Experimental unicode mathematical typesetting: The unicode-math package

Will Robertson

2009/10/21 v0.4

Abstract

Warning! This package is experimental and subject to change without regard for backwards compatibility. Performance issues may be encountered until algorithms are refined.

This package is intended to be a complete implementation of unicode maths for LaTeX using the XeTeX (and later, LuaTeX) typesetting engines. With this package, changing maths fonts will be as easy as changing text fonts — not that there are many unicode maths fonts yet.

Maths input is simplified with unicode since literal glyphs may be entered instead of control sequences.

Contents

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 XƎTEX, although it is conjectured that some effect could be spent to create a cross-format package that would also work with LuaTEX.

Users who desire to specify maths alphabets only (Greek and Latin letters) may wish to use Andrew Moschou's mathspec package instead.

2 Acknowledgements

Many thanks to Microsoft for developing OpenType math as part of Office 2007; Jonathan Kew for implementing unicode math support in XƎTEX; Barbara Beeton for her prodigious effort compiling the definitive list of unicode math glyphs and their LATEX names (inventing them where necessary), and also for her thoughtful replies to my sometimes incessant questions. Ross Moore and Chris Rowley have provided moral and technical support from the very early days with great insight into the issues we face trying to extend and use TEX in the future. Apostolos Syropoulos, Joel Salomon, and Khaled Hosny have been fantastic beta testers.

3 Getting started

Load unicode-math as a regular LATEX package. It should be loaded after any other maths or font-related package in case it needs to overwrite their definitions. Here's an example:

```
\usepackage{amsmath} % if desired
\usepackage{unicode-math}
\setmathfont{Cambria Math}
```

3.1 Package options

Package options may be set when the package as loaded or at any later stage with the `\unimathsetup` command. Therefore, the following two examples are equivalent:

```
\usepackage[math-style=TeX]{unicode-math}
% OR
\usepackage{unicode-math}
\unimathsetup{math-style=TeX}
```

Table 1: Package options.

Option	Description	See…
`math-style`	Style of letters	section §5.1
`bold-style`	Style of bold letters	section §5.2
`sans-style`	Style of sans serif letters	section §5.3
`nabla`	Style of the nabla symbol	section §5.5.1
`partial`	Style of the partial symbol	section §5.5.2
`vargreek-shape`	Style of phi and epsilon	section §5.5.3
`colon`	Behaviour of `\colon`	section §5.5.6
`slash-delimiter`	Glyph to use for 'stretchy' slash	section §5.5.7

Note, however, that some package options affects how maths is initialised and changing an option such as `math-style` will not take effect until a new maths font is set up.

Package options may *also* be used when declaring new maths fonts, passed via options to the `\setmathfont` command. Therefore, the following two examples are equivalent:

```
\unimathsetup{math-style=TeX}
\setmathfont{Cambria Math}
% OR
\setmathfont[math-style=TeX]{Cambria Math}
```

A short list of package options is shown in table 1. See following sections for more information.

4 Unicode maths font setup

In the ideal case, a single unicode font will contain all maths glyphs we need. The file `unicode-math-table.tex` (based on Barbara Beeton's STIX table) provides the mapping between unicode maths glyphs and macro names (all 3298 — or however many — of them!). A single command

`\setmathfont[`⟨*font features*⟩`]{`⟨*font name*⟩`}`

implements 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.

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

Font features specific to unicode-math are shown in table 2. Package options (see table 1) may also be used. Other fontspec features are also valid.

Table 2: Maths font options.

Option	Description	See…
`range`	Style of letters	section §4.1
`script-font`	Font to use for sub- and super-scripts	section §4.2
`script-features`	Font features for sub- and super-scripts	section §4.2
`sscript-font`	Font to use for nested sub- and super-scripts	section §4.2
`sscript-features`	Font features for nested sub- and super-scripts	section §4.2

4.1 Using multiple fonts

There will probably be few cases where a single unicode maths font suffices (simply due to glyph coverage). The upcoming STIX font comes to mind as a possible exception. It will therefore be necessary to delegate specific unicode ranges of glyphs to separate fonts:

`\setmathfont[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}`. You may also use the macro for accessing the glyph, such as `\int`, or whole collection of symbols with the same math type, such as `\mathopen`, or complete math alphabets such as `\mathbb`. (Only numerical slots, however, can be used in ranged declarations.)

4.1.1 Control over maths alphabets

Exact control over maths alphabets can be somewhat involved. Here is the current plan.

- `[range=\mathbb]` to use the font for 'bb' letters only.
- `[range=\mathbfsfit/{greek,Greek}]` for Greek lowercase and uppercase only (with `latin`, `Latin`, `num` as well for Latin lower-/upper-case and numbers).
- `[range=\mathsfit->\mathbfsfit]` to map to different output alphabet(s) (which is rather useless right now but will become less useless in the future).

And now the trick. If a particular math alphabet is not defined in the font, fall back onto the lower-base plane (i.e., upright) glyphs. Therefore, to use an ASCII-encoded fractur font, for example, write

`\setmathfont[range=\mathfrak]{SomeFracturFont}`

and because the math plane fractur glyphs will be missing, unicode-math will know to use the ASCII ones instead. If necessary (but why?) this behaviour can be forced with `[range=\mathfrac->\mathup]`.

4.2 Script and scriptscript fonts/features

Cambria Math uses OpenType font features to activate smaller optical sizes for scriptsize and scriptscriptsize symbols (the B and C, respectively, in A_{B_C}). Other fonts will possibly use entirely separate fonts.

Not yet implemented: Both of these options must be taken into account. I hope this will be mostly automatic from the users' points of view. The `+ssty` feature can be detected and applied automatically, and appropriate optical size information embedded in the fonts will ensure this latter case. Fine tuning should be possible automatically with fontspec options. We might have to wait until MnMath, for example, before we really know.

5 Maths input

XeTeX's unicode support allows maths input through two methods. Like classical TeX, macros such as `\alpha`, `\sum`, `\pm`, `\leq`, and so on, provide verbose access to the entire repertoire of characters defined by unicode. The literal characters themselves may be used instead, for more readable input files.

5.1 Math 'style'

Classically, TeX uses italic lowercase Greek letters and *upright* uppercase Greek letters for variables in mathematics. This is contrary to the ISO standards of using italic forms for both upper- and lowercase. Furthermore, the French (contrary again, *quelle surprise*) have been known to use upright uppercase *Latin* letters as well as upright upper- and lowercase Greek. Finally, it is not unknown to use upright letters for all characters, as seen in the Euler fonts.

The `unicode-math` package accommodates these possibilities with an interface heavily inspired by Walter Schmidt's lucimatx package: a package option `math-style` that takes one of four arguments: `TeX`, `ISO`, `French`, or `upright` (case insensitive).

The philosophy behind the interface to the mathematical alphabet symbols lies in LaTeX's attempt of separating content and formatting. Because input source text may come from a variety of places, the upright and 'mathematical' italic Latin and Greek alphabets are *unified* from the point of view of having a specified meaning in the source text. That is, to get a mathematical 'x', either the ascii ('keyboard') letter `x` may be typed, or the actual unicode character may be used. Similarly for Greek letters. The upright or italic forms are then chosen based on the `math-style` package option.

