Experimental unicode mathematical typesetting: The unicode-math package

Will Robertson

2006/02/20 v0.01

Contents

1 Introduction 1

2 Specification 2
2.1 Dealing with real life 2

I The unicode-math package 2

3 Things we need 3
3.1 Packages 3
3.2 Counters and conditionals 3
3.3 Programming macros 3
3.4 Overcoming `\@onlypreamble` 3

4 Fundamentals 4
4.1 Enlarging the number of maths families 4
4.2 `\DeclareMathSymbol` for unicode ranges 4
4.3 User interface to `\DeclareSymbolFont` 6
4.4 Big operators and radicals 8
4.5 Delimiters 11
4.6 Maths accents 11
4.7 Setting up the ASCII ranges 11

5 fontspec feature hooks 12
5.1 OpenType maths font features 13
5.2 Range processing 13

II STIX table data extraction 15

A Documenting maths support in the NFSS 16
A.1 Overview 16
A.2 Detailed code investigation 17
A.3 Maths symbols 17
A.4 Delimiters 19
A.5 Symbol fonts 22

1 Introduction

This document describes the unicode-math package, which is an *experimental* implementation of a macro to unicode glyph encoding for mathematical characters. Its intended use is for XeTeX, although it is conjectured that small effect needs to be spent to create a cross-format package that would also work with □

As of XeTeX v. 0.995, maths characters can be accessed in unicode ranges. Now, a proper method must be invented for real unicode maths support. Before any code is written, I'm writing a specification in order to work out what is required. Fairly significant pieces of the NFSS may have to be re-written, and I'm a little unsure where to start.

2 Specification

In the ideal case, a single unicode font will contain all maths glyphs we need. Barbara Beeton's STIX table provides the mapping between unicode maths glyphs and macro names (all 3298 of them!). A single command

```
\setmathsfont[⟨font features⟩]{⟨font name⟩}
```

would implement this for every every symbol and alphabetic variant. That means `x` to x, `\xi` to ξ, `\leq` to $\leq$, etc., `\mathcal{H}` to $\mathcal{H}$ and so on, all for unicode glyphs within a single font.

Furthermore, this package should deal well with unicode characters for maths input, as well. This includes using literal Greek letters in formulae, resolving to upright or italic depending on preference.

Finally, maths versions must also be provided for. While I guess version selection in LaTeX will remain the same, the specification for choosing the version fonts will probably be an optional argument:

```
\setmathsfont[Version=Bold,⟨font features⟩]{⟨font name⟩}
```

All instances of 'maths' in command names will be aliased to 'math' for our American (or abbreviatory-minded) friends. Instances above of

```
[⟨font features⟩]{⟨font name⟩}
```

follow from my `fontspec` package, and therefore any additional ⟨*font features*⟩ specific to maths fonts will hook into `fontspec`'s methods.

2.1 Dealing with real life

Let's face it; there will probably be few cases where a single unicode maths font suffices. The upcoming STIX font comes to mind as a notable exception. It will therefore be necessary to delegate specific unicode ranges of glyphs to separate fonts. This syntax will also hook into the `fontspec` font feature processing:

```
\setmathsfont[Range=⟨unicode range⟩,⟨font features⟩]{⟨font name⟩}
```

where ⟨*unicode range*⟩ is a comma-separated list of unicode slots and ranges such as `{27D0-27EB, 27FF, 295B-297F}`. Furthermore, preset names ranges could be used, such as `MiscMathSymbolsA`, with such ranges based on unicode chunks. The amount of optimisation required here to achieve acceptable performance has yet to be determined. Techniques such as saving out unicode subsets based on ⟨*unicode range*⟩ data to be `\input` in the next LaTeX run are a possibility, but at this stage, performance without such measures seems acceptable.

File I

The unicode-math package

This is the package.

```
\ProvidesPackage{unicode-math}
  [2006/02/20 v0.01 Unicode maths definitions]
```

3 Things we need

3.1 Packages

```
\RequirePackage{fontspec}
```

3.2 Counters and conditionals

```
\newcounter{um@fam}
\newif\if@um@fontspec@feature
```

3.3 Programming macros

\um@Loop
\um@Break

See Kees van der Laan's various articles on TEX programming:

```
\def\um@Loop#1\um@Pool{#1\um@Loop#1\um@Pool}
\def\um@Break#1\um@Pool{}
```

\um@FOR

A simple 'for' loop implemented with the above. Takes a (predefined) counter \csname and increments it between two integers, iterating as we go.

```
\long\def\um@FOR #1 = [#2:#3] #4{%
  {\csname#1\endcsname =#2\relax
   \um@Loop #4%
     \expandafter\advance\csname#1\endcsname\@ne
     \expandafter\ifnum\csname#1\endcsname>#3\relax
       \expandafter\um@Break
     \fi
   \um@Pool}}
```

g/h/i/j/k/l/m/

```
\newcount\@ii
\um@FOR @ii = [7:13] {\@alph\@ii/}
```

3.4 Overcoming \@onlypreamble

This will be refined later!

```
\def\@preamblecmds{}
```

4 Fundamentals

4.1 Enlarging the number of maths families

To start with, we've got a power of two as many `\fams` as before. So (from `ltfssbas.dtx`) we want to redefine

```
\def\new@mathgroup{\alloc@8\mathgroup\chardef\@cclvi}
\let\newfam\new@mathgroup
```

Up to math fam 25 of 255.

```
\um@FOR @tempcnta = [1:20]
  {\expandafter\newfam
     \csname mt\@alph\@tempcnta\endcsname}
Up to math fam \the\mtt\ of 255.
```

This is sufficient for LaTeX's `\DeclareSymbolFont`-type commands to be able to define 256 named maths fonts. Now we need a new `\DeclareMathSymbol`.

4.2 `\DeclareMathSymbol` for unicode ranges

This is mostly an adaptation from LaTeX's definition.

