim = "SuperscriptShiftUpCramped"
if not fontdata.MathConstants[dim] then
  fontdata.MathConstants[dim] = sigma_15
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 $-\sigma_{16} + 0.7 * \texttt{A\_height}$ 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.

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

```
  local dim = "UnderbarExtraDescender"
  if not fontdata.MathConstants[dim] then
    fontdata.MathConstants[dim] = rule_thickness
  end
  local dim = "UnderbarRuleThickness"
  if not fontdata.MathConstants[dim] then
    fontdata.MathConstants[dim] = rule_thickness
  end
  local dim = "UnderbarVerticalGap"
  if not fontdata.MathConstants[dim] then
    fontdata.MathConstants[dim] = 2 * rule_thickness
  end
```

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

```
  local dim = "MinConnectorOverlap"
  if not fontdata.MathConstants[dim] then
    fontdata.MathConstants[dim] = 0
  end
end
```

Time for callbacks! We create six of them.

```
luatexbase.create_callback("mathfont.inspect_font", "simple", mathfont.empty)
luatexbase.create_callback("mathfont.pre_adjust", "simple", mathfont.empty)
luatexbase.create_callback("mathfont.disable_nomath", "simple",
  mathfont.set_nomath_true)
luatexbase.create_callback("mathfont.add_math_constants", "simple",
  mathfont.math_constants)
luatexbase.create_callback("mathfont.fix_character_metrics", "simple",
  mathfont.apply_charm_info)
luatexbase.create_callback("mathfont.post_adjust", "simple", mathfont.empty)
```

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

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

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

```
function mathfont.adjust_font(fontdata)
  luatexbase.call_callback("mathfont.inspect_font", fontdata)
  if fontdata.nomath then
    mathfont.info("Adjusting font " .. mathfont.get_font_name(fontdata) .. ".")
    luatexbase.call_callback("mathfont.pre_adjust", fontdata)
    luatexbase.call_callback("mathfont.disable_nomath", fontdata)
    luatexbase.call_callback("mathfont.add_math_constants", fontdata)
    luatexbase.call_callback("mathfont.fix_character_metrics", fontdata)
    luatexbase.call_callback("mathfont.post_adjust", fontdata)
  else
    mathfont.info("No changes made to " ..
      mathfont.get_font_name(fontdata) .. ".")
  end
end
```

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

```
luatexbase.add_to_callback("luaotfload.patch_font", mathfont.adjust_font,
  "mathfont.adjust_font")
```

11 Adjust Fonts: Metrics

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

```
mathfont:new_type_a(97, 1044480, {50, 50, -50, 0})     % a
mathfont:new_type_a(98, 1044481, {50, 50, -50, 0})     % b
mathfont:new_type_a(99, 1044482, {50, 50, 0, 0})       % c
mathfont:new_type_a(100, 1044483, {50, -50, -50, 0})   % d
mathfont:new_type_a(101, 1044484, {50, 50, 0, 0})      % e
mathfont:new_type_a(102, 1044485, {200, 0, 0, 0})      % f
mathfont:new_type_a(103, 1044486, {100, 50, -50, 0})   % g
mathfont:new_type_a(104, 1044487, {50, 0, -50, 0})     % h
mathfont:new_type_a(105, 1044488, {50, 100, -100, 0})  % i
mathfont:new_type_a(106, 1044489, {400, 50, -50, 0})   % j
mathfont:new_type_a(107, 1044490, {50, 50, -100, 0})   % k
mathfont:new_type_a(108, 1044491, {100, 150, -100, 0}) % l
mathfont:new_type_a(109, 1044492, {50, 0, 0, 0})       % m
mathfont:new_type_a(110, 1044493, {50, 0, 0, 0})       % n
mathfont:new_type_a(111, 1044494, {50, 0, 0, 0})       % o
mathfont:new_type_a(112, 1044495, {200, 50, -50, 0})   % p
mathfont:new_type_a(113, 1044496, {50, 0, -50, 0})     % q
mathfont:new_type_a(114, 1044497, {100, 100, -50, 0})  % r
mathfont:new_type_a(115, 1044498, {50, 50, -50, 0})    % s
```

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

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

```
mathfont:new_type_a(65, 1044512, {50, 0, 150, 0})      % A
mathfont:new_type_a(66, 1044513, {50, 0, 0, 0})        % B
mathfont:new_type_a(67, 1044514, {0, 0, 0, 0})         % C
mathfont:new_type_a(68, 1044515, {50, 0, -50, 0})      % D
mathfont:new_type_a(69, 1044516, {50, 0, 0, 0})        % E
mathfont:new_type_a(70, 1044517, {50, 0, 0, 0})        % F
mathfont:new_type_a(71, 1044518, {0, 0, 0, 0})         % G
mathfont:new_type_a(72, 1044519, {50, 0, -50, 0})      % H
mathfont:new_type_a(73, 1044520, {100, 0, 0, 0})       % I
mathfont:new_type_a(74, 1044521, {50, 0, 100, 0})      % J
mathfont:new_type_a(75, 1044522, {50, 0, 0, 0})        % K
mathfont:new_type_a(76, 1044523, {50, 0, -180, 0})     % L
mathfont:new_type_a(77, 1044524, {50, 0, -50, 0})      % M
mathfont:new_type_a(78, 1044525, {50, 0, -50, 0})      % N
mathfont:new_type_a(79, 1044526, {0, 0, 0, 0})         % O
mathfont:new_type_a(80, 1044527, {0, 0, -50, 0})       % P
mathfont:new_type_a(81, 1044528, {0, 50, 0, 0})        % Q
mathfont:new_type_a(82, 1044529, {50, 0, -50, 0})      % R
mathfont:new_type_a(83, 1044530, {0, 0, -50, 0})       % S
mathfont:new_type_a(84, 1044531, {0, 0, -50, 0})       % T
mathfont:new_type_a(85, 1044532, {0, 0, -50, 0})       % U
mathfont:new_type_a(86, 1044533, {0, 50, 0, 0})        % V
mathfont:new_type_a(87, 1044534, {0, 50, -50, 0})      % W
mathfont:new_type_a(88, 1044535, {50, 0, 0, 0})        % X
mathfont:new_type_a(89, 1044536, {0, 0, -50, 0})       % Y
mathfont:new_type_a(90, 1044537, {50, 0, -50, 0})      % Z
```

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

```
mathfont:new_type_u(945, {0, 0})    % \alpha
mathfont:new_type_u(946, {0, 0})    % \beta
mathfont:new_type_u(947, {-50, 0})  % \gamma
mathfont:new_type_u(948, {0, 0})    % \delta
mathfont:new_type_u(1013, {50, 0})  % \epsilon
mathfont:new_type_u(950, {0, 0})    % \zeta
mathfont:new_type_u(951, {-50, 0})  % \eta
mathfont:new_type_u(952, {0, 0})    % \theta
```

```
mathfont:new_type_u(953, {-50, 0})  % \iota
mathfont:new_type_u(954, {0, 0})    % \kappa
mathfont:new_type_u(955, {-150, 0}) % lambda
mathfont:new_type_u(956, {0, 0})    % \mu
mathfont:new_type_u(957, {-50, 0})  % \nu
mathfont:new_type_u(958, {0, 0})    % \xi
mathfont:new_type_u(959, {0, 0})    % \omicron
mathfont:new_type_u(960, {-100, 0}) % \pi
mathfont:new_type_u(961, {-50, 0})  % \rho
mathfont:new_type_u(963, {-100, 0}) % \sigma
mathfont:new_type_u(964, {-100, 0}) % \tau
mathfont:new_type_u(965, {-50, 0})  % \upsilon
mathfont:new_type_u(981, {0, 0})    % \phi
mathfont:new_type_u(967, {-50, 0})  % \chi
mathfont:new_type_u(968, {-50, 0})  % \psi
mathfont:new_type_u(969, {0, 0})    % \omega
mathfont:new_type_u(976, {0, 0})    % \varbeta
mathfont:new_type_u(949, {-50, 0})  % \varepsilon
mathfont:new_type_u(977, {50, 0})   % \vartheta
mathfont:new_type_u(1009, {-50, 0}) % \varrho
mathfont:new_type_u(962, {-50, 0})  % \varsigma
mathfont:new_type_u(966, {0, 0})    % \varphi
```

Upper-case Greek characters. Same as previously.

```
mathfont:new_type_u(913, {0, 0})    % \Alpha
mathfont:new_type_u(914, {0, 0})    % \Beta
mathfont:new_type_u(915, {0, 0})    % \Gamma
mathfont:new_type_u(916, {0, 0})    % \Delta
mathfont:new_type_u(917, {0, 0})    % \Epsilon
mathfont:new_type_u(918, {0, 0})    % \Zeta
mathfont:new_type_u(919, {0, 0})    % \Eta
mathfont:new_type_u(920, {0, 0})    % \Theta
mathfont:new_type_u(921, {0, 0})    % \Iota
mathfont:new_type_u(922, {0, 0})    % \Kappa
mathfont:new_type_u(923, {0, 0})    % \Lambda
mathfont:new_type_u(924, {0, 0})    % \Mu
mathfont:new_type_u(925, {0, 0})    % \Nu
mathfont:new_type_u(926, {0, 0})    % \Xi
mathfont:new_type_u(927, {0, 0})    % \Omicron
mathfont:new_type_u(928, {0, 0})    % \Pi
mathfont:new_type_u(929, {0, 0})    % \Rho
mathfont:new_type_u(931, {0, 0})    % \Sigma
mathfont:new_type_u(932, {0, 0})    % \Tau
mathfont:new_type_u(933, {0, 0})    % \Upsilon
mathfont:new_type_u(934, {0, 0})    % \Phi
mathfont:new_type_u(935, {0, 0})    % \Chi
mathfont:new_type_u(936, {0, 0})    % \Psi
mathfont:new_type_u(937, {0, 0})    % \Omega
```

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

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

```
local delim_glyphs = {40, % (
  41,      % )
  47,      % /
  91,      % [
  92,      % backslash
  93,      % ]
  123,     % {
  125,     % }
  8249,    % \lguil
  8250,    % \rguil
  171,     % \llguil
  187,     % \rrguil
  1044508, % \fakelangle
  1044509, % \fakerangle
  1044510, % \fakellangle
  1044511} % \fakerrangle
local big_op_glyphs = {33, % !
  35,      % #
  36,      % $
  37,      % %
  38,      % &
  43,      % +
  63,      % ?
  64,      % @
  167,     % \S
  215,     % \times
  247,     % \div
  8719,    % \prod
  8721,    % \sum
  8720,    % \coprod
  8897,    % \bigvee
  8896,    % \bigwedge
  8899,    % \bigcup
  8898,    % \bigcap
  10753,   % \bigoplus
  10754,   % \bigotimes
  10752,   % \bigodot
  10757,   % \bigsqcap
  10758}   % \bigsqcup
local vert_glyphs = {124, 8730} % | and \surd
local int_glyphs = {8747, % \intop
```

```
  8748,    % \iint
  8749,    % \iiint
  8750,    % \oint
  8751,    % \oiint
  8752}    % \oiiint
```

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

```
local smash = 1044544
```

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

```
local delim_scale = {}
local big_op_scale = {}
local vert_scale = {}
local int_scale = {}
```

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

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

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

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

We do not modify accent placement.

```
delim_scale[31] = 0
delim_scale[32] = 0
big_op_scale[31] = 0
big_op_scale[32] = 0
vert_scale[31] = 0
vert_scale[32] = 0
int_scale[31] = 0
int_scale[32] = 0
```

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

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

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

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

Add the charm information and increment `smash`.

```
    self:new_type_e(curr_char, curr_smash, curr_slots, scale_list)
    smash = smash + variants + 1
    curr_smash = smash
  end
end
```

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

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

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

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

12 Unicode Hex Values

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

```
\ifM@adjust@font
  \def\M@upper@set{%
    \edef\M@upper@font{M\M@uppershape\@tempa}
    \DeclareMathSymbol{A}{\mathalpha}{\M@upper@font}{1044512}
    \DeclareMathSymbol{B}{\mathalpha}{\M@upper@font}{1044513}
    \DeclareMathSymbol{C}{\mathalpha}{\M@upper@font}{1044514}
    \DeclareMathSymbol{D}{\mathalpha}{\M@upper@font}{1044515}
    \DeclareMathSymbol{E}{\mathalpha}{\M@upper@font}{1044516}
    \DeclareMathSymbol{F}{\mathalpha}{\M@upper@font}{1044517}
    \DeclareMathSymbol{G}{\mathalpha}{\M@upper@font}{1044518}
    \DeclareMathSymbol{H}{\mathalpha}{\M@upper@font}{1044519}
    \DeclareMathSymbol{I}{\mathalpha}{\M@upper@font}{1044520}
    \DeclareMathSymbol{J}{\mathalpha}{\M@upper@font}{1044521}
    \DeclareMathSymbol{K}{\mathalpha}{\M@upper@font}{1044522}
    \DeclareMathSymbol{L}{\mathalpha}{\M@upper@font}{1044523}
    \DeclareMathSymbol{M}{\mathalpha}{\M@upper@font}{1044524}
    \DeclareMathSymbol{N}{\mathalpha}{\M@upper@font}{1044525}
    \DeclareMathSymbol{O}{\mathalpha}{\M@upper@font}{1044526}
    \DeclareMathSymbol{P}{\mathalpha}{\M@upper@font}{1044527}
    \DeclareMathSymbol{Q}{\mathalpha}{\M@upper@font}{1044528}
    \DeclareMathSymbol{R}{\mathalpha}{\M@upper@font}{1044529}
    \DeclareMathSymbol{S}{\mathalpha}{\M@upper@font}{1044530}
    \DeclareMathSymbol{T}{\mathalpha}{\M@upper@font}{1044531}
    \DeclareMathSymbol{U}{\mathalpha}{\M@upper@font}{1044532}
    \DeclareMathSymbol{V}{\mathalpha}{\M@upper@font}{1044533}
    \DeclareMathSymbol{W}{\mathalpha}{\M@upper@font}{1044534}
    \DeclareMathSymbol{X}{\mathalpha}{\M@upper@font}{1044535}
    \DeclareMathSymbol{Y}{\mathalpha}{\M@upper@font}{1044536}
    \DeclareMathSymbol{Z}{\mathalpha}{\M@upper@font}{1044537}}
\else
  \def\M@upper@set{%
    \edef\M@upper@font{M\M@uppershape\@tempa}
    \DeclareMathSymbol{A}{\mathalpha}{\M@upper@font}{`A}
    \DeclareMathSymbol{B}{\mathalpha}{\M@upper@font}{`B}
    \DeclareMathSymbol{C}{\mathalpha}{\M@upper@font}{`C}
    \DeclareMathSymbol{D}{\mathalpha}{\M@upper@font}{`D}
    \DeclareMathSymbol{E}{\mathalpha}{\M@upper@font}{`E}
    \DeclareMathSymbol{F}{\mathalpha}{\M@upper@font}{`F}
    \DeclareMathSymbol{G}{\mathalpha}{\M@upper@font}{`G}
    \DeclareMathSymbol{H}{\mathalpha}{\M@upper@font}{`H}
    \DeclareMathSymbol{I}{\mathalpha}{\M@upper@font}{`I}
    \DeclareMathSymbol{J}{\mathalpha}{\M@upper@font}{`J}
    \DeclareMathSymbol{K}{\mathalpha}{\M@upper@font}{`K}
    \DeclareMathSymbol{L}{\mathalpha}{\M@upper@font}{`L}
    \DeclareMathSymbol{M}{\mathalpha}{\M@upper@font}{`M}
    \DeclareMathSymbol{N}{\mathalpha}{\M@upper@font}{`N}
    \DeclareMathSymbol{O}{\mathalpha}{\M@upper@font}{`O}
    \DeclareMathSymbol{P}{\mathalpha}{\M@upper@font}{`P}
    \DeclareMathSymbol{Q}{\mathalpha}{\M@upper@font}{`Q}
    \DeclareMathSymbol{R}{\mathalpha}{\M@upper@font}{`R}
    \DeclareMathSymbol{S}{\mathalpha}{\M@upper@font}{`S}
    \DeclareMathSymbol{T}{\mathalpha}{\M@upper@font}{`T}
    \DeclareMathSymbol{U}{\mathalpha}{\M@upper@font}{`U}
    \DeclareMathSymbol{V}{\mathalpha}{\M@upper@font}{`V}
    \DeclareMathSymbol{W}{\mathalpha}{\M@upper@font}{`W}
    \DeclareMathSymbol{X}{\mathalpha}{\M@upper@font}{`X}
    \DeclareMathSymbol{Y}{\mathalpha}{\M@upper@font}{`Y}
    \DeclareMathSymbol{Z}{\mathalpha}{\M@upper@font}{`Z}}
\fi
```