If glyphs are desired that do not map as per the package option (for example, an upright 'g' is desired but typing `$g$` yields 'g'), *markup* is required to specify this; to follow from the example: `\mathup{g}`. Maths alphabets commands such as `\mathup` are detailed later.

Table 3: Effects of the `math-style` package option.

	Example	
Package option	Latin	Greek
`math-style=ISO`	(a,z,B,X)	$(\alpha,\beta,\mathit{\Gamma},\mathit{\Xi})$
`math-style=TeX`	(a,z,B,X)	$(\alpha,\beta,\Gamma,\Xi)$
`math-style=French`	$(a,z,\mathrm{B},\mathrm{X})$	$(\upalpha,\upbeta,\Gamma,\Xi)$
`math-style=upright`	$(\mathrm{a},\mathrm{z},\mathrm{B},\mathrm{X})$	$(\upalpha,\upbeta,\Gamma,\Xi)$

Alternative interface However, some users may not like this convention of normalising their input. For them, an upright `x` is an upright 'x' and that's that. (This will be the case when obtaining source text from copy/pasting PDF or Microsoft Word documents, for example.) For these users, the `literal` option to `math-style` will effect this behaviour.

The `math-style` options' effects are shown in brief in table 3.

5.2 Bold style

Similar as in the previous section, ISO standards differ somewhat to TeX's conventions (and classical typesetting) for 'boldness' in mathematics. In the past, it has been customary to use bold *upright* letters to denote things like vectors and matrices. For example, $\mathbf{M} = (M_x, M_y, M_z)$. Presumably, this was due to the relatively scarcity of bold italic fonts in the pre-digital typesetting era. It has been suggested that *italic* bold symbols are used nowadays instead.

Bold Greek letters have simply been bold variant glyphs of their regular weight, as in $\boldsymbol{\xi} = (\xi_r, \xi_\varphi, \xi_\theta)$. Confusingly, the syntax in LaTeX has been different for these two examples: `\mathbf` in the former ('**M**'), and `\bm` (or `\boldsymbol`, deprecated) in the latter ('$\boldsymbol{\xi}$').

In `unicode-math`, the `\mathbf` command works directly with both Greek and Latin maths alphabet characters and depending on package option either switches to upright for Latin letters (`bold-style=TeX`) as well or keeps them italic (`bold-style=ISO`).

To match the package options for non-bold characters, for `bold-style=upright` all bold characters are upright, and `bold-style=literal` does not change the upright/italic shape of the letter.

Upright and italic bold mathematical letters input as direct unicode characters are normalised with the same rules. For example, with `bold-style=TeX`, a literal bold italic latin character will be typeset upright.

Note that `bold-style` is independent of `math-style`, although if the former is not specified then sensible defaults are chosen based on the latter.

The `bold-style` options' effects are shown in brief in table 4.

Table 4: Effects of the `bold-style` package option.

	Example	
Package option	Latin	Greek
`bold-style=ISO`	$(\boldsymbol{a}, \boldsymbol{z}, \boldsymbol{B}, \boldsymbol{X})$	$(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\Gamma}, \boldsymbol{\Xi})$
`bold-style=TeX`	$(\mathbf{a}, \mathbf{z}, \mathbf{B}, \mathbf{X})$	$(\boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{\Gamma}, \mathbf{\Xi})$
`bold-style=upright`	$(\mathbf{a}, \mathbf{z}, \mathbf{B}, \mathbf{X})$	$(\boldsymbol{\upalpha}, \boldsymbol{\upbeta}, \mathbf{\Gamma}, \mathbf{\Xi})$

5.3 Sans serif style

Unicode contains upright and italic, medium and bold mathematical alphabet characters. These may be explicitly selected with the `\mathsfup`, `\mathsfit`, `\mathbfsfup`, and `\mathbfsfit` commands discussed in section §5.4.

How should the generic `\mathsf` behave? Unlike bold, sans serif is used much more sparingly in mathematics. I've seen recommendations to typeset tensors in sans serif italic or sans serif italic bold (e.g., examples in the isomath and mattens packages). But LaTeX's `\mathsf` is *upright* sans serif.

Therefore I reluctantly add the package options `[sans-style=upright]` and `[sans-style=italic]` to control the behaviour of `\mathsf`. The `upright` style sets up the command to use the seemingly-useless upright sans serif, including Greek; the `italic` style switches to using italic in both Latin and Greek alphabets. In other words, this option simply changes the meaning of `\mathsf` to either `\mathsfup` or `\mathsfit`, respectively. Please let me know if more granular control is necessary here.

There is also a `[sans-style=literal]` setting, set automatically with `[math-style=literal]`, which retains the uprightness of the input characters used when selecting the sans serif output.

5.3.1 What about bold sans serif?

While you might want your bold upright and your sans serif italic, I don't believe you'd also want your bold sans serif upright (or all vice versa, if that's even conceivable). Therefore, bold sans serif follows from the setting for sans serif; it is completely independent of the setting for bold.

In other words, `\mathbfsf` is `\mathbfsfup` or `\mathbfsfit` based on `[sans-style=upright]` or `[sans-style=italic]`, respectively. And `[sans-style=literal]` causes `\mathbfsf` to retain the same italic or upright shape as the input, and turns it bold sans serif.

Note well! There is no medium-weight sans serif Greek alphabet in unicode; therefore, `\mathsf{\alpha}` does not make sense (simply produces 'α') while `\mathbfsf{\alpha}` gives 'α'.

Table 5: Mathematical alphabets defined in unicode. Black dots indicate an alphabet exists in the font specified; grey dots indicate shapes that should always be taken from the upright font even in the italic style. See main text for description of `\mathbbit`.

Font				Alphabet		
Style	Shape	Series	Switch	Latin	Greek	Numerals
Serif	Upright	Normal	`\mathup`	●	●	●
		Bold	`\mathbfup`	●	●	●
	Italic	Normal	`\mathit`	●	●	∙
		Bold	`\mathbfit`	●	●	∙
Sans serif	Upright	Normal	`\mathsfup`	●		●
	Italic	Normal	`\mathsfit`	●		∙
	Upright	Bold	`\mathsfbfup`	●	●	●
	Italic	Bold	`\mathsfbfit`	●	●	∙
Typewriter	Upright	Normal	`\mathtt`	●		●
Double-struck	Upright	Normal	`\mathbb`	●		●
	Italic	Normal	`\mathbbit`	●		
Script	Upright	Normal	`\mathscr`	●		
		Bold	`\matbfscr`	●		
Fraktur	Upright	Normal	`\mathfrak`	●		
		Bold	`\mathbffrac`	●		

5.4 All (the rest) of the mathematical alphabets

Unicode contains separate codepoints for most if not all variations of alphabet shape one may wish to use in mathematical notation. The complete list is shown in table 5. Some of these have been covered in the previous sections.