`\DeclareUnicodeMathSymbol`

`#1` : Symbol, *e.g.*, `\alpha` or `a`
`#2` : Type, *e.g.*, `\mathalpha`
`#3` : Math font name, *e.g.*, `operators`
`#4` : Slot, *e.g.*, `"221E`

```
\def\DeclareUnicodeMathSymbol#1#2#3#4{%
```

First ensure the math font (*e.g.*, `operators`) exists:

```
  \expandafter\in@\csname sym#3\expandafter\endcsname
     \expandafter{\group@list}%
  \ifin@
```

No longer need here to perform the obfuscated hex conversion, since `\XeTeXmathchar` (and friends) has a more simplified input than TeX's `\mathchar`.

```
    \begingroup
```

The symbol to be defined can be either a command (`\alpha`) or a character (`a`). Branch for the former:

```
      \if\relax\noexpand#1% is command?
     \edef\reserved@a{\noexpand\in@{\string\XeTeXmathchar}{\meaning#1}}%
        \reserved@a
```

If the symbol command definition contains `\XeTeXmathchar`, then we can provide the info that a previous symbol definition is being overwritten:

```
        \ifin@
          \expandafter\um@set@mathsymbol
             \csname sym#3\endcsname#1#2{#4}%
          \@font@info{Redeclaring math symbol \string#1}%
```

Otherwise, overwrite it if the symbol command definition contains plain old `\mathchar`:

```
        \else
         %\edef\reserved@a{\noexpand\in@{\string\mathchar}{\meaning#1}}%
          %\reserved@a
          %\ifin@
          %  \expandafter\set@xmathsymbol
          %     \csname sym#3\endcsname#1#2{#4}%
```

Otherwise, throw an error if the command name is already taken by a non-symbol definition:

```
          %\else
            %\expandafter\ifx
            %\csname\expandafter\@gobble\string#1\endcsname
            %\relax
              \expandafter\um@set@mathsymbol
                 \csname sym#3\endcsname#1#2{#4}%
            %\else
            %  \@latex@error{Command `\string#1' already defined}\@eha
            %\fi
          %\fi
        \fi
```

And if the symbol input is a character:

```
      \else
        \expandafter\um@set@mathchar
          \csname sym#3\endcsname#1#2{#4}%
      \fi
    \endgroup
```

Everything previous was skipped if the maths font doesn't exist in the first place:

```
  \else
    \@latex@error{Symbol font `#3' is not defined}\@eha
  \fi}
```

The final macros that actually define the maths symbol with XƎTEX primitives.

`\um@set@mathsymbol`

`#1` : Symbol font number
`#2` : Symbol macro, *e.g.*, `\alpha`
`#3` : Type, *e.g.*, `\mathalpha`
`#4` : Slot, *e.g.*, `"221E`

If the symbol definition is for a macro. There are a bunch of tests to perform to process the various characters.

```
\def\um@set@mathsymbol#1#2#3#4{%
```

First test if the character requires a `\nolimits` suffix. This is controlled by the `\um@nolimits` macro, which contains a commalist of such characters. If so, define the mathchar `\`⟨***cs***⟩`op` (where `#2` is `\`⟨***cs***⟩) and define `\`⟨***cs***⟩ as the wrapper around this control sequence.

```
  \expandafter\in@\expandafter#2\expandafter{\um@nolimits}%
  \ifin@
```

```
    \expandafter\global\expandafter\XeTeXmathchardef
      \csname\expandafter\@gobble\string#2 op\endcsname
      ="\mathchar@type#3 #1 #4\relax
    \gdef#2{\csname\expandafter\@gobble\string#2 op\endcsname\nolimits}%
  \else
```

Test for the `\sqrt` radical, which is probably the only one ever.

```
    \unless\ifnum#4="221A\relax
      \global\XeTeXmathchardef#2="\mathchar@type#3 #1 #4\relax
      \ifnum#4<"FFFF
        \global\XeTeXmathcode#4="\mathchar@type#3 #1 #4\relax
      \fi
    \else
      \gdef#2{\XeTeXradical#1 #4\relax}%
    \fi
  \fi}
```

`\um@set@mathchar`

#1 : Symbol font number
#2 : Symbol, *e.g.*, `\alpha` or `a`
#3 : Type, *e.g.*, `\mathalpha`
#4 : Slot, *e.g.*, `"221E`

Or if it's for a character:

```
\def\um@set@mathchar#1#2#3#4{%
  \global\XeTeXmathcode`#2="\mathchar@type#3 #1 #4\relax}
```

∞

```
\zf@fontspec{}{Cambria Math}
\let\glb@currsize\relax
\DeclareSymbolFont{test}{EU1}{\zf@family}{m}{n}
\DeclareUnicodeMathSymbol{\infinity}{\mathord}{test}{"221E}
$\infinity$
```

`\DeclareUnicodeMathCode`

[For later] or if it's for a character code: (just a wrapper around the primitive)

```
\def\DeclareUnicodeMathCode#1#2#3#4{%
  \global\XeTeXmathcode#1=
    "\mathchar@type#2 \csname sym#3\endcsname #4\relax}
```

A

```
\zf@fontspec{}{Cambria Math}
\let\glb@currsize\relax
\DeclareSymbolFont{test2}{EU1}{\zf@family}{m}{n}
\DeclareUnicodeMathCode{65}{\mathalpha}{test2}{119860}
$A$
```

4.3 User interface to `\DeclareSymbolFont`

`\setmathfont` [#1]: font features
#2 : font name

```
\newcommand\setmathfont[2][]{%
```

Erase any conception LaTeX has of previously defined math symbol fonts; this allows `\DeclareSymbolFont` at any point in the document.

To start with, assume we're defining every math symbol character.

```
\let\glb@currsize\relax
\let\um@char@range\@empty
```

Use `fontspec` to select a font to use. The macro `\S@`⟨*size*⟩ contains the definitions of the sizes used for maths letters, subscripts and subsubscripts in `\tf@size`, `\sf@size`, and `\ssf@size`, respectively.

Perhaps in the future we want options to change the hard-coded `fontspec` maths-related features. At this stage, it isn't clear that this is necessary.

```
\begingroup
  \@um@fontspec@featuretrue
  \csname S@\f@size\endcsname
  \zf@fontspec{
    Script=Math,
    SizeFeatures={
      {Size=\tf@size-},
      {Size=\sf@size-\tf@size,ScriptStyle={}},
      {Size=-\sf@size,ScriptScriptStyle={}}},
    #1}{#2}%
\endgroup
```

Probably want to check there that we're not creating multiple symbol fonts with the same NFSS declaration. On that note, `fontspec` doesn't seem to be keeping track of that, either `:(` (check that out!)

```
\stepcounter{um@fam}%
\DeclareSymbolFont{um@fam\theum@fam}
  {EU1}{\zf@family}{\mddefault}{\updefault}%
```

Now when the list of unicode symbols is input, we want a suitable definition of its internal macro. By default, we want to define every single math char:

```
\ifx\um@char@range\@empty
    \um@text@input{um@fam\theum@fam}%
  \PackageWarning{unicode-math}{Defining the default maths font as `#2'}
    \def\unicode@math@symbol##1##2##3##4{%
      \DeclareUnicodeMathSymbol
        {##2}{##3}{um@fam\theum@fam}{##1}}%
\else
```