Set lower-case Latin characters.

```
\ifM@adjust@font
  \def\M@lower@set{%
    \edef\M@lower@font{M\M@lowershape\@tempa}
    \DeclareMathSymbol{a}{\mathalpha}{\M@lower@font}{1044480}
    \DeclareMathSymbol{b}{\mathalpha}{\M@lower@font}{1044481}
    \DeclareMathSymbol{c}{\mathalpha}{\M@lower@font}{1044482}
    \DeclareMathSymbol{d}{\mathalpha}{\M@lower@font}{1044483}
    \DeclareMathSymbol{e}{\mathalpha}{\M@lower@font}{1044484}
    \DeclareMathSymbol{f}{\mathalpha}{\M@lower@font}{1044485}
    \DeclareMathSymbol{g}{\mathalpha}{\M@lower@font}{1044486}
    \DeclareMathSymbol{h}{\mathalpha}{\M@lower@font}{1044487}
    \DeclareMathSymbol{i}{\mathalpha}{\M@lower@font}{1044488}
    \DeclareMathSymbol{j}{\mathalpha}{\M@lower@font}{1044489}
    \DeclareMathSymbol{k}{\mathalpha}{\M@lower@font}{1044490}
    \DeclareMathSymbol{l}{\mathalpha}{\M@lower@font}{1044491}
    \DeclareMathSymbol{m}{\mathalpha}{\M@lower@font}{1044492}
    \DeclareMathSymbol{n}{\mathalpha}{\M@lower@font}{1044493}
    \DeclareMathSymbol{o}{\mathalpha}{\M@lower@font}{1044494}
    \DeclareMathSymbol{p}{\mathalpha}{\M@lower@font}{1044495}
    \DeclareMathSymbol{q}{\mathalpha}{\M@lower@font}{1044496}
    \DeclareMathSymbol{r}{\mathalpha}{\M@lower@font}{1044497}
    \DeclareMathSymbol{s}{\mathalpha}{\M@lower@font}{1044498}
    \DeclareMathSymbol{t}{\mathalpha}{\M@lower@font}{1044499}
    \DeclareMathSymbol{u}{\mathalpha}{\M@lower@font}{1044500}
    \DeclareMathSymbol{v}{\mathalpha}{\M@lower@font}{1044501}
    \DeclareMathSymbol{w}{\mathalpha}{\M@lower@font}{1044502}
    \DeclareMathSymbol{x}{\mathalpha}{\M@lower@font}{1044503}
    \DeclareMathSymbol{y}{\mathalpha}{\M@lower@font}{1044504}
    \DeclareMathSymbol{z}{\mathalpha}{\M@lower@font}{1044505}
    \DeclareMathSymbol{\imath}{\mathalpha}{\M@lower@font}{1044506}
    \DeclareMathSymbol{\jmath}{\mathalpha}{\M@lower@font}{1044507}
    \DeclareMathSymbol{\hbar}{\mathord}{\M@lower@font}{"127}}
\else
  \def\M@lower@set{%
    \edef\M@lower@font{M\M@lowershape\@tempa}
    \DeclareMathSymbol{a}{\mathalpha}{\M@lower@font}{`a}
    \DeclareMathSymbol{b}{\mathalpha}{\M@lower@font}{`b}
    \DeclareMathSymbol{c}{\mathalpha}{\M@lower@font}{`c}
    \DeclareMathSymbol{d}{\mathalpha}{\M@lower@font}{`d}
    \DeclareMathSymbol{e}{\mathalpha}{\M@lower@font}{`e}
    \DeclareMathSymbol{f}{\mathalpha}{\M@lower@font}{`f}
    \DeclareMathSymbol{g}{\mathalpha}{\M@lower@font}{`g}
    \DeclareMathSymbol{h}{\mathalpha}{\M@lower@font}{`h}
    \DeclareMathSymbol{i}{\mathalpha}{\M@lower@font}{`i}
    \DeclareMathSymbol{j}{\mathalpha}{\M@lower@font}{`j}
    \DeclareMathSymbol{k}{\mathalpha}{\M@lower@font}{`k}
    \DeclareMathSymbol{l}{\mathalpha}{\M@lower@font}{`l}
    \DeclareMathSymbol{m}{\mathalpha}{\M@lower@font}{`m}
    \DeclareMathSymbol{n}{\mathalpha}{\M@lower@font}{`n}
    \DeclareMathSymbol{o}{\mathalpha}{\M@lower@font}{`o}
    \DeclareMathSymbol{p}{\mathalpha}{\M@lower@font}{`p}
    \DeclareMathSymbol{q}{\mathalpha}{\M@lower@font}{`q}
    \DeclareMathSymbol{r}{\mathalpha}{\M@lower@font}{`r}
    \DeclareMathSymbol{s}{\mathalpha}{\M@lower@font}{`s}
    \DeclareMathSymbol{t}{\mathalpha}{\M@lower@font}{`t}
    \DeclareMathSymbol{u}{\mathalpha}{\M@lower@font}{`u}
    \DeclareMathSymbol{v}{\mathalpha}{\M@lower@font}{`v}
    \DeclareMathSymbol{w}{\mathalpha}{\M@lower@font}{`w}
    \DeclareMathSymbol{x}{\mathalpha}{\M@lower@font}{`x}
    \DeclareMathSymbol{y}{\mathalpha}{\M@lower@font}{`y}
    \DeclareMathSymbol{z}{\mathalpha}{\M@lower@font}{`z}
    \DeclareMathSymbol{\imath}{\mathalpha}{\M@lower@font}{"131}
    \DeclareMathSymbol{\jmath}{\mathalpha}{\M@lower@font}{"237}
    \DeclareMathSymbol{\hbar}{\mathord}{\M@lower@font}{"127}}
\fi
```

Set diacritics.

```
\def\M@diacritics@set{%
  \edef\M@diacritics@font{M\M@diacriticsshape\@tempa}
  \DeclareMathAccent{\acute}{\mathalpha}{\M@diacritics@font}{"B4}
  \DeclareMathAccent{\aacute}{\mathalpha}{\M@diacritics@font}{"2DD}
  \DeclareMathAccent{\dot}{\mathalpha}{\M@diacritics@font}{"2D9}
  \DeclareMathAccent{\ddot}{\mathalpha}{\M@diacritics@font}{"A8}
  \DeclareMathAccent{\grave}{\mathalpha}{\M@diacritics@font}{"60}
  \DeclareMathAccent{\breve}{\mathalpha}{\M@diacritics@font}{"2D8}
  \DeclareMathAccent{\hat}{\mathalpha}{\M@diacritics@font}{"2C6}
  \DeclareMathAccent{\check}{\mathalpha}{\M@diacritics@font}{"2C7}
  \DeclareMathAccent{\bar}{\mathalpha}{\M@diacritics@font}{"2C9}
  \DeclareMathAccent{\mathring}{\mathalpha}{\M@diacritics@font}{"2DA}
  \DeclareMathAccent{\tilde}{\mathalpha}{\M@diacritics@font}{"2DC}}
```

Set capital Greek characters.

```
\def\M@greekupper@set{%
  \edef\M@greekupper@font{M\M@greekuppershape\@tempa}
  \DeclareMathSymbol{\Alpha}{\mathalpha}{\M@greekupper@font}{"391}
  \DeclareMathSymbol{\Beta}{\mathalpha}{\M@greekupper@font}{"392}
  \DeclareMathSymbol{\Gamma}{\mathalpha}{\M@greekupper@font}{"393}
  \DeclareMathSymbol{\Delta}{\mathalpha}{\M@greekupper@font}{"394}
  \DeclareMathSymbol{\Epsilon}{\mathalpha}{\M@greekupper@font}{"395}
  \DeclareMathSymbol{\Zeta}{\mathalpha}{\M@greekupper@font}{"396}
  \DeclareMathSymbol{\Eta}{\mathalpha}{\M@greekupper@font}{"397}
  \DeclareMathSymbol{\Theta}{\mathalpha}{\M@greekupper@font}{"398}
  \DeclareMathSymbol{\Iota}{\mathalpha}{\M@greekupper@font}{"399}
  \DeclareMathSymbol{\Kappa}{\mathalpha}{\M@greekupper@font}{"39A}
  \DeclareMathSymbol{\Lambda}{\mathalpha}{\M@greekupper@font}{"39B}
  \DeclareMathSymbol{\Mu}{\mathalpha}{\M@greekupper@font}{"39C}
  \DeclareMathSymbol{\Nu}{\mathalpha}{\M@greekupper@font}{"39D}
  \DeclareMathSymbol{\Xi}{\mathalpha}{\M@greekupper@font}{"39E}
  \DeclareMathSymbol{\Omicron}{\mathalpha}{\M@greekupper@font}{"39F}
  \DeclareMathSymbol{\Pi}{\mathalpha}{\M@greekupper@font}{"3A0}
  \DeclareMathSymbol{\Rho}{\mathalpha}{\M@greekupper@font}{"3A1}
  \DeclareMathSymbol{\Sigma}{\mathalpha}{\M@greekupper@font}{"3A3}
  \DeclareMathSymbol{\Tau}{\mathalpha}{\M@greekupper@font}{"3A4}
  \DeclareMathSymbol{\Upsilon}{\mathalpha}{\M@greekupper@font}{"3A5}
  \DeclareMathSymbol{\Phi}{\mathalpha}{\M@greekupper@font}{"3A6}
  \DeclareMathSymbol{\Chi}{\mathalpha}{\M@greekupper@font}{"3A7}
  \DeclareMathSymbol{\Psi}{\mathalpha}{\M@greekupper@font}{"3A8}
  \DeclareMathSymbol{\Omega}{\mathalpha}{\M@greekupper@font}{"3A9}
  \DeclareMathSymbol{\varTheta}{\mathalpha}{\M@greekupper@font}{"3F4}
```

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

```
  \ifM@adjust@font
    \ifM@symbols\else
      \DeclareMathSymbol{\increment}{\mathord}{\M@greekupper@font}{"2206}
      \DeclareMathSymbol{\nabla}{\mathord}{\M@greekupper@font}{"2207}
    \fi
  \else
    \ifM@symbols\else
      \DeclareMathSymbol{\increment}{\mathord}{\M@greekupper@font}{"2206}
    \fi
    \ifM@extsymbols\else
      \DeclareMathSymbol{\nabla}{\mathord}{\M@greekupper@font}{"2207}
    \fi
  \fi}
```

Set minuscule Greek characters.

```
\def\M@greeklower@set{%
  \edef\M@greeklower@font{M\M@greeklowershape\@tempa}
  \DeclareMathSymbol{\alpha}{\mathalpha}{\M@greeklower@font}{"3B1}
  \DeclareMathSymbol{\beta}{\mathalpha}{\M@greeklower@font}{"3B2}
  \DeclareMathSymbol{\gamma}{\mathalpha}{\M@greeklower@font}{"3B3}
  \DeclareMathSymbol{\delta}{\mathalpha}{\M@greeklower@font}{"3B4}
  \DeclareMathSymbol{\epsilon}{\mathalpha}{\M@greeklower@font}{"3B5}
  \DeclareMathSymbol{\zeta}{\mathalpha}{\M@greeklower@font}{"3B6}
  \DeclareMathSymbol{\eta}{\mathalpha}{\M@greeklower@font}{"3B7}
  \DeclareMathSymbol{\theta}{\mathalpha}{\M@greeklower@font}{"3B8}
  \DeclareMathSymbol{\iota}{\mathalpha}{\M@greeklower@font}{"3B9}
  \DeclareMathSymbol{\kappa}{\mathalpha}{\M@greeklower@font}{"3BA}
  \DeclareMathSymbol{\lambda}{\mathalpha}{\M@greeklower@font}{"3BB}
  \DeclareMathSymbol{\mu}{\mathalpha}{\M@greeklower@font}{"3BC}
  \DeclareMathSymbol{\nu}{\mathalpha}{\M@greeklower@font}{"3BD}
  \DeclareMathSymbol{\xi}{\mathalpha}{\M@greeklower@font}{"3BE}
  \DeclareMathSymbol{\omicron}{\mathalpha}{\M@greeklower@font}{"3BF}
  \DeclareMathSymbol{\pi}{\mathalpha}{\M@greeklower@font}{"3C0}
  \DeclareMathSymbol{\rho}{\mathalpha}{\M@greeklower@font}{"3C1}
  \DeclareMathSymbol{\sigma}{\mathalpha}{\M@greeklower@font}{"3C3}
  \DeclareMathSymbol{\tau}{\mathalpha}{\M@greeklower@font}{"3C4}
  \DeclareMathSymbol{\upsilon}{\mathalpha}{\M@greeklower@font}{"3C5}
  \DeclareMathSymbol{\phi}{\mathalpha}{\M@greeklower@font}{"3C6}
  \DeclareMathSymbol{\chi}{\mathalpha}{\M@greeklower@font}{"3C7}
  \DeclareMathSymbol{\psi}{\mathalpha}{\M@greeklower@font}{"3C8}
  \DeclareMathSymbol{\omega}{\mathalpha}{\M@greeklower@font}{"3C9}
  \DeclareMathSymbol{\varbeta}{\mathalpha}{\M@greeklower@font}{"3D0}
  \DeclareMathSymbol{\varepsilon}{\mathalpha}{\M@greeklower@font}{"3F5}
  \DeclareMathSymbol{\varkappa}{\mathalpha}{\M@greeklower@font}{"3F0}
  \DeclareMathSymbol{\vartheta}{\mathalpha}{\M@greeklower@font}{"3D1}
  \DeclareMathSymbol{\varrho}{\mathalpha}{\M@greeklower@font}{"3F1}
  \DeclareMathSymbol{\varsigma}{\mathalpha}{\M@greeklower@font}{"3C2}
  \DeclareMathSymbol{\varphi}{\mathalpha}{\M@greeklower@font}{"3D5}}
```