At present, the math font switching commands do not nest; therefore if you want sans serif bold, you must write `\mathsfbf{...}` rather than `\mathbf{\mathsf{...}}`. This may change in the future.

5.4.1 Double-struck

The double-struck alphabet (also known as 'blackboard bold') consists of upright Latin letters {𝕒–𝕫,𝔸ℤ}, numerals 𝟘–𝟡, summation symbol ⅀, and four Greek letters only: {ℽ▯ℾℿ}.

While `\mathbb{\sum}` does produce a double-struck summation symbol, its limits aren't properly aligned (see section §**??**). Therefore, either the literal character or the control sequence `\Bbbsum` are recommended instead.

There are also five Latin *italic* double-struck letters: ⅅⅆⅇⅈⅉ. These can be accessed (if not with their literal characters or control sequences) with the `\mathbbit`

Table 6: The various forms of nabla.

Description		Glyph
Upright	Serif	∇
	Bold serif	$\boldsymbol{\nabla}$
	Bold sans	⍰
Italic	Serif	$\mathit{\nabla}$
	Bold serif	$\boldsymbol{\mathit{\nabla}}$
	Bold sans	⍰

alphabet switch, but note that only those five letters will give the expected output.

5.5 Miscellanea

5.5.1 Nabla

The symbol ∇ comes in the six forms shown in table 6. We want an individual option to specify whether we want upright or italic nabla by default (when either upright or italic nabla is used in the source). TeX classically uses an upright nabla, but ISO standards differ (I think). The package options `nabla=upright` and `nabla=italic` switch between the two choices. This is then inherited through `\mathbf`; `\mathit` and `\mathup` can be used to force one way or the other.

`nabla=italic` is implicit when using `math-style=ISO` and `nabla=upright` follows both `math-style=TeX` and `math-style=French`.

5.5.2 Partial

The same applies to the symbols U+2202: PARTIAL DIFFERENTIAL and U+1D715: MATH ITALIC PARTIAL DIFFERENTIAL.

At time of writing, both the Cambria Math and STIX fonts display these two glyphs in the same italic style, but this is hopefully a bug that will be corrected in the future — the 'plain' partial differential should really have an upright shape.

Use the `partial=upright` or `partial=italic` package options to specify which one you would like. The default is (always, unless someone requests and argues otherwise) `partial=italic`.[1]

See table 7 for the variations on the partial differential symbol.

[1] A good argument would revolve around some international standards body recommending upright over italic. I just don't have the time right now to look it up.

Table 7: The various forms of the partial differential. Note that in the fonts used to display these glyphs, the first upright partial is incorrectly shown in an italic style.

Description		Glyph
Regular	Upright	∂
	Italic	∂
Bold	Upright	𝛛
	Italic	𝝏
Sans bold	Upright	⍰
	Italic	⍰

5.5.3 Epsilon and phi: ε vs. ϵ and φ vs. ϕ

TeX defines `\epsilon` to look like ϵ and `\varepsilon` to look like ε. The Unicode glyph directly after delta and before zeta is 'epsilon' and looks like ε; there is a subsequent variant of epsilon that looks like ϵ. This creates a problem. People who use unicode input won't want their glyphs transforming; TeX users will be confused that what they think as 'normal epsilon' is actual the 'variant epsilon'. And the same problem exists for 'phi'.

We have a package option to control this behaviour. With `vargreek-shape=TeX`, `\phi` and `\epsilon` produce φ and ε and `\varphi` and `\varepsilon` produce ϕ and ϵ. With `vargreek-shape=unicode`, these symbols are swapped. Note, however, that unicode characters are not affected by this option. That is, no remapping occurs of the characters/glyphs, only the control sequences.

The package default is to use `vargreek-shape=TeX`.

u+3b5: greek small letter epsilon
u+3f5: greek lunate epsilon symbol
u+3c6: greek small letter phi
u+3d5: greek small letter script phi

5.5.4 Primes

Primes (x') may be input in several ways. You may use any combination of ascii straight quote (`'`), unicode prime (′), and `\prime`; when multiple primes occur next to each other, they chain together to form double, triple, or quadruple primes if the font contains pre-drawn glyphs. These may also be accessed with `\primedouble`, `\primetriple`, and `\primequadruple`.

If the font does not contain the pre-drawn glyphs or more than four primes are used, the single prime glyph is used multiple times with a negative kern to get the spacing right. There is no user interface to adjust this negative kern yet (because I haven't decided what it should look like); if you need to, write something

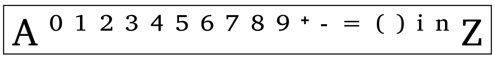

Figure 1: The unicode superscripts supported as input characters. These are the literal glyphs from Charis SIL, not the output seen when used for maths input. The 'A' and 'Z' are to provide context for the size and location of the superscript glyphs.

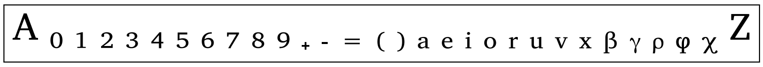

Figure 2: The unicode subscripts supported as input characters. See note from figure 1.

like this:

```
\ExplSyntaxOn
\muskip_gset:Nn \g_um_primekern_muskip { -\thinmuskip/2 }
\ExplySyntaxOff
```

5.5.5 Unicode subscripts and superscripts

You may, if you wish, use unicode subscripts and superscripts in your source document. For basic expressions, the use of these characters can make the input more readable. Adjacent sub- or super-scripts will be concatenated into a single expression.

The range of subscripts and superscripts supported by this package are shown in figures 1 and 2. Please request more if you think it is appropriate.

5.5.6 Colon

The colon is one of the few confusing characters of unicode maths. In TeX, `:` is defined as a colon with relation spacing: '$a : b$'. While `\colon` is defined as a colon with punctuation spacing: '$a\colon b$'.

In unicode, U+003A: COLON is defined as a punctuation symbol, while U+2236: RATIO is the colon-like symbol used in mathematics to denote ratios and other things.

This breaks the usual straightforward mapping from control sequence to unicode input character to (the same) unicode glyph.

To preserve input compatibility, we remap the ASCII input character '`:`' to U+2236: RATIO. Typing a literal U+2236: RATIO char will result in the same output. If amsmath is loaded, then the definition of `\colon` is inherited from there (it looks like a punctuation colon with additional space around it). Otherwise, `\colon` is made to output a colon with `\mathpunct` spacing.

Table 8: Slashes and backslashes.

Slot	Name	Glyph	Command
U+002F	SOLIDUS	/	`\solidus`
U+2044	FRACTION SLASH	⁄	`\fracslash`
U+2215	DIVISION SLASH	∕	`\slash`
U+29F8	BIG SOLIDUS	⧸	`\xsol`
U+005C	REVERSE SOLIDUS	\	`\backslash`
U+2216	SET MINUS	∖	`\smallsetminus`
U+29F5	REVERSE SOLIDUS OPERATOR	⧵	`\setminus`
U+29F9	BIG REVERSE SOLIDUS	⧹	`\xbsol`

The package option `[colon=literal]` forces ASCII input ‘`:`’ to be printed as `\mathcolon` instead.