If the `Range` font feature has been used, then only a subset of the unicode glyphs are to be defined. See section 5.2 for the code that enables this.

```
  \def\unicode@math@symbol##1##2##3##4{%
    \um@parse@term{##1}{##2}{##3}{%
          \PackageWarning{unicode-math}{Defining \string##2 as mathchar ##1 from font `#2'}
      \DeclareUnicodeMathSymbol
        {##2}{##3}{um@fam\theum@fam}{##1}}}%
\fi
```

And now we input every single maths char. See File II for the source to `unicode-math.tex`.

```
   \input uniadd.tex
   \input unicode-math.tex}
\let\setmathsfont\setmathfont
```

Here's the simplest usage:

$Ax \stackrel{\text{def}}{=} \nabla \times \mathcal{Z}$

```
\setmathfont{Cambria Math}
$Ax \eqdef \nabla \times \scrZ$
```

And an example of the `Range` feature:

$(a, \alpha, □, \aleph, □, □)$
$(a, \alpha, \mathcal{M}, \aleph, □, □)$
$(a, \alpha, \mathscr{M}, \aleph, □, □)$

```
\setmathfont{Cambria Math}
$(a, \alpha, \scrM, \aleph, \scrH, \BbbH)$
\setmathfont[Range={"2133-"2135, \scrH,\BbbH}]{Lucida Sans}
$(a, \alpha, \scrM, \aleph, \scrH, \BbbH)$
\setmathfont[Range={"2133-"2135, \scrH,\BbbH}]{Apple Symbols}
$(a, \alpha, \scrM, \aleph, \scrH, \BbbH)$
```

A less useful (perhaps) example of the `Range` feature:

$$F(s) = □\{f(t)\} = \int_0^\infty e^{-st} f(t)\, dt$$

```
\setmathfont[Colour=000000]{Cambria Math}
\setmathfont[Range={\mathop}, Colour=FF0000]{Cambria Math}
\setmathfont[Range={\mathrel}, Colour=009900]{Cambria Math}
\setmathfont[Range={\mathopen,\mathclose},
          Colour=0000FF]{Cambria Math}
\[ F(s)=\scrL\{f(t)\}=\int_0^\infty e^{-st}f(t)\,\upd t \]
```

4.4 Big operators and radicals

Turns out that XeTeX is clever enough to deal with big operators for us automatically with `\XeTeXmathchardef`. Amazing!

However, the limits aren't properly set. Plain TeX defines `\def\int{\nolimits\intop}`, so there needs to be a transformation from `\int` to `\intop` during the expansion of `\unicode@math@symbol` in the appropriate contexts.

Following is a table of every math operator (`\mathop`) defined in `unicode-maths.tex`, from which a subset need to be flagged for `\nolimits` adjustments. The limits are shown here. (But why does the first line — which shouldn't at all — show up half-complete?)

	$!_0^1$	`\exclam`	EXCLAMATION MARK
″02140	$\sum_0^1$	`\Bbbsum`	DOUBLE-STRUCK N-ARY SUMMATION
″0220F	$\prod_0^1$	`\prod`	PRODUCT OPERATOR
″02210	$\coprod_0^1$	`\coprod`	COPRODUCT OPERATOR
″02211	$\sum_0^1$	`\sum`	SUMMATION OPERATOR
″0222B	$\int_0^1$	`\int`	INTEGRAL OPERATOR
″0222C	$\iint_0^1$	`\iint`	DOUBLE INTEGRAL OPERATOR

″0222D	$∭_0^1$	`\iiint`	TRIPLE INTEGRAL OPERATOR
″0222E	$∮_0^1$	`\oint`	CONTOUR INTEGRAL OPERATOR
″0222F	$∯_0^1$	`\oiint`	DOUBLE CONTOUR INTEGRAL OPERATOR
″02230	$∰_0^1$	`\oiiint`	TRIPLE CONTOUR INTEGRAL OPERATOR
″02231	$∱_0^1$	`\intclockwise`	CLOCKWISE INTEGRAL
″02232	$∲_0^1$	`\varointclockwise`	CONTOUR INTEGRAL, CLOCKWISE
″02233	$∳_0^1$	`\ointctrclockwise`	CONTOUR INTEGRAL, ANTICLOCKWISE
″022C0	$\underset{0}{\overset{1}{⋀}}$	`\bigwedge`	LOGICAL OR OPERATOR
″022C1	$\underset{0}{\overset{1}{⋁}}$	`\bigvee`	LOGICAL AND OPERATOR
″022C2	$\underset{0}{\overset{1}{⋂}}$	`\bigcap`	INTERSECTION OPERATOR
″022C3	$\underset{0}{\overset{1}{⋃}}$	`\bigcup`	UNION OPERATOR
″027D5	$\underset{0}{\overset{1}{⟕}}$	`\leftouterjoin`	LEFT OUTER JOIN
″027D6	$\underset{0}{\overset{1}{⟖}}$	`\rightouterjoin`	RIGHT OUTER JOIN
″027D7	$\underset{0}{\overset{1}{⟗}}$	`\fullouterjoin`	FULL OUTER JOIN
″027D8	$\underset{0}{\overset{1}{⟘}}$	`\bigbot`	LARGE UP TACK
″027D9	$\underset{0}{\overset{1}{⟙}}$	`\bigtop`	LARGE DOWN TACK
″029F8	$\underset{0}{\overset{1}{⧸}}$	`\xsol`	BIG SOLIDUS
″029F9	$\underset{0}{\overset{1}{⧹}}$	`\xbsol`	BIG REVERSE SOLIDUS
″02A00	$\underset{0}{\overset{1}{⨀}}$	`\bigodot`	N-ARY CIRCLED DOT OPERATOR
″02A01	$\underset{0}{\overset{1}{⨁}}$	`\bigoplus`	N-ARY CIRCLED PLUS OPERATOR
″02A02	$\underset{0}{\overset{1}{⨂}}$	`\bigotimes`	N-ARY CIRCLED TIMES OPERATOR
″02A03	$\underset{0}{\overset{1}{⨃}}$	`\bigcupdot`	N-ARY UNION OPERATOR WITH DOT
″02A04	$\underset{0}{\overset{1}{⨄}}$	`\biguplus`	N-ARY UNION OPERATOR WITH PLUS
″02A05	$\underset{0}{\overset{1}{⨅}}$	`\bigsqcap`	N-ARY SQUARE INTERSECTION OPERATOR
″02A06	$\underset{0}{\overset{1}{⨆}}$	`\bigsqcup`	N-ARY SQUARE UNION OPERATOR
″02A07	$\underset{0}{\overset{1}{⨇}}$	`\conjquant`	TWO LOGICAL AND OPERATOR
″02A08	$\underset{0}{\overset{1}{⨈}}$	`\disjquant`	TWO LOGICAL OR OPERATOR
″02A09	$\underset{0}{\overset{1}{⨉}}$	`\bigtimes`	N-ARY TIMES OPERATOR
″02A0B	$⨋_0^1$	`\sumint`	SUMMATION WITH INTEGRAL
″02A0C	$⨌_0^1$	`\iiiint`	QUADRUPLE INTEGRAL OPERATOR