Set capital ancient Greek characters.

```
\def\M@agreekupper@set{%
  \edef\M@agreekupper@font{M\M@agreekuppershape\@tempa}
  \DeclareMathSymbol{\Heta}{\mathalpha}{\M@agreekupper@font}{"370}
  \DeclareMathSymbol{\Sampi}{\mathalpha}{\M@agreekupper@font}{"3E0}
  \DeclareMathSymbol{\Digamma}{\mathalpha}{\M@agreekupper@font}{"3DC}
  \DeclareMathSymbol{\Koppa}{\mathalpha}{\M@agreekupper@font}{"3D8}
  \DeclareMathSymbol{\Stigma}{\mathalpha}{\M@agreekupper@font}{"3DA}
  \DeclareMathSymbol{\Sho}{\mathalpha}{\M@agreekupper@font}{"3F7}
  \DeclareMathSymbol{\San}{\mathalpha}{\M@agreekupper@font}{"3FA}
  \DeclareMathSymbol{\varSampi}{\mathalpha}{\M@agreekupper@font}{"372}
  \DeclareMathSymbol{\varDigamma}{\mathalpha}{\M@agreekupper@font}{"376}
  \DeclareMathSymbol{\varKoppa}{\mathalpha}{\M@agreekupper@font}{"3DE}}
```

Set minuscule ancient Greek characters.

```
\def\M@agreeklower@set{%
  \edef\M@agreeklower@font{M\M@agreeklowershape\@tempa}
  \DeclareMathSymbol{\heta}{\mathalpha}{\M@agreeklower@font}{"371}
  \DeclareMathSymbol{\sampi}{\mathalpha}{\M@agreeklower@font}{"3E1}
  \DeclareMathSymbol{\digamma}{\mathalpha}{\M@agreeklower@font}{"3DD}
  \DeclareMathSymbol{\koppa}{\mathalpha}{\M@agreeklower@font}{"3D9}
  \DeclareMathSymbol{\stigma}{\mathalpha}{\M@agreeklower@font}{"3DB}
  \DeclareMathSymbol{\sho}{\mathalpha}{\M@agreeklower@font}{"3F8}
  \DeclareMathSymbol{\san}{\mathalpha}{\M@agreeklower@font}{"3FB}
  \DeclareMathSymbol{\varsampi}{\mathalpha}{\M@agreeklower@font}{"373}
  \DeclareMathSymbol{\vardigamma}{\mathalpha}{\M@agreeklower@font}{"377}
  \DeclareMathSymbol{\varkoppa}{\mathalpha}{\M@agreeklower@font}{"3DF}}
```

Set capital Cyrillic characters.

```
\def\M@cyrillicupper@set{%
  \edef\M@cyrillicupper@font{M\M@cyrillicuppershape\@tempa}
  \DeclareMathSymbol{\cyrA}{\mathalpha}{\M@cyrillicupper@font}{"410}
  \DeclareMathSymbol{\cyrBe}{\mathalpha}{\M@cyrillicupper@font}{"411}
  \DeclareMathSymbol{\cyrVe}{\mathalpha}{\M@cyrillicupper@font}{"412}
  \DeclareMathSymbol{\cyrGhe}{\mathalpha}{\M@cyrillicupper@font}{"413}
  \DeclareMathSymbol{\cyrDe}{\mathalpha}{\M@cyrillicupper@font}{"414}
  \DeclareMathSymbol{\cyrIe}{\mathalpha}{\M@cyrillicupper@font}{"415}
  \DeclareMathSymbol{\cyrZhe}{\mathalpha}{\M@cyrillicupper@font}{"416}
  \DeclareMathSymbol{\cyrZe}{\mathalpha}{\M@cyrillicupper@font}{"417}
  \DeclareMathSymbol{\cyrI}{\mathalpha}{\M@cyrillicupper@font}{"418}
  \DeclareMathSymbol{\cyrKa}{\mathalpha}{\M@cyrillicupper@font}{"41A}
  \DeclareMathSymbol{\cyrEl}{\mathalpha}{\M@cyrillicupper@font}{"41B}
  \DeclareMathSymbol{\cyrEm}{\mathalpha}{\M@cyrillicupper@font}{"41C}
  \DeclareMathSymbol{\cyrEn}{\mathalpha}{\M@cyrillicupper@font}{"41D}
  \DeclareMathSymbol{\cyrO}{\mathalpha}{\M@cyrillicupper@font}{"41E}
  \DeclareMathSymbol{\cyrPe}{\mathalpha}{\M@cyrillicupper@font}{"41F}
  \DeclareMathSymbol{\cyrEr}{\mathalpha}{\M@cyrillicupper@font}{"420}
  \DeclareMathSymbol{\cyrEs}{\mathalpha}{\M@cyrillicupper@font}{"421}
  \DeclareMathSymbol{\cyrTe}{\mathalpha}{\M@cyrillicupper@font}{"422}
  \DeclareMathSymbol{\cyrU}{\mathalpha}{\M@cyrillicupper@font}{"423}
  \DeclareMathSymbol{\cyrEf}{\mathalpha}{\M@cyrillicupper@font}{"424}
  \DeclareMathSymbol{\cyrHa}{\mathalpha}{\M@cyrillicupper@font}{"425}
  \DeclareMathSymbol{\cyrTse}{\mathalpha}{\M@cyrillicupper@font}{"426}
  \DeclareMathSymbol{\cyrChe}{\mathalpha}{\M@cyrillicupper@font}{"427}
  \DeclareMathSymbol{\cyrSha}{\mathalpha}{\M@cyrillicupper@font}{"428}
  \DeclareMathSymbol{\cyrShcha}{\mathalpha}{\M@cyrillicupper@font}{"429}
  \DeclareMathSymbol{\cyrHard}{\mathalpha}{\M@cyrillicupper@font}{"42A}
  \DeclareMathSymbol{\cyrYeru}{\mathalpha}{\M@cyrillicupper@font}{"42B}
  \DeclareMathSymbol{\cyrSoft}{\mathalpha}{\M@cyrillicupper@font}{"42C}
  \DeclareMathSymbol{\cyrE}{\mathalpha}{\M@cyrillicupper@font}{"42D}
  \DeclareMathSymbol{\cyrYu}{\mathalpha}{\M@cyrillicupper@font}{"42E}
  \DeclareMathSymbol{\cyrYa}{\mathalpha}{\M@cyrillicupper@font}{"42F}
  \DeclareMathSymbol{\cyrvarI}{\mathalpha}{\M@cyrillicupper@font}{"419}}
```

Set minuscule Cyrillic characters.

```
\def\M@cyrilliclower@set{%
  \edef\M@cyrilliclower@font{M\M@cyrilliclowershape\@tempa}
  \DeclareMathSymbol{\cyra}{\mathalpha}{\M@cyrilliclower@font}{"430}
  \DeclareMathSymbol{\cyrbe}{\mathalpha}{\M@cyrilliclower@font}{"431}
  \DeclareMathSymbol{\cyrve}{\mathalpha}{\M@cyrilliclower@font}{"432}
  \DeclareMathSymbol{\cyrghe}{\mathalpha}{\M@cyrilliclower@font}{"433}
  \DeclareMathSymbol{\cyrde}{\mathalpha}{\M@cyrilliclower@font}{"434}
  \DeclareMathSymbol{\cyrie}{\mathalpha}{\M@cyrilliclower@font}{"435}
  \DeclareMathSymbol{\cyrzhe}{\mathalpha}{\M@cyrilliclower@font}{"436}
  \DeclareMathSymbol{\cyrze}{\mathalpha}{\M@cyrilliclower@font}{"437}
  \DeclareMathSymbol{\cyri}{\mathalpha}{\M@cyrilliclower@font}{"438}
  \DeclareMathSymbol{\cyrka}{\mathalpha}{\M@cyrilliclower@font}{"43A}
  \DeclareMathSymbol{\cyrel}{\mathalpha}{\M@cyrilliclower@font}{"43B}
  \DeclareMathSymbol{\cyrem}{\mathalpha}{\M@cyrilliclower@font}{"43C}
  \DeclareMathSymbol{\cyren}{\mathalpha}{\M@cyrilliclower@font}{"43D}
  \DeclareMathSymbol{\cyro}{\mathalpha}{\M@cyrilliclower@font}{"43E}
  \DeclareMathSymbol{\cyrpe}{\mathalpha}{\M@cyrilliclower@font}{"43F}
  \DeclareMathSymbol{\cyrer}{\mathalpha}{\M@cyrilliclower@font}{"440}
  \DeclareMathSymbol{\cyres}{\mathalpha}{\M@cyrilliclower@font}{"441}
  \DeclareMathSymbol{\cyrte}{\mathalpha}{\M@cyrilliclower@font}{"442}
  \DeclareMathSymbol{\cyru}{\mathalpha}{\M@cyrilliclower@font}{"443}
  \DeclareMathSymbol{\cyref}{\mathalpha}{\M@cyrilliclower@font}{"444}
  \DeclareMathSymbol{\cyrha}{\mathalpha}{\M@cyrilliclower@font}{"445}
  \DeclareMathSymbol{\cyrtse}{\mathalpha}{\M@cyrilliclower@font}{"446}
  \DeclareMathSymbol{\cyrche}{\mathalpha}{\M@cyrilliclower@font}{"447}
  \DeclareMathSymbol{\cyrsha}{\mathalpha}{\M@cyrilliclower@font}{"448}
  \DeclareMathSymbol{\cyrshcha}{\mathalpha}{\M@cyrilliclower@font}{"449}
  \DeclareMathSymbol{\cyrhard}{\mathalpha}{\M@cyrilliclower@font}{"44A}
  \DeclareMathSymbol{\cyryeru}{\mathalpha}{\M@cyrilliclower@font}{"44B}
  \DeclareMathSymbol{\cyrsoft}{\mathalpha}{\M@cyrilliclower@font}{"44C}
  \DeclareMathSymbol{\cyre}{\mathalpha}{\M@cyrilliclower@font}{"44D}
  \DeclareMathSymbol{\cyryu}{\mathalpha}{\M@cyrilliclower@font}{"44E}
  \DeclareMathSymbol{\cyrya}{\mathalpha}{\M@cyrilliclower@font}{"44F}
  \DeclareMathSymbol{\cyrvari}{\mathalpha}{\M@cyrilliclower@font}{"439}}
```

Set Hebrew characters.

```
\def\M@hebrew@set{%
  \edef\M@hebrew@font{M\M@hebrewshape\@tempa}
  \DeclareMathSymbol{\aleph}{\mathalpha}{\M@hebrew@font}{"5D0}
  \DeclareMathSymbol{\beth}{\mathalpha}{\M@hebrew@font}{"5D1}
  \DeclareMathSymbol{\gimel}{\mathalpha}{\M@hebrew@font}{"5D2}
  \DeclareMathSymbol{\daleth}{\mathalpha}{\M@hebrew@font}{"5D3}
  \DeclareMathSymbol{\he}{\mathalpha}{\M@hebrew@font}{"5D4}
  \DeclareMathSymbol{\vav}{\mathalpha}{\M@hebrew@font}{"5D5}
  \DeclareMathSymbol{\zayin}{\mathalpha}{\M@hebrew@font}{"5D6}
  \DeclareMathSymbol{\het}{\mathalpha}{\M@hebrew@font}{"5D7}
  \DeclareMathSymbol{\tet}{\mathalpha}{\M@hebrew@font}{"5D8}
  \DeclareMathSymbol{\yod}{\mathalpha}{\M@hebrew@font}{"5D9}
  \DeclareMathSymbol{\kaf}{\mathalpha}{\M@hebrew@font}{"5DB}
  \DeclareMathSymbol{\lamed}{\mathalpha}{\M@hebrew@font}{"5DC}
  \DeclareMathSymbol{\mem}{\mathalpha}{\M@hebrew@font}{"5DE}
  \DeclareMathSymbol{\nun}{\mathalpha}{\M@hebrew@font}{"5E0}
  \DeclareMathSymbol{\samekh}{\mathalpha}{\M@hebrew@font}{"5E1}
  \DeclareMathSymbol{\ayin}{\mathalpha}{\M@hebrew@font}{"5E2}
  \DeclareMathSymbol{\pe}{\mathalpha}{\M@hebrew@font}{"5E4}
  \DeclareMathSymbol{\tsadi}{\mathalpha}{\M@hebrew@font}{"5E6}
  \DeclareMathSymbol{\qof}{\mathalpha}{\M@hebrew@font}{"5E7}
  \DeclareMathSymbol{\resh}{\mathalpha}{\M@hebrew@font}{"5E8}
  \DeclareMathSymbol{\shin}{\mathalpha}{\M@hebrew@font}{"5E9}
  \DeclareMathSymbol{\tav}{\mathalpha}{\M@hebrew@font}{"5EA}
  \DeclareMathSymbol{\varkaf}{\mathalpha}{\M@hebrew@font}{"5DA}
  \DeclareMathSymbol{\varmem}{\mathalpha}{\M@hebrew@font}{"5DD}
  \DeclareMathSymbol{\varnun}{\mathalpha}{\M@hebrew@font}{"5DF}
  \DeclareMathSymbol{\varpe}{\mathalpha}{\M@hebrew@font}{"5E3}
  \DeclareMathSymbol{\vartsadi}{\mathalpha}{\M@hebrew@font}{"5E5}}
```