5.5.7 Slashes and backslashes

There are several slash-like symbols defined in unicode. The complete list is shown in table 8.

In regular LaTeX we can write `\left\slash`…`\right\backslash` and so on and obtain extensible delimiter-like symbols. Not all of the unicode slashes are suitable for this (and do not have the font support to do it).

Slash Of U+2044: FRACTION SLASH, TR25 says that it is:

> …used to build up simple fractions in running text…however parsers of mathematical texts should be prepared to handle fraction slash when it is received from other sources.

U+2215: DIVISION SLASH should be used when division is represented without a built-up fraction; $\pi \approx 22/7$, for example.

U+29F8: BIG SOLIDUS is a ‘big operator’ (like $\sum$).

Backslash The U+005C: REVERSE SOLIDUS character `\backslash` is used for denoting double cosets: $A\backslash B$. (So I’m led to believe.) It may be used as a ‘stretchy’ delimiter if supported by the font.

MathML uses U+2216: SET MINUS like this: $A \smallsetminus B$.[2] The LaTeX command name `\smallsetminus` is used for backwards compatibility.

[2] §4.4.5.11 ⍰⍰⍰⍰://⍰⍰⍰.⍰3.⍰⍰⍰/⍰⍰/⍰⍰⍰⍰⍰⍰3/

Presumably, U+29F5: REVERSE SOLIDUS OPERATOR is intended to be used in a similar way, but it could also (perhaps?) be used to represent 'inverse division': $\pi \approx 7 \setminus 22$.[3] The LaTeX name for this character is `\setminus`.

Finally, U+29F9: BIG REVERSE SOLIDUS is a 'big operator' (like $\sum$).

How to use all of these things Unfortunately, font support for the above characters/glyphs is rather inconsistent. In Cambria Math, the only slash that grows (say when writing

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \Big/ \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \quad)$$

is the FRACTION SLASH, which we just established above is sort of only supposed to be used in text.

Of the above characters, the following are allowed to be used after `\left`, `\middle`, and `\right`:

- `\solidus`;
- `\fracslash`;
- `\slash`; and,
- `\backslash` (the only reverse slash).

However, we assume that there is only *one* stretchy slash in the font; this is assumed by default to be U+002F: SOLIDUS. Writing `\left/` or `\left\slash` or `\left`fracslash will all result in the same stretchy delimiter being used.

The delimiter used can be changed with the `slash-delimiter` package option. Allowed values are `ascii`, `frac`, and `div`, corresponding to the respective unicode slots.

For example: as mentioned above, Cambria Math's stretchy slash is U+2044: FRACTION SLASH. When using Cambria Math, then `unicode-math` should be loaded with the `[slash-delimiter=frac]` option. (This should be a font option rather than a package option, but it will change soon.)

5.5.8 Normalising some input characters

I believe all variant forms should be used as legal input that is normalised to a consistent output glyph, because we want to be fault-tolerant in the input. Here are the duplicates:

U+251: LATIN SMALL LETTER ALPHA

U+25B: LATIN SMALL LETTER EPSILON

[3] This is valid syntax in the Octave and Matlab programming languages, in which it means matrix inverse pre-multiplication. I.e., $A \setminus B \equiv A^{-1}B$.

U+263: LATIN SMALL LETTER GAMMA

U+269: LATIN SMALL LETTER IOTA

U+278: LATIN SMALL LETTER PHI

U+28A: LATIN SMALL LETTER UPSILON

U+190: LATIN CAPITAL LETTER EPSILON

U+194: LATIN CAPITAL LETTER GAMMA

U+196: LATIN CAPITAL LETTER IOTA

U+1B1: LATIN CAPITAL LETTER UPSILON

(Not yet implemented.)

File I
The unicode-math package

This is the package.

```
\ProvidesPackage{unicode-math}
  [2009/10/21 v0.4 Unicode maths in XeLaTeX]
```

6 Things we need

Packages

```
\RequirePackage{expl3}[2009/08/12]
\RequirePackage{xparse}[2009/08/31]
\RequirePackage{fontspec}
```

Start using LaTeX3 — finally!

```
\ExplSyntaxOn
```

Extras we need to define:

```
\cs_generate_variant:Nn \tl_put_right:Nn {cx}
\cs_generate_variant:Nn \seq_if_in:NnTF {NV}
\cs_generate_variant:Nn \prop_gput:Nnn {Nxn}
\cs_generate_variant:Nn \prop_get:NnN {cxN}
\cs_generate_variant:Nn \prop_if_in:NnTF {cx}
```

Counters and conditionals

```
\int_new:N \g_um_fam_int
\bool_new:N \l_um_fontspec_feature_bool
\bool_new:N \l_um_ot_math_bool
\bool_new:N \l_um_init_bool
```

For `math-style`:

```
\bool_new:N \g_um_literal_bool
\bool_new:N \g_um_upLatin_bool
\bool_new:N \g_um_uplatin_bool
\bool_new:N \g_um_upGreek_bool
\bool_new:N \g_um_upgreek_bool
```

For `bold-style`:

```
\bool_new:N \g_um_bfliteral_bool
\bool_new:N \g_um_bfupLatin_bool
\bool_new:N \g_um_bfuplatin_bool
\bool_new:N \g_um_bfupGreek_bool
\bool_new:N \g_um_bfupgreek_bool
```

For `nabla`:

```
\bool_new:N \g_um_upNabla_bool
\bool_new:N \g_um_uppartial_bool
\bool_new:N \g_um_texgreek_bool
```

6.0.9 Alphabet unicode positions

Before we begin, let's define the positions of the various unicode alphabets so that our code is a little more readable.[4]

```
\def\g_um_up_num_usv{48}
\def\g_um_up_Latin_usv{65}
\def\g_um_up_latin_usv{97}
\def\g_um_up_Greek_usv{"391}
\def\g_um_up_greek_usv{"3B1}
\def\g_um_it_Latin_usv{"1D434}
\def\g_um_it_latin_usv{"1D44E}
\def\g_um_it_Greek_usv{"1D6E2}
\def\g_um_it_greek_usv{"1D6FC}
\def\g_um_bb_num_usv{"1D7D8}
\def\g_um_bb_Latin_usv{"1D538}
\def\g_um_bb_latin_usv{"1D552}
\def\g_um_scr_Latin_usv{"1D49C}
\def\g_um_scr_latin_usv{"1D4B6}
\def\g_um_frak_Latin_usv{"1D504}
\def\g_um_frak_latin_usv{"1D51E}
\def\g_um_sf_num_usv{"1D7E2}
\def\g_um_sfup_num_usv{"1D7E2}
\def\g_um_sfit_num_usv{"1D7E2}
\def\g_um_sfup_Latin_usv{"1D5A0}
\def\g_um_sf_Latin_usv  {"1D5A0}
\def\g_um_sfup_latin_usv{"1D5BA}
```