″02A0D	$⨍_0^1$	`\intbar`	FINITE PART INTEGRAL
″02A0E	$⨎_0^1$	`\intBar`	INTEGRAL WITH DOUBLE STROKE
″02A0F	$⨏_0^1$	`\fint`	INTEGRAL AVERAGE WITH SLASH
″02A10	$⨐_0^1$	`\cirfnint`	CIRCULATION FUNCTION
″02A11	$⨑_0^1$	`\awint`	ANTICLOCKWISE INTEGRATION
″02A12	$⨒_0^1$	`\rppolint`	LINE INTEGRATION WITH RECTANGULAR PATH AROUND POLE
″02A13	$⨓_0^1$	`\scpolint`	LINE INTEGRATION WITH SEMICIRCULAR PATH AROUND POLE
″02A14	$⨔_0^1$	`\npolint`	LINE INTEGRATION NOT INCLUDING THE POLE
″02A15	$⨕_0^1$	`\pointint`	INTEGRAL AROUND A POINT OPERATOR
″02A16	$⨖_0^1$	`\sqint`	QUATERNION INTEGRAL OPERATOR
″02A17	$⨗_0^1$	`\intlarhk`	INTEGRAL WITH LEFTWARDS ARROW WITH HOOK
″02A18	$⨘_0^1$	`\intx`	INTEGRAL WITH TIMES SIGN
″02A19	$⨙_0^1$	`\intcap`	INTEGRAL WITH INTERSECTION
″02A1A	$⨚_0^1$	`\intcup`	INTEGRAL WITH UNION
″02A1B	$⨛_0^1$	`\upint`	INTEGRAL WITH OVERBAR
″02A1C	$⨜_0^1$	`\lowint`	INTEGRAL WITH UNDERBAR
″02A1D	$\underset{0}{\overset{1}{⨝}}$	`\Join`	JOIN
″02A1E	$\underset{0}{\overset{1}{⨞}}$	`\bigtriangleleft`	LARGE LEFT TRIANGLE OPERATOR
″02A1F	$\underset{0}{\overset{1}{⨟}}$	`\zcmp`	Z NOTATION SCHEMA COMPOSITION
″02A20	$\underset{0}{\overset{1}{⨠}}$	`\zpipe`	Z NOTATION SCHEMA PIPING
″02A21	$\underset{0}{\overset{1}{⨡}}$	`\zproject`	Z NOTATION SCHEMA PROJECTION
″02AFC	$\underset{0}{\overset{1}{⫼}}$	`\biginterleave`	LARGE TRIPLE VERTICAL BAR OPERATOR
″02AFF	$\underset{0}{\overset{1}{⫿}}$	`\bigtalloblong`	N-ARY WHITE VERTICAL BAR

`\um@nolimits` This macro is a commalist containing those maths operators that require a `\nolimits` suffix. This list is used when processing `unicode-math.tex` to define such commands automatically. I've chosen essentially just the operators that look like integrals; hopefully a better mathematician can help me out here.

```
\def\um@nolimits{%
  \int,\iint,\iiint,\iiiint,%
  \oint,\oiint,\oiiint,%
  \intclockwise,\varointclockwise,\ointctrclockwise,%
  \sumint,\intbar,\intBar,\fint,%
  \cirfnint,\awint,\rppolint,\scpolint,\npolint,\pointint,\sqint,%
  \intlarhk,\intx,\intcap,\intcup,\upint,\lowint}
```

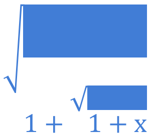
`\addnolimits` This macro appends material to the macro containing the list of operators that don't take limits. Items must be removed manually, at this stage; I'm working on a macro for this too, but it's a bit harder!

```
\newcommand\addnolimits[1]{\g@addto@macro\um@nolimits{,#1}}
```

$$\int_0^1 \sum_0^N \left(\frac{\left(\sum_{i=n}^N \left(\int_0^1 \left(a\times b\right)\right)\right)}{A^{B^C}_{D_E}}\right)$$

```
\setmathfont{Cambria Math}
\[ \int_0^1 \sum_0^N \left(\frac{%
   \left(\sum^N_{i=n}\left(\int^1_0
   \left(a\times b\right)
   \right)\right)}{A^{B^C}_{D_E}}\right) \]
```

The radical for square root is organised in \um@set@mathsymbol on page 6. I think it's the only radical ever, so a more general scheme isn't really necessary. But what about right-to-left square roots?



```
\setmathfont{Cambria Math}
\[ \sqrt{1+\sqrt{1+x}} \]
```

4.5 Delimiters

[TODO]

$$\left(\frac{\left(\sum_{i=n}^N \left(\int_0^1 \left(a\times b\right)\right)\right)}{A^{B^C}_{D_E}}\right)$$

```
\setmathfont{Cambria Math}
\[ \left(\frac{%
   \left(\sum^N_{i=n}\left(\int^1_0
   \left(a\times b\right)
   \right)\right)}{A^{B^C}_{D_E}}\right) \]
```

4.6 Maths accents

[TODO]

4.7 Setting up the ascii ranges

We want it to be convenient for users to actually type in maths. The ASCII Latin characters should be used for italic maths, and the text Greek characters should be used for upright/italic (depending on preference) Greek, if desired.