Set digits.

```
\def\M@digits@set{%
  \edef\M@digits@font{M\M@digitsshape\@tempa}
  \DeclareMathSymbol{0}{\mathalpha}{\M@digits@font}{`0}
  \DeclareMathSymbol{1}{\mathalpha}{\M@digits@font}{`1}
  \DeclareMathSymbol{2}{\mathalpha}{\M@digits@font}{`2}
  \DeclareMathSymbol{3}{\mathalpha}{\M@digits@font}{`3}
  \DeclareMathSymbol{4}{\mathalpha}{\M@digits@font}{`4}
  \DeclareMathSymbol{5}{\mathalpha}{\M@digits@font}{`5}
  \DeclareMathSymbol{6}{\mathalpha}{\M@digits@font}{`6}
  \DeclareMathSymbol{7}{\mathalpha}{\M@digits@font}{`7}
  \DeclareMathSymbol{8}{\mathalpha}{\M@digits@font}{`8}
  \DeclareMathSymbol{9}{\mathalpha}{\M@digits@font}{`9}}
```

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

```
\def\M@operator@set{%
  \ifM@adjust@font
    \edef\M@operator@num{\number\csname symM\M@operatorshape\@tempa\endcsname}
    \protected\edef\M@operator@mathcodes{%
      \Umathcode`A=7+\M@operator@num+`A\relax
      \Umathcode`B=7+\M@operator@num+`B\relax
      \Umathcode`C=7+\M@operator@num+`C\relax
      \Umathcode`D=7+\M@operator@num+`D\relax
      \Umathcode`E=7+\M@operator@num+`E\relax
      \Umathcode`F=7+\M@operator@num+`F\relax
      \Umathcode`G=7+\M@operator@num+`G\relax
      \Umathcode`H=7+\M@operator@num+`H\relax
      \Umathcode`I=7+\M@operator@num+`I\relax
      \Umathcode`J=7+\M@operator@num+`J\relax
      \Umathcode`K=7+\M@operator@num+`K\relax
      \Umathcode`L=7+\M@operator@num+`L\relax
      \Umathcode`M=7+\M@operator@num+`M\relax
      \Umathcode`N=7+\M@operator@num+`N\relax
      \Umathcode`O=7+\M@operator@num+`O\relax
      \Umathcode`P=7+\M@operator@num+`P\relax
      \Umathcode`Q=7+\M@operator@num+`Q\relax
      \Umathcode`R=7+\M@operator@num+`R\relax
      \Umathcode`S=7+\M@operator@num+`S\relax
      \Umathcode`T=7+\M@operator@num+`T\relax
      \Umathcode`U=7+\M@operator@num+`U\relax
      \Umathcode`V=7+\M@operator@num+`V\relax
      \Umathcode`W=7+\M@operator@num+`W\relax
      \Umathcode`X=7+\M@operator@num+`X\relax
      \Umathcode`Y=7+\M@operator@num+`Y\relax
      \Umathcode`Z=7+\M@operator@num+`Z\relax
      \Umathcode`a=7+\M@operator@num+`a\relax
      \Umathcode`b=7+\M@operator@num+`b\relax
      \Umathcode`c=7+\M@operator@num+`c\relax
      \Umathcode`d=7+\M@operator@num+`d\relax
      \Umathcode`e=7+\M@operator@num+`e\relax
      \Umathcode`f=7+\M@operator@num+`f\relax
      \Umathcode`g=7+\M@operator@num+`g\relax
      \Umathcode`h=7+\M@operator@num+`h\relax
      \Umathcode`i=7+\M@operator@num+`i\relax
      \Umathcode`j=7+\M@operator@num+`j\relax
      \Umathcode`k=7+\M@operator@num+`k\relax
      \Umathcode`l=7+\M@operator@num+`l\relax
      \Umathcode`m=7+\M@operator@num+`m\relax
      \Umathcode`n=7+\M@operator@num+`n\relax
      \Umathcode`o=7+\M@operator@num+`o\relax
      \Umathcode`p=7+\M@operator@num+`p\relax
      \Umathcode`q=7+\M@operator@num+`q\relax
      \Umathcode`r=7+\M@operator@num+`r\relax
      \Umathcode`s=7+\M@operator@num+`s\relax
      \Umathcode`t=7+\M@operator@num+`t\relax
      \Umathcode`u=7+\M@operator@num+`u\relax
      \Umathcode`v=7+\M@operator@num+`v\relax
      \Umathcode`w=7+\M@operator@num+`w\relax
      \Umathcode`x=7+\M@operator@num+`x\relax
      \Umathcode`y=7+\M@operator@num+`y\relax
      \Umathcode`z=7+\M@operator@num+`z\relax
      \Umathchardef\imath=7+\M@operator@num+1044506\relax
      \Umathchardef\jmath=7+\M@operator@num+1044500\relax}
  \else
    \let\M@operator@mathcodes\@empty
  \fi
```

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

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

Set delimiters.

```
\ifM@adjust@font
  \def\M@delimiters@set{%
```

```
\edef\M@delimiters@font{M\M@delimitersshape\@tempa}
\DeclareMathSymbol{(}{\mathopen}{\M@delimiters@font}{"28}
\DeclareMathSymbol{)}{\mathclose}{\M@delimiters@font}{"29}
\DeclareMathSymbol{[}{\mathopen}{\M@delimiters@font}{"5B}
\DeclareMathSymbol{]}{\mathclose}{\M@delimiters@font}{"5D}
\ifM@symbols\else
  \DeclareMathSymbol{|}{\mathord}{\M@delimiters@font}{"7C}
\fi
\DeclareMathSymbol{\leftbrace}{\mathopen}{\M@delimiters@font}{"7B}
\DeclareMathSymbol{\rightbrace}{\mathclose}{\M@delimiters@font}{"7D}
\global\Udelcode40=+\csname sym\M@delimiters@font\endcsname+40\relax % (
\global\Udelcode41=+\csname sym\M@delimiters@font\endcsname+41\relax % )
\global\Udelcode47=+\csname sym\M@delimiters@font\endcsname+47\relax % /
\global\Udelcode91=+\csname sym\M@delimiters@font\endcsname+91\relax % [
\global\Udelcode93=+\csname sym\M@delimiters@font\endcsname+93\relax % ]
\global\Udelcode124=+\csname sym\M@delimiters@font\endcsname+124\relax % |
\global\let\vert=|
\protected\gdef\backslash{\ifmmode\mathbackslash\else\textbackslash\fi}
\protected\xdef\mathbackslash{%
  \Udelimiter+2+\number\csname sym\M@delimiters@font\endcsname
    +92\relax} % backslash
\protected\xdef\lbrace{%
  \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname
    +123\relax} % {
\protected\xdef\rbrace{%
  \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
    +125\relax} % }
\protected\xdef\lguil{%
  \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname
    +8249\relax} % single left guilement
\protected\xdef\rguil{%
  \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
    +8250\relax} % single right guilement
\protected\xdef\llguil{%
  \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname
    +171\relax} % double left guilement
\protected\xdef\rrguil{%
  \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
    +187\relax} % double right guilement
\protected\xdef\fakelangle{%
  \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname
    +1044508\relax} % fake left angle
\protected\xdef\fakerangle{%
  \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
    +1044509\relax} % fake right angle
\protected\xdef\fakellangle{%
  \Udelimiter+4+\number\csname sym\M@delimiters@font\endcsname
```

```
        +1044510\relax} % fake double left angle
    \protected\xdef\fakerrangle{%
      \Udelimiter+5+\number\csname sym\M@delimiters@font\endcsname
        +1044511\relax} % fake double right angle
    }
\else
  \def\M@delimiters@set{%
    \edef\M@delimiters@font{M\M@delimitersshape\@tempa}
    \DeclareMathSymbol{(}{\mathopen}{\M@delimiters@font}{"28}
    \DeclareMathSymbol{)}{\mathclose}{\M@delimiters@font}{"29}
    \DeclareMathSymbol{[}{\mathopen}{\M@delimiters@font}{"5B}
    \DeclareMathSymbol{]}{\mathclose}{\M@delimiters@font}{"5D}
    \DeclareMathSymbol{\lguil}{\mathopen}{\M@delimiters@font}{"2039}
    \DeclareMathSymbol{\rguil}{\mathclose}{\M@delimiters@font}{"203A}
    \DeclareMathSymbol{\llguil}{\mathopen}{\M@delimiters@font}{"AB}
    \DeclareMathSymbol{\rrguil}{\mathclose}{\M@delimiters@font}{"BB}
    \DeclareMathSymbol{\leftbrace}{\mathopen}{\M@delimiters@font}{"7B}
    \DeclareMathSymbol{\rightbrace}{\mathclose}{\M@delimiters@font}{"7D}}
\fi
```

Radicals.

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

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

```
    \gdef\r@@t##1##2{%
      \setbox\z@\hbox{$\m@th##1\sqrtsign{##2}$}%
      \setbox\surdbox\hbox{$\m@th##1\@sqrts@gn{%
        \hbox{\vphantom{$\m@th##1##2$}}}$}
      \dimen@\ht\surdbox
      \advance\dimen@\dp\surdbox
      \dimen@=0.6\dimen@
      \advance\dimen@-\dp\surdbox
      \ifdim\wd\rootbox<0.6\wd\surdbox
        \kern0.6\wd\surdbox
      \else
        \kern\wd\rootbox
      \fi
      \raise\dimen@\hbox{\llap{\copy\rootbox}}
      \kern-0.6\wd\surdbox
      \box\z@}
    \gdef\sqrtsign##1{\@sqrts@gn{\mkern\radicandoffset##1}}}
```

```
\else
  \def\M@radical@set{%
    \edef\M@radical@font{M\M@radicalshape\@tempa}
    \DeclareMathSymbol{\surd}{\mathord}{\M@radical@font}{"221A}}
\fi
```

Big operators.

```
\def\M@bigops@set{%
  \edef\M@bigops@font{M\M@bigopsshape\@tempa}
  \let\sum\@undefined
  \let\prod\@undefined
  \DeclareMathSymbol{\sum}{\mathop}{\M@bigops@font}{"2211}
  \DeclareMathSymbol{\prod}{\mathop}{\M@bigops@font}{"220F}
  \DeclareMathSymbol{\intop}{\mathop}{\M@bigops@font}{"222B}}
```

Extended big operators.

```
\def\M@extbigops@set{%
  \edef\M@extbigops@font{M\M@extbigopsshape\@tempa}
  \let\coprod\@undefined
  \let\bigvee\@undefined
  \let\bigwedge\@undefined
  \let\bigcup\@undefined
  \let\bigcap\@undefined
  \let\bigoplus\@undefined
  \let\bigotimes\@undefined
  \let\bigodot\@undefined
  \let\bigsqcup\@undefined
  \DeclareMathSymbol{\coprod}{\mathop}{\M@extbigops@font}{"2210}
  \DeclareMathSymbol{\bigvee}{\mathop}{\M@extbigops@font}{"22C1}
  \DeclareMathSymbol{\bigwedge}{\mathop}{\M@extbigops@font}{"22C0}
  \DeclareMathSymbol{\bigcup}{\mathop}{\M@extbigops@font}{"22C3}
  \DeclareMathSymbol{\bigcap}{\mathop}{\M@extbigops@font}{"22C2}
  \DeclareMathSymbol{\iintop}{\mathop}{\M@extbigops@font}{"222C}
    \protected\gdef\iint{\iintop\nolimits}
  \DeclareMathSymbol{\iiintop}{\mathop}{\M@extbigops@font}{"222D}
    \protected\gdef\iiint{\iiintop\nolimits}
  \DeclareMathSymbol{\ointop}{\mathop}{\M@extbigops@font}{"222E}
    \protected\gdef\oint{\ointop\nolimits}
  \DeclareMathSymbol{\oiintop}{\mathop}{\M@extbigops@font}{"222F}
    \protected\gdef\oiint{\oiintop\nolimits}
  \DeclareMathSymbol{\oiiintop}{\mathop}{\M@extbigops@font}{"2230}
    \protected\gdef\oiiint{\oiiintop\nolimits}
  \DeclareMathSymbol{\bigoplus}{\mathop}{\M@extbigops@font}{"2A01}
  \DeclareMathSymbol{\bigotimes}{\mathop}{\M@extbigops@font}{"2A02}
  \DeclareMathSymbol{\bigodot}{\mathop}{\M@extbigops@font}{"2A00}
  \DeclareMathSymbol{\bigsqcap}{\mathop}{\M@extbigops@font}{"2A05}
  \DeclareMathSymbol{\bigsqcup}{\mathop}{\M@extbigops@font}{"2A06}}
```