[4] 'u.s.v.' stands for 'unicode scalar value'.

```
\def\g_um_sf_latin_usv{"1D5BA}
\def\g_um_sfit_Latin_usv{"1D608}
\def\g_um_sfit_latin_usv{"1D622}
\def\g_um_tt_num_usv{"1D7F6}
\def\g_um_tt_Latin_usv{"1D670}
\def\g_um_tt_latin_usv{"1D68A}
```

Bold:

```
\def\g_um_bf_num_usv  {"1D7CE}
\def\g_um_bfup_num_usv{"1D7CE}
\def\g_um_bfit_num_usv{"1D7CE}
\def\g_um_bfup_Latin_usv{"1D400}
\def\g_um_bfup_latin_usv{"1D41A}
\def\g_um_bfup_Greek_usv{"1D6A8}
\def\g_um_bfup_greek_usv{"1D6C2}
\def\g_um_bfit_Latin_usv{"1D468}
\def\g_um_bfit_latin_usv{"1D482}
\def\g_um_bfit_Greek_usv{"1D71C}
\def\g_um_bfit_greek_usv{"1D736}
\def\g_um_bffrak_Latin_usv{"1D56C}
\def\g_um_bffrak_latin_usv{"1D586}
\def\g_um_bfscr_Latin_usv{"1D4D0}
\def\g_um_bfscr_latin_usv{"1D4EA}
\def\g_um_bfsf_num_usv  {"1D7EC}
\def\g_um_bfsfup_num_usv{"1D7EC}
\def\g_um_bfsfit_num_usv{"1D7EC}
\def\g_um_bfsfup_Latin_usv{"1D5D4}
\def\g_um_bfsfup_latin_usv{"1D5EE}
\def\g_um_bfsfup_Greek_usv{"1D756}
\def\g_um_bfsfup_greek_usv{"1D770}
\def\g_um_bfsfit_Latin_usv{"1D63C}
\def\g_um_bfsfit_latin_usv{"1D656}
\def\g_um_bfsfit_Greek_usv{"1D790}
\def\g_um_bfsfit_greek_usv{"1D7AA}
\def\g_um_bfsf_Latin_usv  { \bool_if:NTF \g_um_upLatin_bool \g_um_bfsfup_Latin_usv \g_um_bfsfi
\def\g_um_bfsf_latin_usv  { \bool_if:NTF \g_um_uplatin_bool \g_um_bfsfup_latin_usv \g_um_bfsfi
\def\g_um_bfsf_Greek_usv  { \bool_if:NTF \g_um_upGreek_bool \g_um_bfsfup_Greek_usv \g_um_bfsfi
\def\g_um_bfsf_greek_usv  { \bool_if:NTF \g_um_upgreek_bool \g_um_bfsfup_greek_usv \g_um_bfsfi
\def\g_um_bf_Latin_usv  { \bool_if:NTF \g_um_bfupLatin_bool \g_um_bfup_Latin_usv \g_um_bfit_La
\def\g_um_bf_latin_usv  { \bool_if:NTF \g_um_bfuplatin_bool \g_um_bfup_latin_usv \g_um_bfit_la
\def\g_um_bf_Greek_usv  { \bool_if:NTF \g_um_bfupGreek_bool \g_um_bfup_Greek_usv \g_um_bfit_Gr
\def\g_um_bf_greek_usv  { \bool_if:NTF \g_um_bfupgreek_bool \g_um_bfup_greek_usv \g_um_bfit_gr
```

Greek variants:

```
\def\g_um_up_varTheta_usv{"3F4}
\def\g_um_up_Digamma_usv{"3DC}
\def\g_um_up_varepsilon_usv{"3F5}
\def\g_um_up_vartheta_usv{"3D1}
```

```
\def\g_um_up_varkappa_usv{"3F0}
\def\g_um_up_varphi_usv{"3D5}
\def\g_um_up_varrho_usv{"3F1}
\def\g_um_up_varpi_usv{"3D6}
\def\g_um_up_digamma_usv{"3DD}
```

Bold:

```
\def\g_um_bfup_varTheta_usv{"1D6B9}
\def\g_um_bfup_Digamma_usv{"1D7CA}
\def\g_um_bfup_varepsilon_usv{"1D6DC}
\def\g_um_bfup_vartheta_usv{"1D6DD}
\def\g_um_bfup_varkappa_usv{"1D6DE}
\def\g_um_bfup_varphi_usv{"1D6DF}
\def\g_um_bfup_varrho_usv{"1D6E0}
\def\g_um_bfup_varpi_usv{"1D6E1}
\def\g_um_bfup_digamma_usv{"1D7CB}
```

Italic Greek variants:

```
\def\g_um_it_varTheta_usv{"1D6F3}
\def\g_um_it_varepsilon_usv{"1D716}
\def\g_um_it_vartheta_usv{"1D717}
\def\g_um_it_varkappa_usv{"1D718}
\def\g_um_it_varphi_usv{"1D719}
\def\g_um_it_varrho_usv{"1D71A}
\def\g_um_it_varpi_usv{"1D71B}
```

Bold italic:

```
\def\g_um_bfit_varTheta_usv{"1D72D}
\def\g_um_bfit_varepsilon_usv{"1D750}
\def\g_um_bfit_vartheta_usv{"1D751}
\def\g_um_bfit_varkappa_usv{"1D752}
\def\g_um_bfit_varphi_usv{"1D753}
\def\g_um_bfit_varrho_usv{"1D754}
\def\g_um_bfit_varpi_usv{"1D755}
```

Bold sans:

```
\def\g_um_bfsfup_varTheta_usv{"1D767}
\def\g_um_bfsfup_varepsilon_usv{"1D78A}
\def\g_um_bfsfup_vartheta_usv{"1D78B}
\def\g_um_bfsfup_varkappa_usv{"1D78C}
\def\g_um_bfsfup_varphi_usv{"1D78D}
\def\g_um_bfsfup_varrho_usv{"1D78E}
\def\g_um_bfsfup_varpi_usv{"1D78F}
```

Bold sans italic:

```
\def\g_um_bfsfit_varTheta_usv{"1D7A1}
\def\g_um_bfsfit_varepsilon_usv{"1D7C4}
\def\g_um_bfsfit_vartheta_usv{"1D7C5}
\def\g_um_bfsfit_varkappa_usv{"1D7C6}
```

```
\def\g_um_bfsfit_varphi_usv{"1D7C7}
\def\g_um_bfsfit_varrho_usv{"1D7C8}
\def\g_um_bfsfit_varpi_usv{"1D7C9}
```

Nabla:

```
\def\g_um_up_Nabla_usv{"2207}
\def\g_um_it_Nabla_usv{"1D6FB}
\def\g_um_bfup_Nabla_usv{"1D6C1}
\def\g_um_bfit_Nabla_usv{"1D735}
\def\g_um_bfsfup_Nabla_usv{"1D76F}
\def\g_um_bfsfit_Nabla_usv{"1D7A9}
```

Partial:

```
\def\g_um_up_partial_usv{"2202}
\def\g_um_it_partial_usv{"1D715}
\def\g_um_bfup_partial_usv{"1D6DB}
\def\g_um_bfit_partial_usv{"1D74F}
\def\g_um_bfsfup_partial_usv{"1D789}
\def\g_um_bfsfit_partial_usv{"1D7C3}
```