\um@text@input And here're the text input to maths output mappings, wrapped up in a macro.

```
\newcommand\um@text@input[1]{%
```

Numbers, zero to nine:

```
  \um@FOR @tempcnta = [0:9] {%
    \um@mathcode@offset{#1}{48}{48}%
  }%
```

Latin alphabet, uppercase and lowercase respectively:

```
  \um@FOR @tempcnta = [0:25] {%
    \um@mathcode@offset{#1}{65}{119860}%
    \um@mathcode@offset{#1}{97}{119886}%
  }%
```

Filling a hole for 'h', which maps to U+210E: PLANCK CONSTANT instead of the expected U+1D455: MATHEMATICAL ITALIC SMALL H (which is not assigned):

```
  \DeclareUnicodeMathCode{104}{\mathalpha}{#1}{8462}%
```

Greek alphabet, uppercase (note the hole after U+03A1: GREEK CAPITAL LETTER RHO):

```
  \um@FOR @tempcnta = [0:23] {%
    \DeclareUnicodeMathCode
      {\ifnum\@tempcnta>16
         \numexpr\the\@tempcnta+913\relax
       \else
         \numexpr\the\@tempcnta+913+1\relax
       \fi}
      {\mathalpha}{#1}
      {\numexpr\the\@tempcnta+120546\relax}%
```

And Greek lowercase:

```
    \um@mathcode@offset{#1}{945}{120572}%
  }%
}
```

\um@mathcode@offset

This is a wrapper macro to save space:

```
\newcommand\um@mathcode@offset[3]{%
  \DeclareUnicodeMathCode
    {\numexpr\the\@tempcnta+#2\relax}
    {\mathalpha}{#1}
    {\numexpr\the\@tempcnta+#3\relax}%
}
```

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ
αβγδεζηθικλμνξοπρστυφχψω

```
\setmathsfont{Cambria Math}
$ABCDEFGHIJKLMNOPQRSTUVWXYZ$ \\
$abcdefghijklmnopqrstuvwxyz$ \\
$ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ$ \\
$αβγδεζηθικλμνξοπρστυφχψω$ \\
```

5 fontspec feature hooks

\um@zf@feature

Use the same method as fontspec for feature definition (*i.e.*, using xkeyval) but with a conditional to restrict the scope of these features to unicode-math commands.

```
\newcommand\um@zf@feature[2]{%
  \define@key[zf]{options}{#1}{%
    \if@um@fontspec@feature
      #2
    \else
      \PackageError{fontspec/unicode-math}
        {The `#1' font feature can only be used for maths fonts}
        {The feature you tried to use can only be in commands
          like \protect\setmathsfont}
    \fi}}
```

5.1 OpenType maths font features

These aren't defined in fontspec because they aren't useful in non-maths contexts.

```
\um@zf@feature{ScriptStyle}{%
  \zf@update@ff{+ssty=0}%
}
\um@zf@feature{ScriptScriptStyle}{%
  \zf@update@ff{+ssty=1}%
}
```

5.2 Range processing

```
\um@zf@feature{Range}{\xdef\um@char@range{\zap@space#1 \@empty}}
```

Pretty basic comma separated range processing. Donald Arseneau's selectp package has a cleverer technique.

\um@parse@term

#1 : unicode character slot
#2 : control sequence (character macro)
#3 : control sequence (math type)
#4 : code to execute

This macro expands to #4 (Unless I've got my terminology twisted again.) if any of its arguments are contained in the commalist \um@char@range. This list can contain either character ranges (for checking with #1) or control sequences. These latter can either be the command name of a specific character, *or* the math type of one (*e.g.*, \mathbin).

Character ranges are passed to \um@parse@range, which accepts input in the form shown in table 2.

Input	Range
x	$r = x$
x-	$r \geq x$
-x	$r \leq x$
x-y	$x \leq r \leq y$

Table 2: Ranges accepted by \um@parse@range

Start by iterating over the commalist, ignoring empties, and initialising the scratch conditional:

```
\newcommand\um@parse@term[4]{%
  \@for\@ii:=\um@char@range\do{%
    \unless\ifx\@ii\@empty
      \@tempswafalse
```

If `\if\relax\noexpand##` is true if `##` is a control sequence; then match to either the character macro or the math type:

```
      \expandafter\if\expandafter\relax\expandafter\noexpand\@ii
        \expandafter\ifx\@ii#2
          \@tempswatrue
        \else
          \expandafter\ifx\@ii#3
            \@tempswatrue
          \fi
        \fi
```

Otherwise, we have a number range, which is passed to another macro:

```
      \else
        \expandafter\um@parse@range\@ii-\@marker-\@nil#1\@nil
      \fi
```

If we have a match, execute the code!

```
      \if@tempswa
        #4
      \fi
    \fi}}
```

‘1’ or ‘\a’ or ‘\b’ is included ‘1’ or ‘\b’ or ‘\c’ is included ‘3’ or ‘\a’ or ‘\b’ is included ‘3’ or ‘\a’ or ‘\b’ is included

```
\def\um@char@range{\a,2-4,\c}
\um@parse@term{1}{\a}{\b}
   {`1' or `\string\a' or `\string\b' is included}
\um@parse@term{1}{\b}{\c}
   {`1' or `\string\b' or `\string\c' is included}
\um@parse@term{3}{\a}{\b}
   {`3' or `\string\a' or `\string\b' is included}
```

`\um@parse@range` Weird syntax. As shown previously in table 2, this macro can be passed four different input types via `\um@parse@term`.

```
\def\um@parse@range#1-#2-#3\@nil#4\@nil{%
  \def\@tempa{#1}%
  \def\@tempb{#2}%
```

Range	r = x
C-list input	`\@ii=X`
Macro input	`\um@parse@range X-\@marker-\@nil#1\@nil`
Arguments	`#1-#2-#3` = `X-\@marker-{}`

```
  \ifx\@marker\@tempb\relax
    \ifnum#4=#1\relax
      \@tempswatrue
    \fi
  \else
```

Range	$r \geq x$
C-list input	`\@ii=X-`
Macro input	`\um@parse@range X--\@marker-\@nil#1\@nil`
Arguments	`#1-#2-#3` = `X-{}-\@marker-`

```
  \ifx\@empty\@tempb
   \ifnum#4>\numexpr#1-1\relax
    \@tempswatrue
   \fi
  \else
```

Range	$r \leq x$
C-list input	`\@ii=-Y`
Macro input	`\um@parse@range -Y-\@marker-\@nil#1\@nil`
Arguments	`#1-#2-#3` = `{}-Y-\@marker-`

```
   \ifx\@empty\@tempa
    \ifnum#4<\numexpr#2+1\relax
     \@tempswatrue
    \fi
```

Range	$x \leq r \leq y$
C-list input	`\@ii=X-Y`
Macro input	`\um@parse@range X-Y-\@marker-\@nil#1\@nil`
Arguments	`#1-#2-#3` = `X-Y-\@marker-`

```
   \else
    \ifnum#4>\numexpr#1-1\relax
     \ifnum#4<\numexpr#2+1\relax
      \@tempswatrue
     \fi
    \fi
   \fi
  \fi
 \fi}
```

File II

STIX table data extraction

The source for the TeX names for the very large number of mathematical glyphs are provided via Barbara Beeton's table file for the STIX project (`ams.org/STIX`). A version is located at `http://www.ams.org/STIX/bnb/stix-tbl.asc` but it's not currently up to date.