Set symbols.

```
\def\M@symbols@set{%
  \edef\M@symbols@font{M\M@symbolsshape\@tempa}
  \let\colon\@undefined
  \let\mathellipsis\@undefined
  \DeclareMathSymbol{.}{\mathord}{\M@symbols@font}{"2E}
  \DeclareMathSymbol{@}{\mathord}{\M@symbols@font}{"40}
  \DeclareMathSymbol{'}{\mathord}{\M@symbols@font}{"2032}
  \DeclareMathSymbol{\prime}{\mathord}{\M@symbols@font}{"2032}
  \DeclareMathSymbol{"}{\mathord}{\M@symbols@font}{"2033}
  \DeclareMathSymbol{\mathhash}{\mathord}{\M@symbols@font}{"23}
  \DeclareMathSymbol{\mathdollar}{\mathord}{\M@symbols@font}{"24}
  \DeclareMathSymbol{\mathpercent}{\mathord}{\M@symbols@font}{"25}
  \DeclareMathSymbol{\mathand}{\mathord}{\M@symbols@font}{"26}
  \DeclareMathSymbol{\mathparagraph}{\mathord}{\M@symbols@font}{"B6}
  \DeclareMathSymbol{\mathsection}{\mathord}{\M@symbols@font}{"A7}
  \DeclareMathSymbol{\mathsterling}{\mathord}{\M@symbols@font}{"A3}
  \DeclareMathSymbol{\neg}{\mathord}{\M@symbols@font}{"AC}
  \DeclareMathSymbol{|}{\mathord}{\M@symbols@font}{"7C}
  \DeclareMathSymbol{\infty}{\mathord}{\M@symbols@font}{"221E}
  \DeclareMathSymbol{\partial}{\mathord}{\M@symbols@font}{"2202}
  \DeclareMathSymbol{\degree}{\mathord}{\M@symbols@font}{"B0}
  \DeclareMathSymbol{\increment}{\mathord}{\M@symbols@font}{"2206}
  \DeclareMathSymbol{\comma}{\mathord}{\M@symbols@font}{"2C}
  \DeclareMathSymbol{+}{\mathbin}{\M@symbols@font}{"2B}
  \DeclareMathSymbol{-}{\mathbin}{\M@symbols@font}{"2212}
  \DeclareMathSymbol{*}{\mathbin}{\M@symbols@font}{"2A}
  \DeclareMathSymbol{\times}{\mathbin}{\M@symbols@font}{"D7}
  \DeclareMathSymbol{/}{\mathord}{\M@symbols@font}{"2F}
  \DeclareMathSymbol{\fractionslash}{\mathord}{\M@symbols@font}{"2215}
  \DeclareMathSymbol{\div}{\mathbin}{\M@symbols@font}{"F7}
  \DeclareMathSymbol{\pm}{\mathbin}{\M@symbols@font}{"B1}
  \DeclareMathSymbol{\bullet}{\mathbin}{\M@symbols@font}{"2022}
  \DeclareMathSymbol{\dagger}{\mathbin}{\M@symbols@font}{"2020}
  \DeclareMathSymbol{\ddagger}{\mathbin}{\M@symbols@font}{"2021}
  \DeclareMathSymbol{\cdot}{\mathbin}{\M@symbols@font}{"2219}
  \DeclareMathSymbol{\setminus}{\mathbin}{\M@symbols@font}{"5C}
  \DeclareMathSymbol{=}{\mathrel}{\M@symbols@font}{"3D}
  \DeclareMathSymbol{<}{\mathrel}{\M@symbols@font}{"3C}
  \DeclareMathSymbol{>}{\mathrel}{\M@symbols@font}{"3E}
  \DeclareMathSymbol{\leq}{\mathrel}{\M@symbols@font}{"2264}
  \DeclareMathSymbol{\geq}{\mathrel}{\M@symbols@font}{"2265}
  \DeclareMathSymbol{\sim}{\mathrel}{\M@symbols@font}{"7E}
  \DeclareMathSymbol{\approx}{\mathrel}{\M@symbols@font}{"2248}
  \DeclareMathSymbol{\equiv}{\mathrel}{\M@symbols@font}{"2261}
  \DeclareMathSymbol{\mid}{\mathrel}{\M@symbols@font}{"7C}
  \DeclareMathSymbol{\parallel}{\mathrel}{\M@symbols@font}{"2016}
  \DeclareMathSymbol{:}{\mathrel}{\M@symbols@font}{"3A}
  \DeclareMathSymbol{?}{\mathclose}{\M@symbols@font}{"3F}
  \DeclareMathSymbol{!}{\mathclose}{\M@symbols@font}{"21}
  \DeclareMathSymbol{,}{\mathpunct}{\M@symbols@font}{"2C}
  \DeclareMathSymbol{;}{\mathpunct}{\M@symbols@font}{"3B}
  \DeclareMathSymbol{\colon}{\mathord}{\M@symbols@font}{"3A}
  \DeclareMathSymbol{\mathellipsis}{\mathinner}{\M@symbols@font}{"2026}
```

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

```
  \protected\gdef\#{\ifmmode\mathhash\else\char"23\relax\fi}
  \protected\gdef\%{\ifmmode\mathpercent\else\char"25\relax\fi}
  \protected\gdef\&{\ifmmode\mathand\else\char"26\relax\fi}
  \DeclareMathSymbol{\@relbar}{\mathbin}{symbols}{"00}
  \DeclareMathSymbol{\@Relbar}{\mathrel}{operators}{"3D}
  \DeclareMathSymbol{\@verticalbar}{\mathord}{symbols}{"6A}
  \ifM@extsymbols\else
    \protected\gdef\simeq{\mathrel{\mathpalette\stack@flatrel{{-}{\sim}}}}
    \protected\gdef\cong{\mathrel{\mathpalette\stack@flatrel{{=}{\sim}}}}
  \fi
  \protected\gdef\relbar{\mathrel{\smash\@relbar}}
  \protected\gdef\Relbar{\mathrel{\@Relbar}}
  \protected\gdef\models{\mathrel{\@verticalbar}\joinrel\Relbar}
```

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

```
  \ifM@adjust@font
    \DeclareMathSymbol{\bigat}{\mathop}{\M@symbols@font}{"40}
    \DeclareMathSymbol{\bighash}{\mathop}{\M@symbols@font}{"23}
    \DeclareMathSymbol{\bigdollar}{\mathop}{\M@symbols@font}{"24}
    \DeclareMathSymbol{\bigpercent}{\mathop}{\M@symbols@font}{"25}
    \DeclareMathSymbol{\bigand}{\mathop}{\M@symbols@font}{"26}
    \DeclareMathSymbol{\bigplus}{\mathop}{\M@symbols@font}{"2B}
    \DeclareMathSymbol{\bigp}{\mathop}{\M@symbols@font}{"21}
    \DeclareMathSymbol{\bigq}{\mathop}{\M@symbols@font}{"3F}
    \DeclareMathSymbol{\bigS}{\mathop}{\M@symbols@font}{"A7}
    \DeclareMathSymbol{\bigtimes}{\mathop}{\M@symbols@font}{"D7}
    \DeclareMathSymbol{\bigdiv}{\mathop}{\M@symbols@font}{"F7}
```

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

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

Set extended symbols.

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

```
\let\simeq\@undefined
\let\sqsubset\@undefined
\let\sqsupset\@undefined
\let\bowtie\@undefined
\let\doteq\@undefined
\let\neq\@undefined
\DeclareMathSymbol{\wp}{\mathord}{\M@extsymbols@font}{"2118}
\DeclareMathSymbol{\Re}{\mathord}{\M@extsymbols@font}{"211C}
\DeclareMathSymbol{\Im}{\mathord}{\M@extsymbols@font}{"2111}
\DeclareMathSymbol{\ell}{\mathord}{\M@extsymbols@font}{"2113}
\DeclareMathSymbol{\forall}{\mathord}{\M@extsymbols@font}{"2200}
\DeclareMathSymbol{\exists}{\mathord}{\M@extsymbols@font}{"2203}
\DeclareMathSymbol{\emptyset}{\mathord}{\M@extsymbols@font}{"2205}
\DeclareMathSymbol{\in}{\mathord}{\M@extsymbols@font}{"2208}
\DeclareMathSymbol{\ni}{\mathord}{\M@extsymbols@font}{"220B}
\DeclareMathSymbol{\mp}{\mathord}{\M@extsymbols@font}{"2213}
\DeclareMathSymbol{\angle}{\mathord}{\M@extsymbols@font}{"2220}
\DeclareMathSymbol{\top}{\mathord}{\M@extsymbols@font}{"22A4}
\DeclareMathSymbol{\bot}{\mathord}{\M@extsymbols@font}{"22A5}
\DeclareMathSymbol{\vdash}{\mathord}{\M@extsymbols@font}{"22A2}
\DeclareMathSymbol{\dashv}{\mathord}{\M@extsymbols@font}{"22A3}
\DeclareMathSymbol{\flat}{\mathord}{\M@extsymbols@font}{"266D}
\DeclareMathSymbol{\natural}{\mathord}{\M@extsymbols@font}{"266E}
\DeclareMathSymbol{\sharp}{\mathord}{\M@extsymbols@font}{"266F}
\DeclareMathSymbol{\fflat}{\mathord}{\M@extsymbols@font}{"1D12B}
\DeclareMathSymbol{\ssharp}{\mathord}{\M@extsymbols@font}{"1D12A}
\DeclareMathSymbol{\bclubsuit}{\mathord}{\M@extsymbols@font}{"2663}
\DeclareMathSymbol{\bdiamondsuit}{\mathord}{\M@extsymbols@font}{"2666}
\DeclareMathSymbol{\bheartsuit}{\mathord}{\M@extsymbols@font}{"2665}
\DeclareMathSymbol{\bspadesuit}{\mathord}{\M@extsymbols@font}{"2660}
\DeclareMathSymbol{\wclubsuit}{\mathord}{\M@extsymbols@font}{"2667}
\DeclareMathSymbol{\wdiamondsuit}{\mathord}{\M@extsymbols@font}{"2662}
\DeclareMathSymbol{\wheartsuit}{\mathord}{\M@extsymbols@font}{"2661}
\DeclareMathSymbol{\wspadesuit}{\mathord}{\M@extsymbols@font}{"2664}
  \global\let\spadesuit\bspadesuit
  \global\let\heartsuit\wheartsuit
  \global\let\diamondsuit\wdiamondsuit
  \global\let\clubsuit\bclubsuit
\DeclareMathSymbol{\wedge}{\mathbin}{\M@extsymbols@font}{"2227}
\DeclareMathSymbol{\vee}{\mathbin}{\M@extsymbols@font}{"2228}
\DeclareMathSymbol{\cap}{\mathord}{\M@extsymbols@font}{"2229}
\DeclareMathSymbol{\cup}{\mathbin}{\M@extsymbols@font}{"222A}
\DeclareMathSymbol{\sqcap}{\mathbin}{\M@extsymbols@font}{"2293}
\DeclareMathSymbol{\sqcup}{\mathbin}{\M@extsymbols@font}{"2294}
\DeclareMathSymbol{\amalg}{\mathbin}{\M@extsymbols@font}{"2A3F}
\DeclareMathSymbol{\wr}{\mathbin}{\M@extsymbols@font}{"2240}
\DeclareMathSymbol{\ast}{\mathbin}{\M@extsymbols@font}{"2217}
```

```
\DeclareMathSymbol{\star}{\mathbin}{\M@extsymbols@font}{"22C6}
\DeclareMathSymbol{\diamond}{\mathbin}{\M@extsymbols@font}{"22C4}
\DeclareMathSymbol{\varcdot}{\mathbin}{\M@extsymbols@font}{"22C5}
\DeclareMathSymbol{\varsetminus}{\mathbin}{\M@extsymbols@font}{"2216}
\DeclareMathSymbol{\oplus}{\mathbin}{\M@extsymbols@font}{"2295}
\DeclareMathSymbol{\otimes}{\mathbin}{\M@extsymbols@font}{"2297}
\DeclareMathSymbol{\ominus}{\mathbin}{\M@extsymbols@font}{"2296}
\DeclareMathSymbol{\odiv}{\mathbin}{\M@extsymbols@font}{"2A38}
\DeclareMathSymbol{\oslash}{\mathbin}{\M@extsymbols@font}{"2298}
\DeclareMathSymbol{\odot}{\mathbin}{\M@extsymbols@font}{"2299}
\DeclareMathSymbol{\sqplus}{\mathbin}{\M@extsymbols@font}{"229E}
\DeclareMathSymbol{\sqtimes}{\mathbin}{\M@extsymbols@font}{"22A0}
\DeclareMathSymbol{\sqminus}{\mathbin}{\M@extsymbols@font}{"229F}
\DeclareMathSymbol{\sqdot}{\mathbin}{\M@extsymbols@font}{"22A1}
\DeclareMathSymbol{\in}{\mathrel}{\M@extsymbols@font}{"2208}
\DeclareMathSymbol{\ni}{\mathrel}{\M@extsymbols@font}{"220B}
\DeclareMathSymbol{\subset}{\mathrel}{\M@extsymbols@font}{"2282}
\DeclareMathSymbol{\supset}{\mathrel}{\M@extsymbols@font}{"2283}
\DeclareMathSymbol{\subseteq}{\mathrel}{\M@extsymbols@font}{"2286}
\DeclareMathSymbol{\supseteq}{\mathrel}{\M@extsymbols@font}{"2287}
\DeclareMathSymbol{\sqsubset}{\mathrel}{\M@extsymbols@font}{"228F}
\DeclareMathSymbol{\sqsupset}{\mathrel}{\M@extsymbols@font}{"2290}
\DeclareMathSymbol{\sqsubseteq}{\mathrel}{\M@extsymbols@font}{"2291}
\DeclareMathSymbol{\sqsupseteq}{\mathrel}{\M@extsymbols@font}{"2292}
\DeclareMathSymbol{\triangleleft}{\mathrel}{\M@extsymbols@font}{"22B2}
\DeclareMathSymbol{\triangleright}{\mathrel}{\M@extsymbols@font}{"22B3}
\DeclareMathSymbol{\trianglelefteq}{\mathrel}{\M@extsymbols@font}{"22B4}
\DeclareMathSymbol{\trianglerighteq}{\mathrel}{\M@extsymbols@font}{"22B5}
\DeclareMathSymbol{\propto}{\mathrel}{\M@extsymbols@font}{"221D}
\DeclareMathSymbol{\bowtie}{\mathrel}{\M@extsymbols@font}{"22C8}
\DeclareMathSymbol{\hourglass}{\mathrel}{\M@extsymbols@font}{"29D6}
\DeclareMathSymbol{\therefore}{\mathrel}{\M@extsymbols@font}{"2234}
\DeclareMathSymbol{\because}{\mathrel}{\M@extsymbols@font}{"2235}
\DeclareMathSymbol{\ratio}{\mathrel}{\M@extsymbols@font}{"2236}
\DeclareMathSymbol{\proportion}{\mathrel}{\M@extsymbols@font}{"2237}
\DeclareMathSymbol{\ll}{\mathrel}{\M@extsymbols@font}{"226A}
\DeclareMathSymbol{\gg}{\mathrel}{\M@extsymbols@font}{"226B}
\DeclareMathSymbol{\lll}{\mathrel}{\M@extsymbols@font}{"22D8}
\DeclareMathSymbol{\ggg}{\mathrel}{\M@extsymbols@font}{"22D9}
\DeclareMathSymbol{\leqq}{\mathrel}{\M@extsymbols@font}{"2266}
\DeclareMathSymbol{\geqq}{\mathrel}{\M@extsymbols@font}{"2267}
\DeclareMathSymbol{\lapprox}{\mathrel}{\M@extsymbols@font}{"2A85}
\DeclareMathSymbol{\gapprox}{\mathrel}{\M@extsymbols@font}{"2A86}
\DeclareMathSymbol{\simeq}{\mathrel}{\M@extsymbols@font}{"2243}
\DeclareMathSymbol{\eqsim}{\mathrel}{\M@extsymbols@font}{"2242}
\DeclareMathSymbol{\simeqq}{\mathrel}{\M@extsymbols@font}{"2245}
  \global\let\cong\simeqq
```

```
\DeclareMathSymbol{\approxeq}{\mathrel}{\M@extsymbols@font}{"224A}
\DeclareMathSymbol{\sssim}{\mathrel}{\M@extsymbols@font}{"224B}
\DeclareMathSymbol{\seq}{\mathrel}{\M@extsymbols@font}{"224C}
\DeclareMathSymbol{\doteq}{\mathrel}{\M@extsymbols@font}{"2250}
\DeclareMathSymbol{\coloneq}{\mathrel}{\M@extsymbols@font}{"2254}
\DeclareMathSymbol{\eqcolon}{\mathrel}{\M@extsymbols@font}{"2255}
\DeclareMathSymbol{\ringeq}{\mathrel}{\M@extsymbols@font}{"2257}
\DeclareMathSymbol{\arceq}{\mathrel}{\M@extsymbols@font}{"2258}
\DeclareMathSymbol{\wedgeeq}{\mathrel}{\M@extsymbols@font}{"2259}
\DeclareMathSymbol{\veeeq}{\mathrel}{\M@extsymbols@font}{"225A}
\DeclareMathSymbol{\stareq}{\mathrel}{\M@extsymbols@font}{"225B}
\DeclareMathSymbol{\triangleeq}{\mathrel}{\M@extsymbols@font}{"225C}
\DeclareMathSymbol{\defeq}{\mathrel}{\M@extsymbols@font}{"225D}
\DeclareMathSymbol{\qeq}{\mathrel}{\M@extsymbols@font}{"225F}
\DeclareMathSymbol{\lsim}{\mathrel}{\M@extsymbols@font}{"2272}
\DeclareMathSymbol{\gsim}{\mathrel}{\M@extsymbols@font}{"2273}
\DeclareMathSymbol{\prec}{\mathrel}{\M@extsymbols@font}{"227A}
\DeclareMathSymbol{\succ}{\mathrel}{\M@extsymbols@font}{"227B}
\DeclareMathSymbol{\preceq}{\mathrel}{\M@extsymbols@font}{"227C}
\DeclareMathSymbol{\succeq}{\mathrel}{\M@extsymbols@font}{"227D}
\DeclareMathSymbol{\preceqq}{\mathrel}{\M@extsymbols@font}{"2AB3}
\DeclareMathSymbol{\succeqq}{\mathrel}{\M@extsymbols@font}{"2AB4}
\DeclareMathSymbol{\precsim}{\mathrel}{\M@extsymbols@font}{"227E}
\DeclareMathSymbol{\succsim}{\mathrel}{\M@extsymbols@font}{"227F}
\DeclareMathSymbol{\precapprox}{\mathrel}{\M@extsymbols@font}{"2AB7}
\DeclareMathSymbol{\succapprox}{\mathrel}{\M@extsymbols@font}{"2AB8}
\DeclareMathSymbol{\precprec}{\mathrel}{\M@extsymbols@font}{"2ABB}
\DeclareMathSymbol{\succsucc}{\mathrel}{\M@extsymbols@font}{"2ABC}
\DeclareMathSymbol{\asymp}{\mathrel}{\M@extsymbols@font}{"224D}
\DeclareMathSymbol{\nin}{\mathrel}{\M@extsymbols@font}{"2209}
\DeclareMathSymbol{\nni}{\mathrel}{\M@extsymbols@font}{"220C}
\DeclareMathSymbol{\nsubset}{\mathrel}{\M@extsymbols@font}{"2284}
\DeclareMathSymbol{\nsupset}{\mathrel}{\M@extsymbols@font}{"2285}
\DeclareMathSymbol{\nsubseteq}{\mathrel}{\M@extsymbols@font}{"2288}
\DeclareMathSymbol{\nsupseteq}{\mathrel}{\M@extsymbols@font}{"2289}
\DeclareMathSymbol{\subsetneq}{\mathrel}{\M@extsymbols@font}{"228A}
\DeclareMathSymbol{\supsetneq}{\mathrel}{\M@extsymbols@font}{"228B}
\DeclareMathSymbol{\nsqsubseteq}{\mathrel}{\M@extsymbols@font}{"22E2}
\DeclareMathSymbol{\nsqsupseteq}{\mathrel}{\M@extsymbols@font}{"22E3}
\DeclareMathSymbol{\sqsubsetneq}{\mathrel}{\M@extsymbols@font}{"22E4}
\DeclareMathSymbol{\sqsupsetneq}{\mathrel}{\M@extsymbols@font}{"22E5}
\DeclareMathSymbol{\neq}{\mathrel}{\M@extsymbols@font}{"2260}
\DeclareMathSymbol{\nl}{\mathrel}{\M@extsymbols@font}{"226E}
\DeclareMathSymbol{\nleq}{\mathrel}{\M@extsymbols@font}{"2270}
\DeclareMathSymbol{\ngeq}{\mathrel}{\M@extsymbols@font}{"2271}
\DeclareMathSymbol{\lneq}{\mathrel}{\M@extsymbols@font}{"2A87}
\DeclareMathSymbol{\gneq}{\mathrel}{\M@extsymbols@font}{"2A88}
```

```
  \DeclareMathSymbol{\lneqq}{\mathrel}{\M@extsymbols@font}{"2268}
  \DeclareMathSymbol{\gneqq}{\mathrel}{\M@extsymbols@font}{"2269}
  \DeclareMathSymbol{\ntriangleleft}{\mathrel}{\M@extsymbols@font}{"22EA}
  \DeclareMathSymbol{\ntriangleright}{\mathrel}{\M@extsymbols@font}{"22EB}
  \DeclareMathSymbol{\ntrianglelefteq}{\mathrel}{\M@extsymbols@font}{"22EC}
  \DeclareMathSymbol{\ntrianglerighteq}{\mathrel}{\M@extsymbols@font}{"22ED}
  \DeclareMathSymbol{\nsim}{\mathrel}{\M@extsymbols@font}{"2241}
  \DeclareMathSymbol{\napprox}{\mathrel}{\M@extsymbols@font}{"2249}
  \DeclareMathSymbol{\nsimeq}{\mathrel}{\M@extsymbols@font}{"2244}
  \DeclareMathSymbol{\nsimeqq}{\mathrel}{\M@extsymbols@font}{"2247}
  \DeclareMathSymbol{\simneqq}{\mathrel}{\M@extsymbols@font}{"2246}
  \DeclareMathSymbol{\nlsim}{\mathrel}{\M@extsymbols@font}{"2274}
  \DeclareMathSymbol{\ngsim}{\mathrel}{\M@extsymbols@font}{"2275}
  \DeclareMathSymbol{\lnsim}{\mathrel}{\M@extsymbols@font}{"22E6}
  \DeclareMathSymbol{\gnsim}{\mathrel}{\M@extsymbols@font}{"22E7}
  \DeclareMathSymbol{\lnapprox}{\mathrel}{\M@extsymbols@font}{"2A89}
  \DeclareMathSymbol{\gnapprox}{\mathrel}{\M@extsymbols@font}{"2A8A}
  \DeclareMathSymbol{\nprec}{\mathrel}{\M@extsymbols@font}{"2280}
  \DeclareMathSymbol{\nsucc}{\mathrel}{\M@extsymbols@font}{"2281}
  \DeclareMathSymbol{\npreceq}{\mathrel}{\M@extsymbols@font}{"22E0}
  \DeclareMathSymbol{\nsucceq}{\mathrel}{\M@extsymbols@font}{"22E1}
  \DeclareMathSymbol{\precneq}{\mathrel}{\M@extsymbols@font}{"2AB1}
  \DeclareMathSymbol{\succneq}{\mathrel}{\M@extsymbols@font}{"2AB2}
  \DeclareMathSymbol{\precneqq}{\mathrel}{\M@extsymbols@font}{"2AB5}
  \DeclareMathSymbol{\succneqq}{\mathrel}{\M@extsymbols@font}{"2AB6}
  \DeclareMathSymbol{\precnsim}{\mathrel}{\M@extsymbols@font}{"22E8}
  \DeclareMathSymbol{\succnsim}{\mathrel}{\M@extsymbols@font}{"22E9}
  \DeclareMathSymbol{\precnapprox}{\mathrel}{\M@extsymbols@font}{"2AB9}
  \DeclareMathSymbol{\succnapprox}{\mathrel}{\M@extsymbols@font}{"2ABA}
  \DeclareMathSymbol{\nequiv}{\mathrel}{\M@extsymbols@font}{"2262}
```