Latin 'h':

```
\def\g_um_up_h_usv  {"0068}
\def\g_um_it_h_usv  {"210E}
\def\g_um_bb_h_usv  {"1D559}
\def\g_um_tt_h_usv  {"1D691}
\def\g_um_scr_h_usv {"1D4BD}
\def\g_um_frak_h_usv{"1D525}
\def\g_um_bfup_h_usv{"1D421}
\def\g_um_bfit_h_usv{"1D489}
\def\g_um_sfup_h_usv{"1D5C1}
\def\g_um_sfit_h_usv{"1D629}
\def\g_um_bffrak_h_usv{"1D58D}
\def\g_um_bfscr_h_usv {"1D4F1}
\def\g_um_bfsfup_h_usv {"1D5F5}
\def\g_um_bfsfit_h_usv {"1D65D}
```

6.1 Options

xkeyval's package support is used here. I'll switch over to l3keys2e at some stage.

`\unimathsetup` This macro can be used in lieu of or later to override options declared when the package is loaded.

```
\DeclareDocumentCommand \unimathsetup {m} {
  \setkeys{unicode-math.sty}{#1}
}
```

math-style

```
\define@choicekey*{unicode-math.sty}
    {math-style}[\@tempa\@tempb]{iso,tex,french,upright,literal}{
  \ifcase\@tempb\relax
    \bool_set_false:N \g_um_upGreek_bool
    \bool_set_false:N \g_um_upgreek_bool
    \bool_set_false:N \g_um_upLatin_bool
    \bool_set_false:N \g_um_uplatin_bool
    \bool_set_false:N \g_um_bfupGreek_bool
    \bool_set_false:N \g_um_bfupgreek_bool
    \bool_set_false:N \g_um_bfupLatin_bool
    \bool_set_false:N \g_um_bfuplatin_bool
    \bool_set_false:N \g_um_upNabla_bool
    \bool_set_false:N \g_um_uppartial_bool
    \bool_set_false:N \g_um_upsans_bool
    \bool_set_false:N \g_um_texgreek_bool
    \bool_set_false:N \g_um_literal_bool
  \or
    \bool_set_true:N \g_um_upGreek_bool
    \bool_set_false:N \g_um_upgreek_bool
    \bool_set_false:N \g_um_upLatin_bool
    \bool_set_false:N \g_um_uplatin_bool
    \bool_set_true:N \g_um_bfupGreek_bool
    \bool_set_false:N \g_um_bfupgreek_bool
    \bool_set_true:N \g_um_bfupLatin_bool
    \bool_set_true:N \g_um_bfuplatin_bool
    \bool_set_true:N \g_um_upNabla_bool
    \bool_set_false:N \g_um_uppartial_bool
    \bool_set_true:N \g_um_upsans_bool
    \bool_set_false:N \g_um_texgreek_bool
    \bool_set_false:N \g_um_literal_bool
  \or
    \bool_set_true:N \g_um_upGreek_bool
    \bool_set_true:N \g_um_upgreek_bool
    \bool_set_true:N \g_um_upLatin_bool
    \bool_set_false:N \g_um_uplatin_bool
    \bool_set_true:N \g_um_bfupGreek_bool
    \bool_set_true:N \g_um_bfupgreek_bool
    \bool_set_true:N \g_um_bfupLatin_bool
    \bool_set_true:N \g_um_bfuplatin_bool
    \bool_set_true:N \g_um_upNabla_bool
    \bool_set_true:N \g_um_uppartial_bool
    \bool_set_true:N \g_um_upsans_bool
    \bool_set_false:N \g_um_texgreek_bool
    \bool_set_false:N \g_um_literal_bool
  \or
```

```
    \bool_set_true:N \g_um_upGreek_bool
    \bool_set_true:N \g_um_upgreek_bool
    \bool_set_true:N \g_um_upLatin_bool
    \bool_set_true:N \g_um_uplatin_bool
    \bool_set_true:N \g_um_bfupGreek_bool
    \bool_set_true:N \g_um_bfupgreek_bool
    \bool_set_true:N \g_um_bfupLatin_bool
    \bool_set_true:N \g_um_bfuplatin_bool
    \bool_set_true:N \g_um_upNabla_bool
    \bool_set_true:N \g_um_uppartial_bool
    \bool_set_true:N \g_um_upsans_bool
    \bool_set_false:N \g_um_texgreek_bool
    \bool_set_false:N \g_um_literal_bool
  \or
    \bool_set_true:N \g_um_literal_bool
    \bool_set_true:N \g_um_bfliteral_bool
    \bool_set_true:N \g_um_sfliteral_bool
    \bool_set_false:N \g_um_texgreek_bool
  \fi
}
```

bold-style

```
\define@choicekey*{unicode-math.sty}{bold-style}[\@tempa\@tempb]{iso,tex,upright,literal}{
  \ifcase\@tempb\relax
    \bool_set_false:N \g_um_bfliteral_bool
    \bool_set_false:N \g_um_bfupGreek_bool
    \bool_set_false:N \g_um_bfupgreek_bool
    \bool_set_false:N \g_um_bfupLatin_bool
    \bool_set_false:N \g_um_bfuplatin_bool
  \or
    \bool_set_false:N \g_um_bfliteral_bool
    \bool_set_true:N \g_um_bfupGreek_bool
    \bool_set_false:N \g_um_bfupgreek_bool
    \bool_set_true:N \g_um_bfupLatin_bool
    \bool_set_true:N \g_um_bfuplatin_bool
  \or
    \bool_set_false:N \g_um_bfliteral_bool
    \bool_set_true:N \g_um_bfupGreek_bool
    \bool_set_true:N \g_um_bfupgreek_bool
    \bool_set_true:N \g_um_bfupLatin_bool
    \bool_set_true:N \g_um_bfuplatin_bool
  \or
    \bool_set_true:N \g_um_bfliteral_bool
  \fi
}
```

sans-style

```
\bool_new:N \g_um_upsans_bool
\bool_new:N \g_um_sfliteral_bool
\define@choicekey*{unicode-math.sty}
    {sans-style}[\@tempa\@tempb]{italic,upright,literal}{
  \ifcase\@tempb\relax
    \bool_set_false:N \g_um_upsans_bool
  \or
    \bool_set_true:N \g_um_upsans_bool
  \or
    \bool_set_true:N \g_um_sfliteral_bool
  \fi
}
```