A single file is produced containing all (more than 3298) symbols. Future optimisations might include generating various (possibly overlapping) subsets so not all definitions must be read just to redefine a small range of symbols..

```
#!/bin/sh

cat stix-tbl.asc |
awk '
```

```
 BEGIN {OFS="|"}
 {if (usv != substr($0,2,5) )
   {if (substr($0,2,1) != " ")
    {usv = substr($0,2,5);
     texname = substr($0,84,25);
     class = substr($0,57,1);
     description = tolower(substr($0,233,350));
     {if (texname ~ /[\\]/)
       print usv, texname, class, description;}}}}' - |
awk -F"|" '
  ($3 != " ") {
    print "\\unicode@math@symbol{" "\"" $1 "}{" $2 "}{" $3 "}{" $4 "}";
 }' - |
sed -e ' s/{N}/{\\mathord}/   ' \
    -e ' s/{F}/{\\mathord}/ ' \
    -e ' s/{A}/{\\mathalpha}/ ' \
    -e ' s/{P}/{\\mathpunct}/ ' \
    -e ' s/{B}/{\\mathbin}/   ' \
    -e ' s/{R}/{\\mathrel}/   ' \
    -e ' s/{L}/{\\mathop}/    ' \
    -e ' s/{O}/{\\mathopen}/  ' \
    -e ' s/{C}/{\\mathclose}/ ' > unicode-math.tex
```

A Documenting maths support in the NFSS

A.1 Overview

In the following, ⟨*NFSS decl.*⟩ stands for something like `{T1}{lmr}{m}{n}`.

Maths symbol fonts Fonts for symbols: $\propto$, $\leq$, $\rightarrow$

`\DeclareSymbolFont{`⟨*name*⟩`}`⟨*NFSS decl.*⟩
Declares a named maths font such as `operators` from which symbols are defined with `\DeclareMathSymbol`.

Maths alphabet fonts Fonts for $ABC - xyz$, $\mathfrak{ABC} - \mathcal{XYZ}$, etc.

`\DeclareMathAlphabet{`⟨*cmd*⟩`}`⟨*NFSS decl.*⟩

For commands such as `\mathbf`, accessed through maths mode that are unaffected by the current text font, and which are used for alphabetic symbols in the ASCII range.

`\DeclareSymbolFontAlphabet{`⟨*cmd*⟩`}{`⟨*name*⟩`}`

Alternative (and optimisation) for `\DeclareMathAlphabet` if a single font is being used for both alphabetic characters (as above) and symbols.

Maths 'versions' Different maths weights can be defined with the following, switched in text with the `\mathversion{`⟨*maths version*⟩`}` command.

`\SetSymbolFont{`⟨*name*⟩`}{`⟨*maths version*⟩`}`⟨*NFSS decl.*⟩
`\SetMathAlphabet{`⟨*cmd*⟩`}{`⟨*maths version*⟩`}`⟨*NFSS decl.*⟩

Maths symbols Symbol definitions in maths for both characters (=) and macros (`\eqdef`): `\DeclareMathSymbol{`⟨*symbol*⟩`}{`⟨*type*⟩`}{`⟨*named font*⟩`}{`⟨*slot*⟩`}` This is the macro that actually defines which font each symbol comes from and how they behave.

Delimiters and radicals use wrappers around TeX's `\delimiter`/`\radical` primitives, which are re-designed in XeTeX. The syntax used in LaTeX's NFSS is therefore not so relevant here.

Delimiters A special class of maths symbol which enlarge themselves in certain contexts.

`\DeclareMathDelimiter{`⟨*symbol*⟩`}{`⟨*type*⟩`}`⟨*sym. font*⟩⟨*slot*⟩⟨*sym. font*⟩⟨*slot*⟩

Radicals Similar to delimiters (`\DeclareMathRadical` takes the same syntax) but behave 'weirdly'. `\sqrt` might very well be the only one.

In those cases, glyph slots in *two* symbol fonts are required; one for the small ('regular') case, the other for situations when the glyph is larger.

Accents are not included yet.

A.2 Detailed code investigation

This section contains an abridged and documented version of (bits and pieces of) LaTeX's NFSS. Changes are mostly cosmetic and omission of irrelevant things.

A.3 Maths symbols

`\DeclareMathSymbol`

`#1` : Symbol, e.g., `\alpha` or 'a'
`#2` : Type, e.g., `\mathalpha`
`#3` : Math font name, e.g., `operators`
`#4` : Slot, e.g., `F1`

```
\def\DeclareMathSymbol#1#2#3#4{%
```

First ensure the math font (e.g., `operators`) exists:

```
  \expandafter\in@\csname sym#3\expandafter\endcsname
    \expandafter{\group@list}%
  \ifin@
```

Convert the slot number to two hex digits stored in `\count\z@` and `\count\tw@`, respectively:

```
    \begingroup
      \count\z@=#4\relax
      \count\tw@\count\z@
      \divide\count\z@\sixt@@n
      \count@\count\z@
      \multiply\count@\sixt@@n
      \advance\count\tw@-\count@
```

The symbol to be defined can be either a command (`\alpha`) or a character (`a`). Branch for the former:

```
      \if\relax\noexpand#1% is command?
        \edef\reserved@a{\noexpand\in@{\string\mathchar}{\meaning#1}}%
        \reserved@a
```

If the symbol command definition contains `\mathchar`, then we can provide the info that a previous symbol definition is being overwritten:

```
        \ifin@
          \expandafter\set@mathsymbol
             \csname sym#3\endcsname#1#2%
             {\hexnumber@{\count\z@}\hexnumber@{\count\tw@}}%
          \@font@info{Redeclaring math symbol \string#1}%
```

Otherwise, throw an error if the command name is already taken by a non-symbol definition:

```
        \else
            \expandafter\ifx
            \csname\expandafter\@gobble\string#1\endcsname
            \relax
            \expandafter\set@mathsymbol
               \csname sym#3\endcsname#1#2%
               {\hexnumber@{\count\z@}\hexnumber@{\count\tw@}}%
          \else
            \@latex@error{Command `\string#1' already defined}\@eha
          \fi
        \fi
```

And if the symbol input is a character:

```
      \else
        \expandafter\set@mathchar
          \csname sym#3\endcsname#1#2
          {\hexnumber@{\count\z@}\hexnumber@{\count\tw@}}%
      \fi
    \endgroup
```

Everything previous was skipped if the maths font doesn't exist in the first place:

```
  \else
    \@latex@error{Symbol font `#3' is not defined}\@eha
  \fi}
```