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

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

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

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

Set arrows.

```
\def\M@arrows@set{%
  \edef\M@arrows@font{M\M@arrowsshape\@tempa}
  \let\uparrow\@undefined
  \let\Uparrow\@undefined
  \let\downarrow\@undefined
```

```
\let\Downarrow\@undefined
\let\updownarrow\@undefined
\let\Updownarrow\@undefined
\let\longrightarrow\@undefined
\let\longleftarrow\@undefined
\let\longleftrightarrow\@undefined
\let\hookrightarrow\@undefined
\let\hookleftarrow\@undefined
\let\Longrightarrow\@undefined
\let\Longleftarrow\@undefined
\let\Longleftrightarrow\@undefined
\let\rightleftharpoons\@undefined
\DeclareMathSymbol{\rightarrow}{\mathrel}{\M@arrows@font}{"2192}
  \global\let\to\rightarrow
\DeclareMathSymbol{\nrightarrow}{\mathrel}{\M@arrows@font}{"219B}
\DeclareMathSymbol{\Rightarrow}{\mathrel}{\M@arrows@font}{"21D2}
\DeclareMathSymbol{\nRightarrow}{\mathrel}{\M@arrows@font}{"21CF}
\DeclareMathSymbol{\Rrightarrow}{\mathrel}{\M@arrows@font}{"21DB}
\DeclareMathSymbol{\longrightarrow}{\mathrel}{\M@arrows@font}{"27F6}
\DeclareMathSymbol{\Longrightarrow}{\mathrel}{\M@arrows@font}{"27F9}
\DeclareMathSymbol{\rightbararrow}{\mathrel}{\M@arrows@font}{"21A6}
  \global\let\mapsto\rightbararrow
\DeclareMathSymbol{\Rightbararrow}{\mathrel}{\M@arrows@font}{"2907}
\DeclareMathSymbol{\longrightbararrow}{\mathrel}{\M@arrows@font}{"27FC}
  \global\let\longmapsto\longrightbararrow
\DeclareMathSymbol{\Longrightbararrow}{\mathrel}{\M@arrows@font}{"27FE}
\DeclareMathSymbol{\hookrightarrow}{\mathrel}{\M@arrows@font}{"21AA}
\DeclareMathSymbol{\rightdasharrow}{\mathrel}{\M@arrows@font}{"21E2}
\DeclareMathSymbol{\rightharpoonup}{\mathrel}{\M@arrows@font}{"21C0}
\DeclareMathSymbol{\rightharpoondown}{\mathrel}{\M@arrows@font}{"21C1}
\DeclareMathSymbol{\rightarrowtail}{\mathrel}{\M@arrows@font}{"21A3}
\DeclareMathSymbol{\rightoplusarrow}{\mathrel}{\M@arrows@font}{"27F4}
\DeclareMathSymbol{\rightwavearrow}{\mathrel}{\M@arrows@font}{"219D}
\DeclareMathSymbol{\rightsquigarrow}{\mathrel}{\M@arrows@font}{"21DD}
\DeclareMathSymbol{\longrightsquigarrow}{\mathrel}{\M@arrows@font}{"27FF}
\DeclareMathSymbol{\looparrowright}{\mathrel}{\M@arrows@font}{"21AC}
\DeclareMathSymbol{\curvearrowright}{\mathrel}{\M@arrows@font}{"293B}
\DeclareMathSymbol{\circlearrowright}{\mathrel}{\M@arrows@font}{"21BB}
\DeclareMathSymbol{\twoheadrightarrow}{\mathrel}{\M@arrows@font}{"21A0}
\DeclareMathSymbol{\rightarrowtobar}{\mathrel}{\M@arrows@font}{"21E5}
\DeclareMathSymbol{\rightwhitearrow}{\mathrel}{\M@arrows@font}{"21E8}
\DeclareMathSymbol{\rightrightarrows}{\mathrel}{\M@arrows@font}{"21C9}
\DeclareMathSymbol{\rightrightrightarrows}{\mathrel}{\M@arrows@font}{"21F6}
\DeclareMathSymbol{\leftarrow}{\mathrel}{\M@arrows@font}{"2190}
  \global\let\from\leftarrow
\DeclareMathSymbol{\nleftarrow}{\mathrel}{\M@arrows@font}{"219A}
\DeclareMathSymbol{\Leftarrow}{\mathrel}{\M@arrows@font}{"21D0}
```

```
\DeclareMathSymbol{\nLeftarrow}{\mathrel}{\M@arrows@font}{"21CD}
\DeclareMathSymbol{\Lleftarrow}{\mathrel}{\M@arrows@font}{"21DA}
\DeclareMathSymbol{\longleftarrow}{\mathrel}{\M@arrows@font}{"27F5}
\DeclareMathSymbol{\Longleftarrow}{\mathrel}{\M@arrows@font}{"27F8}
\DeclareMathSymbol{\leftbararrow}{\mathrel}{\M@arrows@font}{"21A4}
  \global\let\mapsfrom\leftbararrow
\DeclareMathSymbol{\Leftbararrow}{\mathrel}{\M@arrows@font}{"2906}
\DeclareMathSymbol{\longleftbararrow}{\mathrel}{\M@arrows@font}{"27FB}
  \global\let\longmapsfrom\longleftbararrow
\DeclareMathSymbol{\Longleftbararrow}{\mathrel}{\M@arrows@font}{"27FD}
\DeclareMathSymbol{\hookleftarrow}{\mathrel}{\M@arrows@font}{"21A9}
\DeclareMathSymbol{\leftdasharrow}{\mathrel}{\M@arrows@font}{"21E0}
\DeclareMathSymbol{\leftharpoonup}{\mathrel}{\M@arrows@font}{"21BC}
\DeclareMathSymbol{\leftharpoondown}{\mathrel}{\M@arrows@font}{"21BD}
\DeclareMathSymbol{\leftarrowtail}{\mathrel}{\M@arrows@font}{"21A2}
\DeclareMathSymbol{\leftoplusarrow}{\mathrel}{\M@arrows@font}{"2B32}
\DeclareMathSymbol{\leftwavearrow}{\mathrel}{\M@arrows@font}{"219C}
\DeclareMathSymbol{\leftsquigarrow}{\mathrel}{\M@arrows@font}{"21DC}
\DeclareMathSymbol{\longleftsquigarrow}{\mathrel}{\M@arrows@font}{"2B33}
\DeclareMathSymbol{\looparrowleft}{\mathrel}{\M@arrows@font}{"21AB}
\DeclareMathSymbol{\curvearrowleft}{\mathrel}{\M@arrows@font}{"293A}
\DeclareMathSymbol{\circlearrowleft}{\mathrel}{\M@arrows@font}{"21BA}
\DeclareMathSymbol{\twoheadleftarrow}{\mathrel}{\M@arrows@font}{"219E}
\DeclareMathSymbol{\leftarrowtobar}{\mathrel}{\M@arrows@font}{"21E4}
\DeclareMathSymbol{\leftwhitearrow}{\mathrel}{\M@arrows@font}{"21E6}
\DeclareMathSymbol{\leftleftarrows}{\mathrel}{\M@arrows@font}{"21C7}
\DeclareMathSymbol{\leftleftleftarrows}{\mathrel}{\M@arrows@font}{"2B31}
\DeclareMathSymbol{\leftrightarrow}{\mathrel}{\M@arrows@font}{"2194}
\DeclareMathSymbol{\Leftrightarrow}{\mathrel}{\M@arrows@font}{"21D4}
\DeclareMathSymbol{\nLeftrightarrow}{\mathrel}{\M@arrows@font}{"21CE}
\DeclareMathSymbol{\longleftrightarrow}{\mathrel}{\M@arrows@font}{"27F7}
\DeclareMathSymbol{\Longleftrightarrow}{\mathrel}{\M@arrows@font}{"27FA}
\DeclareMathSymbol{\leftrightwavearrow}{\mathrel}{\M@arrows@font}{"21AD}
\DeclareMathSymbol{\leftrightarrows}{\mathrel}{\M@arrows@font}{"21C6}
\DeclareMathSymbol{\leftrightharpoons}{\mathrel}{\M@arrows@font}{"21CB}
\DeclareMathSymbol{\leftrightarrowstobar}{\mathrel}{\M@arrows@font}{"21B9}
\DeclareMathSymbol{\rightleftarrows}{\mathrel}{\M@arrows@font}{"21C4}
\DeclareMathSymbol{\rightleftharpoons}{\mathrel}{\M@arrows@font}{"21CC}
\DeclareMathSymbol{\uparrow}{\mathrel}{\M@arrows@font}{"2191}
\DeclareMathSymbol{\Uparrow}{\mathrel}{\M@arrows@font}{"21D1}
\DeclareMathSymbol{\Uuparrow}{\mathrel}{\M@arrows@font}{"290A}
\DeclareMathSymbol{\upbararrow}{\mathrel}{\M@arrows@font}{"21A5}
\DeclareMathSymbol{\updasharrow}{\mathrel}{\M@arrows@font}{"21E1}
\DeclareMathSymbol{\upharpoonleft}{\mathrel}{\M@arrows@font}{"21BF}
\DeclareMathSymbol{\upharpoonright}{\mathrel}{\M@arrows@font}{"21BE}
\DeclareMathSymbol{\twoheaduparrow}{\mathrel}{\M@arrows@font}{"219F}
\DeclareMathSymbol{\uparrowtobar}{\mathrel}{\M@arrows@font}{"2912}
```

```
\DeclareMathSymbol{\upwhitearrow}{\mathrel}{\M@arrows@font}{"21E7}
\DeclareMathSymbol{\upwhitebararrow}{\mathrel}{\M@arrows@font}{"21EA}
\DeclareMathSymbol{\upuparrows}{\mathrel}{\M@arrows@font}{"21C8}
\DeclareMathSymbol{\downarrow}{\mathrel}{\M@arrows@font}{"2193}
\DeclareMathSymbol{\Downarrow}{\mathrel}{\M@arrows@font}{"21D3}
\DeclareMathSymbol{\Ddownarrow}{\mathrel}{\M@arrows@font}{"290B}
\DeclareMathSymbol{\downbararrow}{\mathrel}{\M@arrows@font}{"21A7}
\DeclareMathSymbol{\downdasharrow}{\mathrel}{\M@arrows@font}{"21E3}
\DeclareMathSymbol{\zigzagarrow}{\mathrel}{\M@arrows@font}{"21AF}
  \global\let\lightningboltarrow\zigzagarrow
\DeclareMathSymbol{\downharpoonleft}{\mathrel}{\M@arrows@font}{"21C3}
\DeclareMathSymbol{\downharpoonright}{\mathrel}{\M@arrows@font}{"21C2}
\DeclareMathSymbol{\twoheaddownarrow}{\mathrel}{\M@arrows@font}{"21A1}
\DeclareMathSymbol{\downarrowtobar}{\mathrel}{\M@arrows@font}{"2913}
\DeclareMathSymbol{\downwhitearrow}{\mathrel}{\M@arrows@font}{"21E9}
\DeclareMathSymbol{\downdownarrows}{\mathrel}{\M@arrows@font}{"21CA}
\DeclareMathSymbol{\updownarrow}{\mathrel}{\M@arrows@font}{"2195}
\DeclareMathSymbol{\Updownarrow}{\mathrel}{\M@arrows@font}{"21D5}
\DeclareMathSymbol{\updownarrows}{\mathrel}{\M@arrows@font}{"21C5}
\DeclareMathSymbol{\downuparrows}{\mathrel}{\M@arrows@font}{"21F5}
\DeclareMathSymbol{\updownharpoons}{\mathrel}{\M@arrows@font}{"296E}
\DeclareMathSymbol{\downupharpoons}{\mathrel}{\M@arrows@font}{"296F}
\DeclareMathSymbol{\nearrow}{\mathrel}{\M@arrows@font}{"2197}
\DeclareMathSymbol{\Nearrow}{\mathrel}{\M@arrows@font}{"21D7}
\DeclareMathSymbol{\nwarrow}{\mathrel}{\M@arrows@font}{"2196}
\DeclareMathSymbol{\Nwarrow}{\mathrel}{\M@arrows@font}{"21D6}
\DeclareMathSymbol{\searrow}{\mathrel}{\M@arrows@font}{"2198}
\DeclareMathSymbol{\Searrow}{\mathrel}{\M@arrows@font}{"21D8}
\DeclareMathSymbol{\swarrow}{\mathrel}{\M@arrows@font}{"2199}
\DeclareMathSymbol{\Swarrow}{\mathrel}{\M@arrows@font}{"21D9}
\DeclareMathSymbol{\nwsearrow}{\mathrel}{\M@arrows@font}{"2921}
\DeclareMathSymbol{\neswarrow}{\mathrel}{\M@arrows@font}{"2922}
\DeclareMathSymbol{\lcirclearrow}{\mathrel}{\M@arrows@font}{"27F2}
\DeclareMathSymbol{\rcirclearrow}{\mathrel}{\M@arrows@font}{"27F3}}
```