Symbol obliqueness

```
\define@choicekey*{unicode-math.sty}{nabla}[\@tempa\@tempb]{upright,italic}{
  \ifcase\@tempb
    \bool_set_true:N \g_um_upNabla_bool
  \or
    \bool_set_false:N \g_um_upNabla_bool
  \fi
}
\cs_set:Nn \um_setup_nabla: {
  \bool_if:NTF \g_um_upNabla_bool {
    \tl_set:Nn \g_um_Nabla_up_or_it_usv     { \g_um_up_Nabla_usv }
    \tl_set:Nn \g_um_bfNabla_up_or_it_usv   { \g_um_bfup_Nabla_usv }
    \tl_set:Nn \g_um_bfsfNabla_up_or_it_usv { \g_um_bfsfup_Nabla_usv }
  }{
    \tl_set:Nn \g_um_Nabla_up_or_it_usv     { \g_um_it_Nabla_usv }
    \tl_set:Nn \g_um_bfNabla_up_or_it_usv   { \g_um_bfit_Nabla_usv }
    \tl_set:Nn \g_um_bfsfNabla_up_or_it_usv { \g_um_bfsfit_Nabla_usv }
  }
}
```

```
\define@choicekey*{unicode-math.sty}{partial}[\@tempa\@tempb]{upright,italic}{
  \ifcase\@tempb
    \bool_set_true:N \g_um_uppartial_bool
  \or
    \bool_set_false:N \g_um_uppartial_bool
  \fi
}
\cs_set:Nn \um_setup_partial: {
  \bool_if:NTF \g_um_uppartial_bool {
    \tl_set:Nn \g_um_partial_up_or_it_usv     { \g_um_up_partial_usv }
    \tl_set:Nn \g_um_bfpartial_up_or_it_usv   { \g_um_bfup_partial_usv }
    \tl_set:Nn \g_um_bfsfpartial_up_or_it_usv { \g_um_bfsfup_partial_usv }
  }{
    \tl_set:Nn \g_um_partial_up_or_it_usv     { \g_um_it_partial_usv }
    \tl_set:Nn \g_um_bfpartial_up_or_it_usv   { \g_um_bfit_partial_usv }
    \tl_set:Nn \g_um_bfsfpartial_up_or_it_usv { \g_um_bfsfit_partial_usv }
  }
}
```

Epsilon and phi shapes

```
\define@choicekey*{unicode-math.sty}{vargreek-shape}[\@tempa\@tempb]{unicode,TeX}{
  \ifcase\@tempb
    \bool_set_false:N \g_um_texgreek_bool
  \or
    \bool_set_true:N \g_um_texgreek_bool
  \fi
}
```

Colon style

```
\bool_new:N \g_um_literal_colon_bool
\define@choicekey*{unicode-math.sty}{colon}[\@tempa\@tempb]{literal,TeX}{
  \ifcase\@tempb
    \bool_set_true:N \g_um_literal_colon_bool
  \or
    \bool_set_false:N \g_um_literal_colon_bool
  \fi
}
```

Slash delimiter style

```
\define@choicekey*{unicode-math.sty}{slash-delimiter}[\@tempa\@tempb]{ascii,frac,div}{
  \ifcase\@tempb
    \tl_set:Nn \g_um_slash_delimiter_usv {"002F}
  \or
    \tl_set:Nn \g_um_slash_delimiter_usv {"2044}
  \or
    \tl_set:Nn \g_um_slash_delimiter_usv {"2215}
  \fi
}
```

```
\ExecuteOptionsX{math-style=TeX,slash-delimiter=ascii}
\ProcessOptionsX
```

6.2 Overcoming \@onlypreamble

The requirement of only setting up the maths fonts in the preamble is now removed. The following list might be overly ambitious.

```
\tl_map_inline:nn {
```

```
\new@mathgroup\cdp@list\cdp@elt\DeclareMathSizes
\@DeclareMathSizes\newmathalphabet\newmathalphabet@@\newmathalphabet@@@
\DeclareMathVersion\define@mathalphabet\define@mathgroup\addtoversion
\version@list\version@elt\alpha@list\alpha@elt
\restore@mathversion\init@restore@version\dorestore@version\process@table
\new@mathversion\DeclareSymbolFont\group@list\group@elt
\new@symbolfont\SetSymbolFont\SetSymbolFont@\get@cdp
\DeclareMathAlphabet\new@mathalphabet\SetMathAlphabet\SetMathAlphabet@
\DeclareMathAccent\set@mathaccent\DeclareMathSymbol\set@mathchar
\set@mathsymbol\DeclareMathDelimiter\@xxDeclareMathDelimiter\@DeclareMathDelimiter
\@xDeclareMathDelimiter\set@mathdelimiter\set@@mathdelimiter\DeclareMathRadical
\mathchar@type\DeclareSymbolFontAlphabet\DeclareSymbolFontAlphabet@
}{
  \tl_remove_in:Nn \@preamblecmds {\do#1}
}
```

6.3 Other things

`\um_fontdimen_to_percent:nn`

#1 : Font dimen number

`\fontdimens` 10, 11, and 65 aren't actually dimensions, they're percentage values given in units of `sp`. This macro takes a font dimension number and outputs the decimal value of the associated parameter.

```
\def\um_fontdimen_to_percent:nn#1#2{
  0.\strip@pt\dimexpr\fontdimen#1#2 *65536\relax
}
```

`\um@scaled@apply`

#1 : A math style

#2 : Macro that takes a non-delimited length argument (like `\kern`)

#3 : Length control sequence to be scaled according to the math style

This macro is used to scale the lengths reported by `\fontdimen` according to the scale factor for script- and scriptscript-size objects.

```
\def\um@scaled@apply#1#2#3{
  \ifx#1\scriptstyle
    #2\um_fontdimen_to_percent:nn{10}\l_um_font#3
  \else
    \ifx#1\scriptscriptstyle
      #2\um_fontdimen_to_percent:nn{11}\l_um_font#3
    \else
      #2#3%
    \fi
  \fi
}
```

7 Fundamentals

7.1 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
```

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

7.2 `\DeclareMathSymbol` for unicode ranges

This command is a bit funny at the moment; it doesn't define the actual macro for almost all of the symbols passed to it, but it does assign the `\XeTeXmathchar`.

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

`\um_set_mathsymbol:nNNn`

`#1` : Symbol font number, e.g., `\symoperators`
`#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.

```
\cs_set:Nn \um_set_mathsymbol:nNNn {
```

Operators In the examples following, say we're defining for the symbol `\sum`(∑).

```
  \ifx\mathop#3\relax
```

In order for literal unicode characters to be used in the source and still have the correct limits behaviour, big operators are made math-active.

The active math char is defined to expand to the macro `\sum_sym`.

```
    \begingroup
      \char_make_active:n {#4}
      \global\mathcode#4="8000\relax
     \um@scanactivedef #4 \@nil { \csname\cs_to_str:N #2 _sym\endcsname }
    \endgroup
```

Some of these require a `\nolimits` suffix. This is controlled by the `\um@nolimits` macro, which contains a list of such characters. This list is checked dynamically because we're not interested in efficiency. Or something. This allows the list to be updated in the middle of a document.

Declare the plain old mathchardef for the control sequence `\sumop`.

```
    \expandafter\global\expandafter\XeTeXmathchardef
      \csname\cs_to_str:N #2 op\endcsname ="\mathchar@type#3 #1 #4\relax
```

Now define \sum_sym as \sumop, followed by \nolimits if necessary.