The final macros that actually define the maths symbol with TEX primitives. If the symbol definition is for a macro:

```
\def\set@mathsymbol#1#2#3#4{%
  \global\mathchardef#2"\mathchar@type#3\hexnumber@#1#4\relax}
```

Or if it's for a character:

```
\def\set@mathchar#1#2#3#4{%
  \global\mathcode‘#2="\mathchar@type#3\hexnumber@#1#4\relax}
```

Summary For symbols, something like:

```
\def\DeclareMathSymbol#1#2#3#4{%
  \global\mathchardef#1"\mathchar@type#2
    \expandafter\hexnumber@\csname sym#2\endcsname
    {\hexnumber@{\count\z@}\hexnumber@{\count\tw@}}}
```

For characters, something like:

```
\def\DeclareMathSymbol#1#2#3#4{%
  \global\mathcode`#1"\mathchar@type#2
    \expandafter\hexnumber@\csname sym#2\endcsname
    {\hexnumber@{\count\z@}\hexnumber@{\count\tw@}}}
```

A.4 Delimiters

The code here is slightly better documented originally than the other maths commands.

\DeclareMathDelimiter

```
\def\DeclareMathDelimiter#1{%
  \if\relax\noexpand#1%
    \expandafter\@DeclareMathDelimiter
  \else
    \expandafter\@xxDeclareMathDelimiter
  \fi
  #1}
\@onlypreamble\DeclareMathDelimiter
```

\@xxDeclareMathDelimiter This macro checks if the second arg is a "math type" such as \mathopen. The undocumented original code didn't use math types when the delimiter was a single letter. For this reason the coding is a bit strange as it tries to support the undocumented syntax for compatibility reasons.

```
\def\@xxDeclareMathDelimiter#1#2#3#4{%
```

7 is the default value returned in the case that \mathchar@type is passed something unexpected, like a math symbol font name. We locally move \mathalpha out of the way so if you use that the right branch is taken. This will still fail if an explicit number 7 is used!

```
   \begingroup
    \let\mathalpha\mathord
    \ifnum7=\mathchar@type{#2}%
      \endgroup
```

If this branch is taken we have old syntax (5 arguments).

```
      \expandafter\@firstofone
    \else
```

If this branch is taken `\mathchar@type` is different from 7 so we assume new syntax. In this case we also use the arguments to set up the letter as a math symbol for the case where it is not used as a delimiter.

```
      \endgroup
      \DeclareMathSymbol#1{#2}{#3}{#4}%
```

Then we arrange that `\@xDeclareMathDelimiter` only gets `#1`, `#3`, `#4` … as it does not expect a math type as argument.

```
      \expandafter\@firstoftwo
    \fi
    {\@xDeclareMathDelimiter#1}{#2}{#3}{#4}}
\@onlypreamble\@xxDeclareMathDelimiter
```

`\@DeclareMathDelimiter`

```
\def\@DeclareMathDelimiter#1#2#3#4#5#6{%
  \expandafter\in@\csname sym#3\expandafter\endcsname
     \expandafter{\group@list}%
  \ifin@
    \expandafter\in@\csname sym#5\expandafter\endcsname
       \expandafter{\group@list}%
    \ifin@
      \begingroup
        \count\z@=#4\relax
        \count\tw@\count\z@
        \divide\count\z@\sixt@@n
        \count@\count\z@
        \multiply\count@\sixt@@n
        \advance\count\tw@-\count@
        \edef\reserved@c{\hexnumber@{\count\z@}\hexnumber@{\count\tw@}}%
      %
        \count\z@=#6\relax
        \count\tw@\count\z@
        \divide\count\z@\sixt@@n
        \count@\count\z@
        \multiply\count@\sixt@@n
        \advance\count\tw@-\count@
        \edef\reserved@d{\hexnumber@{\count\z@}\hexnumber@{\count\tw@}}%
      %
        \edef\reserved@a{\noexpand\in@{\string\delimiter}{\meaning#1}}%
        \reserved@a
        \ifin@
          \expandafter\set@mathdelimiter
             \csname sym#3\expandafter\endcsname
             \csname sym#5\endcsname#1#2%
             \reserved@c\reserved@d
          \@font@info{Redeclaring math delimiter \string#1}%
        \else
            \expandafter\ifx
            \csname\expandafter\@gobble\string#1\endcsname
            \relax
            \expandafter\set@mathdelimiter
```

```
              \csname sym#3\expandafter\endcsname
              \csname sym#5\endcsname#1#2%
              \reserved@c\reserved@d
          \else
            \@latex@error{Command `\string#1' already defined}\@eha
          \fi
        \fi
      \endgroup
    \else
      \@latex@error{Symbol font `#5' is not defined}\@eha
    \fi
  \else
    \@latex@error{Symbol font `#3' is not defined}\@eha
  \fi
}
\@onlypreamble\@DeclareMathDelimiter
```

\@xDeclareMathDelimiter

```
\def\@xDeclareMathDelimiter#1#2#3#4#5{%
  \expandafter\in@\csname sym#2\expandafter\endcsname
     \expandafter{\group@list}%
  \ifin@
    \expandafter\in@\csname sym#4\expandafter\endcsname
       \expandafter{\group@list}%
    \ifin@
      \begingroup
        \count\z@=#3\relax
        \count\tw@\count\z@
        \divide\count\z@\sixt@@n
        \count@\count\z@
        \multiply\count@\sixt@@n
        \advance\count\tw@-\count@
        \edef\reserved@c{\hexnumber@{\count\z@}\hexnumber@{\count\tw@}}%
      %
        \count\z@=#5\relax
        \count\tw@\count\z@
        \divide\count\z@\sixt@@n
        \count@\count\z@
        \multiply\count@\sixt@@n
        \advance\count\tw@-\count@
        \edef\reserved@d{\hexnumber@{\count\z@}\hexnumber@{\count\tw@}}%
        \expandafter\set@@mathdelimiter
           \csname sym#2\expandafter\endcsname\csname sym#4\endcsname#1%
           \reserved@c\reserved@d
      \endgroup
    \else
      \@latex@error{Symbol font `#4' is not defined}\@eha
    \fi
  \else
    \@latex@error{Symbol font `#2' is not defined}\@eha
  \fi
}
\@onlypreamble\@xDeclareMathDelimiter
```

\set@mathdelimiter

We have to end the definition of a math delimiter like `\lfloor` with a space and not with `\relax` as we did before, because otherwise contructs involving `\abovewithdelims` will prematurely end (pr/1329)

```
\def\set@mathdelimiter#1#2#3#4#5#6{%
  \xdef#3{\delimiter"\mathchar@type#4\hexnumber@#1#5%
                                     \hexnumber@#2#6 }}
\@onlypreamble\set@mathdelimiter
```