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

```
\def\M@bb@set{%
  \protected\def\mathbb##1{\relax
    \ifmmode\else
      \M@HModeError\mathbb
      $%
    \fi
    \begingroup
      \M@bb@mathcodes
      ##1%
    \endgroup}
  \edef\M@bb@num{\number\csname symM\M@bbshape\@tempa\endcsname}
  \protected\edef\M@bb@mathcodes{%
  \Umathcode`A=0+\M@bb@num"1D538\relax
  \Umathcode`B=0+\M@bb@num"1D539\relax
  \Umathcode`C=0+\M@bb@num"2102\relax
  \Umathcode`D=0+\M@bb@num"1D53B\relax
  \Umathcode`E=0+\M@bb@num"1D53C\relax
  \Umathcode`F=0+\M@bb@num"1D53D\relax
  \Umathcode`G=0+\M@bb@num"1D53E\relax
  \Umathcode`H=0+\M@bb@num"210D\relax
  \Umathcode`I=0+\M@bb@num"1D540\relax
  \Umathcode`J=0+\M@bb@num"1D541\relax
  \Umathcode`K=0+\M@bb@num"1D542\relax
  \Umathcode`L=0+\M@bb@num"1D543\relax
  \Umathcode`M=0+\M@bb@num"1D544\relax
  \Umathcode`N=0+\M@bb@num"2115\relax
  \Umathcode`O=0+\M@bb@num"1D546\relax
  \Umathcode`P=0+\M@bb@num"2119\relax
  \Umathcode`Q=0+\M@bb@num"211A\relax
  \Umathcode`R=0+\M@bb@num"211D\relax
  \Umathcode`S=0+\M@bb@num"1D54A\relax
  \Umathcode`T=0+\M@bb@num"1D54B\relax
  \Umathcode`U=0+\M@bb@num"1D54C\relax
  \Umathcode`V=0+\M@bb@num"1D54D\relax
  \Umathcode`W=0+\M@bb@num"1D54E\relax
  \Umathcode`X=0+\M@bb@num"1D54F\relax
  \Umathcode`Y=0+\M@bb@num"1D550\relax
  \Umathcode`Z=0+\M@bb@num"2124\relax
  \Umathcode`a=0+\M@bb@num"1D552\relax
  \Umathcode`b=0+\M@bb@num"1D553\relax
  \Umathcode`c=0+\M@bb@num"1D554\relax
  \Umathcode`d=0+\M@bb@num"1D555\relax
  \Umathcode`e=0+\M@bb@num"1D556\relax
  \Umathcode`f=0+\M@bb@num"1D557\relax
  \Umathcode`g=0+\M@bb@num"1D558\relax
  \Umathcode`h=0+\M@bb@num"1D559\relax
  \Umathcode`i=0+\M@bb@num"1D55A\relax
  \Umathcode`j=0+\M@bb@num"1D55B\relax
  \Umathcode`k=0+\M@bb@num"1D55C\relax
  \Umathcode`l=0+\M@bb@num"1D55D\relax
  \Umathcode`m=0+\M@bb@num"1D55E\relax
  \Umathcode`n=0+\M@bb@num"1D55F\relax
  \Umathcode`o=0+\M@bb@num"1D560\relax
```

```
  \Umathcode`p=0+\M@bb@num"1D561\relax
  \Umathcode`q=0+\M@bb@num"1D562\relax
  \Umathcode`r=0+\M@bb@num"1D563\relax
  \Umathcode`s=0+\M@bb@num"1D564\relax
  \Umathcode`t=0+\M@bb@num"1D565\relax
  \Umathcode`u=0+\M@bb@num"1D566\relax
  \Umathcode`v=0+\M@bb@num"1D567\relax
  \Umathcode`w=0+\M@bb@num"1D568\relax
  \Umathcode`x=0+\M@bb@num"1D569\relax
  \Umathcode`y=0+\M@bb@num"1D56A\relax
  \Umathcode`z=0+\M@bb@num"1D56B\relax
  \Umathcode`0=0+\M@bb@num"1D7D8\relax
  \Umathcode`1=0+\M@bb@num"1D7D9\relax
  \Umathcode`2=0+\M@bb@num"1D7DA\relax
  \Umathcode`3=0+\M@bb@num"1D7DB\relax
  \Umathcode`4=0+\M@bb@num"1D7DC\relax
  \Umathcode`5=0+\M@bb@num"1D7DD\relax
  \Umathcode`6=0+\M@bb@num"1D7DE\relax
  \Umathcode`7=0+\M@bb@num"1D7DF\relax
  \Umathcode`8=0+\M@bb@num"1D7E0\relax
  \Umathcode`9=0+\M@bb@num"1D7E1\relax}}
```

Set caligraphic letters.

```
\def\M@cal@set{%
  \protected\def\mathcal##1{\relax
    \ifmmode\else
      \M@HModeError\mathcal
      $%
    \fi
    \begingroup
      \M@cal@mathcodes
      ##1%
    \endgroup}
  \edef\M@cal@num{\number\csname symM\M@calshape\@tempa\endcsname}
  \protected\edef\M@cal@mathcodes{%
  \Umathcode`A=0+\M@cal@num"1D49C\relax
  \Umathcode`B=0+\M@cal@num"212C\relax
  \Umathcode`C=0+\M@cal@num"1D49E\relax
  \Umathcode`D=0+\M@cal@num"1D49F\relax
  \Umathcode`E=0+\M@cal@num"2130\relax
  \Umathcode`F=0+\M@cal@num"2131\relax
  \Umathcode`G=0+\M@cal@num"1D4A2\relax
  \Umathcode`H=0+\M@cal@num"210B\relax
  \Umathcode`I=0+\M@cal@num"2110\relax
  \Umathcode`J=0+\M@cal@num"1D4A5\relax
  \Umathcode`K=0+\M@cal@num"1D4A6\relax
  \Umathcode`L=0+\M@cal@num"2112\relax
  \Umathcode`M=0+\M@cal@num"2133\relax
  \Umathcode`N=0+\M@cal@num"1D4A9\relax
  \Umathcode`O=0+\M@cal@num"1D4AA\relax
  \Umathcode`P=0+\M@cal@num"1D4AB\relax
  \Umathcode`Q=0+\M@cal@num"1D4AC\relax
  \Umathcode`R=0+\M@cal@num"211B\relax
  \Umathcode`S=0+\M@cal@num"1D4AE\relax
  \Umathcode`T=0+\M@cal@num"1D4AF\relax
  \Umathcode`U=0+\M@cal@num"1D4B0\relax
  \Umathcode`V=0+\M@cal@num"1D4B1\relax
  \Umathcode`W=0+\M@cal@num"1D4B2\relax
  \Umathcode`X=0+\M@cal@num"1D4B3\relax
  \Umathcode`Y=0+\M@cal@num"1D4B4\relax
  \Umathcode`Z=0+\M@cal@num"1D4B5\relax
  \Umathcode`a=0+\M@cal@num"1D4B6\relax
  \Umathcode`b=0+\M@cal@num"1D4B7\relax
  \Umathcode`c=0+\M@cal@num"1D4B8\relax
  \Umathcode`d=0+\M@cal@num"1D4B9\relax
  \Umathcode`e=0+\M@cal@num"212F\relax
  \Umathcode`f=0+\M@cal@num"1D4BB\relax
  \Umathcode`g=0+\M@cal@num"210A\relax
  \Umathcode`h=0+\M@cal@num"1D4BD\relax
  \Umathcode`i=0+\M@cal@num"1D4BE\relax
  \Umathcode`j=0+\M@cal@num"1D4BF\relax
  \Umathcode`k=0+\M@cal@num"1D4C0\relax
  \Umathcode`l=0+\M@cal@num"1D4C1\relax
  \Umathcode`m=0+\M@cal@num"1D4C2\relax
  \Umathcode`n=0+\M@cal@num"1D4C3\relax
  \Umathcode`o=0+\M@cal@num"2134\relax
  \Umathcode`p=0+\M@cal@num"1D4C5\relax
  \Umathcode`q=0+\M@cal@num"1D4C6\relax
  \Umathcode`r=0+\M@cal@num"1D4C7\relax
  \Umathcode`s=0+\M@cal@num"1D4C8\relax
  \Umathcode`t=0+\M@cal@num"1D4C9\relax
  \Umathcode`u=0+\M@cal@num"1D4CA\relax
  \Umathcode`v=0+\M@cal@num"1D4CB\relax
  \Umathcode`w=0+\M@cal@num"1D4CC\relax
  \Umathcode`x=0+\M@cal@num"1D4CD\relax
  \Umathcode`y=0+\M@cal@num"1D4CE\relax
  \Umathcode`z=0+\M@cal@num"1D4CF\relax}}
```

Set fraktur letters.

```
\def\M@frak@set{%
  \protected\def\mathfrak##1{\relax
    \ifmmode\else
      \M@HModeError\mathfrak
      $%
    \fi
    \begingroup
      \M@frak@mathcodes
      ##1%
    \endgroup}
  \edef\M@frak@num{\number\csname symM\M@frakshape\@tempa\endcsname}
  \protected\edef\M@frak@mathcodes{%
  \Umathcode`A=0+\M@frak@num"1D504\relax
  \Umathcode`B=0+\M@frak@num"1D505\relax
  \Umathcode`C=0+\M@frak@num"212D\relax
  \Umathcode`D=0+\M@frak@num"1D507\relax
  \Umathcode`E=0+\M@frak@num"1D508\relax
  \Umathcode`F=0+\M@frak@num"1D509\relax
  \Umathcode`G=0+\M@frak@num"1D50A\relax
  \Umathcode`H=0+\M@frak@num"210C\relax
  \Umathcode`I=0+\M@frak@num"2111\relax
  \Umathcode`J=0+\M@frak@num"1D50D\relax
  \Umathcode`K=0+\M@frak@num"1D50E\relax
  \Umathcode`L=0+\M@frak@num"1D50F\relax
  \Umathcode`M=0+\M@frak@num"1D510\relax
  \Umathcode`N=0+\M@frak@num"1D511\relax
  \Umathcode`O=0+\M@frak@num"1D512\relax
  \Umathcode`P=0+\M@frak@num"1D513\relax
  \Umathcode`Q=0+\M@frak@num"1D514\relax
  \Umathcode`R=0+\M@frak@num"211C\relax
  \Umathcode`S=0+\M@frak@num"1D516\relax
  \Umathcode`T=0+\M@frak@num"1D517\relax
  \Umathcode`U=0+\M@frak@num"1D518\relax
  \Umathcode`V=0+\M@frak@num"1D519\relax
  \Umathcode`W=0+\M@frak@num"1D51A\relax
  \Umathcode`X=0+\M@frak@num"1D51B\relax
  \Umathcode`Y=0+\M@frak@num"1D51C\relax
  \Umathcode`Z=0+\M@frak@num"2128\relax
  \Umathcode`a=0+\M@frak@num"1D51E\relax
  \Umathcode`b=0+\M@frak@num"1D51F\relax
  \Umathcode`c=0+\M@frak@num"1D520\relax
  \Umathcode`d=0+\M@frak@num"1D521\relax
  \Umathcode`e=0+\M@frak@num"1D522\relax
  \Umathcode`f=0+\M@frak@num"1D523\relax
  \Umathcode`g=0+\M@frak@num"1D524\relax
  \Umathcode`h=0+\M@frak@num"1D525\relax
  \Umathcode`i=0+\M@frak@num"1D526\relax
  \Umathcode`j=0+\M@frak@num"1D527\relax
  \Umathcode`k=0+\M@frak@num"1D528\relax
  \Umathcode`l=0+\M@frak@num"1D529\relax
  \Umathcode`m=0+\M@frak@num"1D52A\relax
  \Umathcode`n=0+\M@frak@num"1D52B\relax
  \Umathcode`o=0+\M@frak@num"1D52C\relax
  \Umathcode`p=0+\M@frak@num"1D52D\relax
```

```
  \Umathcode`q=0+\M@frak@num"1D52E\relax
  \Umathcode`r=0+\M@frak@num"1D52F\relax
  \Umathcode`s=0+\M@frak@num"1D530\relax
  \Umathcode`t=0+\M@frak@num"1D531\relax
  \Umathcode`u=0+\M@frak@num"1D532\relax
  \Umathcode`v=0+\M@frak@num"1D533\relax
  \Umathcode`w=0+\M@frak@num"1D534\relax
  \Umathcode`x=0+\M@frak@num"1D535\relax
  \Umathcode`y=0+\M@frak@num"1D536\relax
  \Umathcode`z=0+\M@frak@num"1D537\relax}}
```

Set bold caligraphic letters.

```
\def\M@bcal@set{%
  \protected\def\mathbcal##1{\relax
    \ifmmode\else
      \M@HModeError\mathbcal
      $%
    \fi
    \begingroup
      \M@bcal@mathcodes
      ##1%
    \endgroup}
  \edef\M@bcal@num{\number\csname symM\M@bcalshape\@tempa\endcsname}
  \protected\edef\M@bcal@mathcodes{%
  \Umathcode`A=0+\M@bcal@num"1D4D0\relax
  \Umathcode`B=0+\M@bcal@num"1D4D1\relax
  \Umathcode`C=0+\M@bcal@num"1D4D2\relax
  \Umathcode`D=0+\M@bcal@num"1D4D3\relax
  \Umathcode`E=0+\M@bcal@num"1D4D4\relax
  \Umathcode`F=0+\M@bcal@num"1D4D5\relax
  \Umathcode`G=0+\M@bcal@num"1D4D6\relax
  \Umathcode`H=0+\M@bcal@num"1D4D7\relax
  \Umathcode`I=0+\M@bcal@num"1D4D8\relax
  \Umathcode`J=0+\M@bcal@num"1D4D9\relax
  \Umathcode`K=0+\M@bcal@num"1D4DA\relax
  \Umathcode`L=0+\M@bcal@num"1D4DB\relax
  \Umathcode`M=0+\M@bcal@num"1D4DC\relax
  \Umathcode`N=0+\M@bcal@num"1D4DD\relax
  \Umathcode`O=0+\M@bcal@num"1D4DE\relax
  \Umathcode`P=0+\M@bcal@num"1D4DF\relax
  \Umathcode`Q=0+\M@bcal@num"1D4E0\relax
  \Umathcode`R=0+\M@bcal@num"1D4E1\relax
  \Umathcode`S=0+\M@bcal@num"1D4E2\relax
  \Umathcode`T=0+\M@bcal@num"1D4E3\relax
  \Umathcode`U=0+\M@bcal@num"1D4E4\relax
  \Umathcode`V=0+\M@bcal@num"1D4E5\relax
  \Umathcode`W=0+\M@bcal@num"1D4E6\relax
  \Umathcode`X=0+\M@bcal@num"1D4E7\relax
  \Umathcode`Y=0+\M@bcal@num"1D4E8\relax
  \Umathcode`Z=0+\M@bcal@num"1D4E9\relax
  \Umathcode`a=0+\M@bcal@num"1D4EA\relax
  \Umathcode`b=0+\M@bcal@num"1D4EB\relax
  \Umathcode`c=0+\M@bcal@num"1D4EC\relax
  \Umathcode`d=0+\M@bcal@num"1D4ED\relax
  \Umathcode`e=0+\M@bcal@num"1D4EE\relax
  \Umathcode`f=0+\M@bcal@num"1D4EF\relax
  \Umathcode`g=0+\M@bcal@num"1D4F0\relax
  \Umathcode`h=0+\M@bcal@num"1D4F1\relax
  \Umathcode`i=0+\M@bcal@num"1D4F2\relax
  \Umathcode`j=0+\M@bcal@num"1D4F3\relax
  \Umathcode`k=0+\M@bcal@num"1D4F4\relax
  \Umathcode`l=0+\M@bcal@num"1D4F5\relax
  \Umathcode`m=0+\M@bcal@num"1D4F6\relax
  \Umathcode`n=0+\M@bcal@num"1D4F7\relax
  \Umathcode`o=0+\M@bcal@num"1D4F8\relax
  \Umathcode`p=0+\M@bcal@num"1D4F9\relax
  \Umathcode`q=0+\M@bcal@num"1D4FA\relax
  \Umathcode`r=0+\M@bcal@num"1D4FB\relax
  \Umathcode`s=0+\M@bcal@num"1D4FC\relax
  \Umathcode`t=0+\M@bcal@num"1D4FD\relax
  \Umathcode`u=0+\M@bcal@num"1D4FE\relax
  \Umathcode`v=0+\M@bcal@num"1D4FF\relax
  \Umathcode`w=0+\M@bcal@num"1D500\relax
  \Umathcode`x=0+\M@bcal@num"1D501\relax
  \Umathcode`y=0+\M@bcal@num"1D502\relax
  \Umathcode`z=0+\M@bcal@num"1D503\relax}}
```

Set bold fraktur letters.

```
\def\M@bfrak@set{%
  \protected\def\mathbfrak##1{\relax
    \ifmmode\else
      \M@HModeError\mathbfrak
      $%
    \fi
    \begingroup
      \M@bfrak@mathcodes
      ##1%
    \endgroup}
  \edef\M@bfrak@num{\number\csname symM\M@bfrakshape\@tempa\endcsname}
  \protected\edef\M@bfrak@mathcodes{%
  \Umathcode`A=0+\M@bfrak@num"1D56C\relax
  \Umathcode`B=0+\M@bfrak@num"1D56D\relax
  \Umathcode`C=0+\M@bfrak@num"1D56E\relax
  \Umathcode`D=0+\M@bfrak@num"1D56F\relax
  \Umathcode`E=0+\M@bfrak@num"1D570\relax
  \Umathcode`F=0+\M@bfrak@num"1D571\relax
  \Umathcode`G=0+\M@bfrak@num"1D572\relax
  \Umathcode`H=0+\M@bfrak@num"1D573\relax
  \Umathcode`I=0+\M@bfrak@num"1D574\relax
  \Umathcode`J=0+\M@bfrak@num"1D575\relax
  \Umathcode`K=0+\M@bfrak@num"1D576\relax
  \Umathcode`L=0+\M@bfrak@num"1D577\relax
  \Umathcode`M=0+\M@bfrak@num"1D578\relax
  \Umathcode`N=0+\M@bfrak@num"1D579\relax
  \Umathcode`O=0+\M@bfrak@num"1D57A\relax
  \Umathcode`P=0+\M@bfrak@num"1D57B\relax
  \Umathcode`Q=0+\M@bfrak@num"1D57C\relax
  \Umathcode`R=0+\M@bfrak@num"1D57D\relax
  \Umathcode`S=0+\M@bfrak@num"1D57E\relax
  \Umathcode`T=0+\M@bfrak@num"1D57F\relax
  \Umathcode`U=0+\M@bfrak@num"1D580\relax
  \Umathcode`V=0+\M@bfrak@num"1D581\relax
  \Umathcode`W=0+\M@bfrak@num"1D582\relax
  \Umathcode`X=0+\M@bfrak@num"1D583\relax
  \Umathcode`Y=0+\M@bfrak@num"1D584\relax
  \Umathcode`Z=0+\M@bfrak@num"1D585\relax
  \Umathcode`a=0+\M@bfrak@num"1D586\relax
  \Umathcode`b=0+\M@bfrak@num"1D587\relax
  \Umathcode`c=0+\M@bfrak@num"1D588\relax
  \Umathcode`d=0+\M@bfrak@num"1D589\relax
  \Umathcode`e=0+\M@bfrak@num"1D58A\relax
  \Umathcode`f=0+\M@bfrak@num"1D58B\relax
  \Umathcode`g=0+\M@bfrak@num"1D58C\relax
  \Umathcode`h=0+\M@bfrak@num"1D58D\relax
  \Umathcode`i=0+\M@bfrak@num"1D58E\relax
  \Umathcode`j=0+\M@bfrak@num"1D58F\relax
  \Umathcode`k=0+\M@bfrak@num"1D590\relax
  \Umathcode`l=0+\M@bfrak@num"1D591\relax
  \Umathcode`m=0+\M@bfrak@num"1D592\relax
  \Umathcode`n=0+\M@bfrak@num"1D593\relax
  \Umathcode`o=0+\M@bfrak@num"1D594\relax
  \Umathcode`p=0+\M@bfrak@num"1D595\relax
  \Umathcode`q=0+\M@bfrak@num"1D596\relax
  \Umathcode`r=0+\M@bfrak@num"1D597\relax
  \Umathcode`s=0+\M@bfrak@num"1D598\relax
  \Umathcode`t=0+\M@bfrak@num"1D599\relax
  \Umathcode`u=0+\M@bfrak@num"1D59A\relax
  \Umathcode`v=0+\M@bfrak@num"1D59B\relax
  \Umathcode`w=0+\M@bfrak@num"1D59C\relax
  \Umathcode`x=0+\M@bfrak@num"1D59D\relax
  \Umathcode`y=0+\M@bfrak@num"1D59E\relax
  \Umathcode`z=0+\M@bfrak@num"1D59F\relax}}
```

And that's everything!

Version History

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

1.1b . . . . July 2018
—initial release

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

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

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

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

1.6 . . . . December 2019
—separated implementation and user documentation
—created `mathfont_heading.tex`
—created `mathfont_doc_patch.tex` for use with the index
—changed `mathfont_greek.pdf` to `mathfont_symbol_list.pdf`
—eliminated `mathfont_example.pdf`
—eliminated `operators` package option
—eliminated `packages` package option
—font name can be package option
—added Hebrew and Cyrillic characters
—separated ancient Greek from modern Greek characters
—created new keywords: `extsymbols`, `delimiters`, `arrows`, `diacritics`, `bigops`, `extbigops`
—improved messaging
—improved internal code for local font-change commands
—improved space parsing for the optional argument of `\mathfont`
—bug fix for `\#`, etc. commands
—bad input for `\mathbb`, etc. now gives a warning
—improved error checking for `\newmathrm`, etc. commands
—`\mathfont` now ignores bad options (on top of issuing an error)
—iternal commands now begin with `\M@...`
—added Easter Egg!
—improved indexing
—`mathfont.dtx` renamed as `mathfont_code.dtx`
—`\newmathbold` renamed as `\newmathbf`
—default local font changes now use `\updefault`, etc.
—added fatal error for missing fontspec
—fatal errors result in `\endinput` rather than `\@@end`

2.0 . . . . December 2021

Big Change: Font adjustments for 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 `\Umathcode`s of letters instead of `\M@`⟨*bb,etc.*⟩`@`⟨*letter*⟩ commands
—removed warnings about non-letter contents of `\mathbb`, etc.
—fonts loaded twice, once with default settings (for text) and once in base mode (for math)
—`mathconstantsfont` accepts “upright” or “italic” as optional argument

2.2 . . . . December 2022
—changed the easter egg text
—updated patch for `\DeclareSymbolFont` to work with changes to the kernel (eliminated `\M@p@tch@decl@re` error message)
—calling Plain TEX on `mathfont_code.dtx` produces sty file and no pdf file

2.2a . . . . December 2022
—bug fix for `\mathconstantsfont`
—bug fix for `\M@check@int`
—added `doc2` option to `ltxdoc` in `mathfont_code.dtx`

2.2b . . . . August 2023
—minor changes to code and documentation
—`\ng` now works in math (as not greater than symbol) and text (as pronounciation symbol)

2.3 . . . . September 2023
—solidus and `\fractionslash` are `\mathord` instead of `\mathbin`
—removed `\mathfont{fontspec}` functionality
—redesigned font-loader
—added package options `default-loader` and `fontspec-loader`

Index

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