\set@@mathdelimiter

```
\def\set@@mathdelimiter#1#2#3#4#5{%
  \global\delcode`#3="\hexnumber@#1#4\hexnumber@#2#5\relax}
\@onlypreamble\set@@mathdelimiter
```

A.5 Symbol fonts

\DeclareSymbolFont

`#1` : font name, *e.g.*, `letters`
`#2` : font encoding, *e.g.*, `OT1`
`#3` : font family, *e.g.*, `cmr`
`#4` : font series, *e.g.*, `m`
`#5` : font shape, *e.g.*, `n`

```
\def\DeclareSymbolFont#1#2#3#4#5{%
```

First check that the font encoding is defined.

```
 \@tempswafalse
 \edef\reserved@b{#2}%
 \def\cdp@elt##1##2##3##4{\def\reserved@c{##1}%
      \ifx\reserved@b\reserved@c \@tempswatrue\fi}%
 \cdp@list
```

So far so good. Now branch depending if this symbol font has been declared yet or not. If not, the symbol font is defined as the macro `\sym#1`; *i.e.*, for the `letters` symbol font, the associated command name is `\symletters`. (Funny it's not `\sym@#1`.)

```
 \if@tempswa
   \@ifundefined{sym#1}{%
      \expandafter\new@mathgroup\csname sym#1\endcsname
     \expandafter\new@symbolfont\csname sym#1\endcsname{#2}{#3}{#4}{#5}%
   }%
```

If the symbol font has been already declared:

```
     {\@font@info{Redeclaring symbol font `#1'}%
```

[Update the group list.]

```
      \def\group@elt##1##2{%
           \noexpand\group@elt\noexpand##1%
           \expandafter\ifx\csname sym#1\endcsname##1%
             \expandafter\noexpand\csname#2/#3/#4/#5\endcsname
           \else
```

```
               \noexpand##2%
           \fi}%
      \xdef\group@list{\group@list}%
```

[Update the version list.]

```
      \def\version@elt##1{%
          \expandafter
          \SetSymbolFont@\expandafter##1\csname#2/#3/#4/#5\expandafter
              \endcsname \csname sym#1\endcsname
          }%
      \version@list
     }%
```

If the font encoding wasn't defined, all of the above was skipped.

```
  \else
    \@latex@error{Encoding scheme  `#2' unknown}\@eha
  \fi}
```

\new@symbolfont

#1 : internal symbol font name, *e.g.*, \symletters
#2 : font encoding, *e.g.*, OT1
#3 : font family, *e.g.*, cmr
#4 : font series, *e.g.*, m
#5 : font shape, *e.g.*, n

```
\def\new@symbolfont#1#2#3#4#5{%
```

Update the group list:

```
    \toks@\expandafter{\group@list}%
    \edef\group@list{\the\toks@\noexpand\group@elt\noexpand#1%
                     \expandafter\noexpand\csname#2/#3/#4/#5\endcsname}%
    \def\version@elt##1{\toks@\expandafter{##1}%
                   \edef##1{\the\toks@\noexpand\getanddefine@fonts
                   #1\expandafter\noexpand\csname#2/#3/#4/#5\endcsname}%
                  \global\advance\csname c@\expandafter
                                 \@gobble\string##1\endcsname\@ne
                 }%
    \version@list}
```

\SetSymbolFont

#1 : math font version, *e.g.*, normal
#2 : font name, *e.g.*, letters
#3 : font encoding, *e.g.*, OT1
#4 : font family, *e.g.*, cmr
#5 : font series, *e.g.*, m
#6 : font shape, *e.g.*, n

```
\def\SetSymbolFont#1#2#3#4#5#6{%
 \@tempswafalse
 \edef\reserved@b{#3}%
 \def\cdp@elt##1##2##3##4{\def\reserved@c{##1}%
      \ifx\reserved@b\reserved@c \@tempswatrue\fi}%
 \cdp@list
 \if@tempswa
```

```
  \expandafter\SetSymbolFont@
    \csname mv@#2\expandafter\endcsname\csname#3/#4/#5/#6\expandafter
    \endcsname \csname sym#1\endcsname
 \else
  \@latex@error{Encoding scheme  `#3' unknown}\@eha
 \fi
}
```

`\SetSymbolFont@` #1 : internal math font version, *e.g.*, `\mv@normal`
#2 : NFSS font, *e.g.*, `\OT1/cmr/m/n`
#3 : internal symbol name, *e.g.*, `\symletters`

```
\def\SetSymbolFont@#1#2#3{%
```

If the maths version has been defined:

```
  \expandafter\in@\expandafter#1\expandafter{\version@list}%
  \ifin@
```

If the symbol font has been defined:

```
    \expandafter\in@\expandafter#3\expandafter{\group@list}%
    \ifin@
      \begingroup
        \expandafter\get@cdp\string#2\@nil\reserved@a
        \toks@{}%
        \def\install@mathalphabet##1##2{%
             \addto@hook\toks@{\install@mathalphabet##1{##2}}%
            }%
        \def\getanddefine@fonts##1##2{%
          \ifnum##1=#3%
             \addto@hook\toks@{\getanddefine@fonts#3#2}%
             \expandafter\get@cdp\string##2\@nil\reserved@b
             \ifx\reserved@a\reserved@b\else
                \@font@warning{Encoding `\reserved@b' has changed
                    to `\reserved@a' for symbol font\MessageBreak
                   `\expandafter\@gobblefour\string#3' in the
                    math version `\expandafter
                    \@gobblefour\string#1'}%
             \fi
             \@font@info{%
                Overwriting symbol font
                `\expandafter\@gobblefour\string#3' in
                 version `\expandafter
                \@gobblefour\string#1'\MessageBreak
                \@spaces \expandafter\@gobble\string##2 -->
                         \expandafter\@gobble\string#2}%
          \else
             \addto@hook\toks@{\getanddefine@fonts##1##2}%
          \fi}%
         #1%
         \xdef#1{\the\toks@}%
      \endgroup
```

If the symbol font wasn't defined, all of the above was skipped:

```
    \else
       \@latex@error{Symbol font `\expandafter\@gobblefour\string#3'
                  not defined}\@eha
    \fi
```

If the maths version wasn't defined, all of the above was skipped:

```
  \else
    \@latex@error{Math version `\expandafter\@gobblefour\string#1'
       is not
       defined}{You probably mispelled the name of the math
       version.^^JOr you have to specify an additional package.}%
  \fi}
```