```
    \cs_gset:cpx { \cs_to_str:N #2 _sym } {
      \exp_not:c {\cs_to_str:N #2 op}
      \exp_not:n {\tl_if_in:NnT \l_um_nolimits_tl {#2} \nolimits}
    }
```

Don't forget that the actual \sum macro is simply defined in terms of the literal unicode symbol!

```
  \else
```

Delimiters and radicals Sqrt radical is defined as a csmathopen.

```
    \ifx\mathopen#3\relax
      \tl_if_in:NnTF \l_um_radicals_tl #2 {
       \cs_gset:cpn {\cs_to_str:N #2 sign} { \XeTeXradical #1 #4 \relax }
      }{
        \cs_gset:Npn #2 {\XeTeXdelimiter "\mathchar@type#3 #1 #4\relax}
        \global\XeTeXdelcode#4=#1 #4\relax
        \global\XeTeXmathcode#4="\mathchar@type#3 #1 #4\relax
      }
    \else
      \ifx\mathclose#3\relax
        \cs_gset:Npn #2 {\XeTeXdelimiter "\mathchar@type#3 #1 #4\relax}
        \global\XeTeXdelcode#4=#1 #4\relax
        \global\XeTeXmathcode#4="\mathchar@type#3 #1 #4\relax
      \else
```

Fences

```
        \ifx\mathfence#3
          \global\XeTeXmathcode#4="\mathchar@type#3 #1 #4\relax
          \global\XeTeXdelcode#4=#1 #4\relax
              \cs_gset:cpn {l \cs_to_str:N #2} {\XeTeXdelimiter "\mathchar@type\mathopen #1 #4\relax}
              \cs_gset:cpn {r \cs_to_str:N #2} {\XeTeXdelimiter "\mathchar@type\mathclose #1 #4\relax}
        \else
```

Accents

```
          \ifx\mathaccent#3\relax
          \cs_gset:Npx #2 {\XeTeXmathaccent "\mathchar@type#3 #1 #4\relax}
          \else
```

And finally, the general case. We define the unicode mathcode for the character. The macro is defined later on generically in terms of the unicode character.

```
            \global\XeTeXmathcode#4="\mathchar@type#3 #1 #4\relax
          \fi
```

```
        \fi
      \fi
    \fi
  \fi
}
```

\um_set_mathcode:nnnn Note that this declaration *isn't* global so that it can be constrained by grouping inside math alphabet switches.

```
\cs_set:Nn \um_set_mathcode:nnnn {
  \XeTeXmathcode#1="\mathchar@type#2 \csname sym#3\endcsname #4\relax
}
```

7.3 The main \setmathfont macro

Using a range including large character sets such as \mathrel, \mathalpha, *etc.*, is *very slow*! I hope to improve the performance somehow.

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

```
\DeclareDocumentCommand \setmathfont { O{} m } {
```

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

```
       \let\glb@currsize\relax
```

- To start with, assume we're defining the font for every math symbol character.

```
       \bool_set_true:N \l_um_init_bool
       \seq_clear:N \l_um_char_range_seq
       \let\um@char@num@range\@empty
```

- Grab the current size information (is this robust enough? Maybe it should be preceded by \normalsize).

```
       \csname S@\f@size\endcsname
```

- Set the name of the math version being defined. (obviously more needs to be done here!)

```
  \tl_set:Nn \l_um_mversion_tf {normal}
  \DeclareMathVersion{\l_um_mversion_tf}
```

Define default font features for the script and scriptscript font.

```
  \tl_set:Nn \l_um_script_features_tl  {ScriptStyle}
  \tl_set:Nn \l_um_sscript_features_tl {ScriptScriptStyle}
  \tl_set:Nn \l_um_script_font_tl      {#2}
  \tl_set:Nn \l_um_sscript_font_tl     {#2}
```

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.

```
  \setkeys*{unicode-math.sty}{#1}
  \cs_set:Npx \um_tmp: {
    \exp_not:N \setkeys*[um]{options}{\exp_not:V \XKV@rm}
  }
  \um_tmp:
  \cs_set:Npx \um_tmp: {
    \exp_not:N \zf@fontspec {
      BoldFont = {}, ItalicFont = {},
      Script = Math,
      SizeFeatures = {
        {Size = \tf@size-} ,
        {Size = \sf@size-\tf@size ,
         Font = \l_um_script_font_tl ,
         \l_um_script_features_tl
        } ,
        {Size = -\sf@size ,
         Font = \l_um_sscript_font_tl ,
         \l_um_sscript_features_tl
        }
      },
      \XKV@rm
    }{#2}
  }
  \bool_set_true:N \l_um_fontspec_feature_bool
  \um_tmp:
  \bool_set_false:N \l_um_fontspec_feature_bool
```

Check for the correct number of `\fontdimen`s:

```
  \font\l_um_font="#2"\relax
%%  \ifdim \dimexpr\fontdimen9\l_um_font*65536\relax =65pt\relax
%%    \bool_set_true:N \l_um_ot_math_bool
%%  \else
%%    \bool_set_false:N \l_um_ot_math_bool
%%    \PackageWarningNoLine{unicode-math}{
%%      The~ font~ '#2' ~is~ not~ a~ valid~ OpenType~ maths~ font.~
%%      Some~ maths~ features~ will~ not~ be~ available~ or~ behave~
%%      in~ a~ substandard~ manner
%%    }
%%  \fi
```

If we're defining the full unicode math repetoire, then we skip all the parsing processing needed if we're only defining a subset.

- Math symbols are defined with `\UnicodeMathSymbol`; see section §7.3.1 for the individual definitions

```
  \bool_if:NTF \l_um_init_bool {
    \tl_set:Nn \um_symfont_tl {um_allsym}
   \PackageInfo{unicode-math}{Defining~ the~ default~ maths~ font~ as~ '#2'}
    \cs_set_eq:NN \UnicodeMathSymbol \um_process_symbol_noparse:nnnn
    \cs_set_eq:NN \um_mathmap:Nnn \um_mathmap_noparse:Nnn
    \cs_set_eq:NN \um_remap_symbol:nnn \um_remap_symbol_noparse:nnn
    \cs_set_eq:NN \um_maybe_init_alphabet:n \um_init_alphabet:n
    \cs_set_eq:NN \um_map_char_internal:nn \um_map_char_noparse:nn
  }{
    \int_incr:N \g_um_fam_int
    \tl_set:Nx \um_symfont_tl {um_fam\int_use:N\g_um_fam_int}
    \cs_set_eq:NN \UnicodeMathSymbol \um_process_symbol_parse:nnnn
    \cs_set_eq:NN \um_mathmap:Nnn \um_mathmap_parse:Nnn
    \cs_set_eq:NN \um_remap_symbol:nnn \um_remap_symbol_parse:nnn
    \cs_set_eq:NN \um_maybe_init_alphabet:n \use_none:n
    \cs_set_eq:NN \um_map_char_internal:nn \um_map_char_parse:nn
  }
```

Now defined `\um_symfont_tl` as the LaTeX math font to access everything:

```
  \DeclareSymbolFont{\um_symfont_tl}
    {\encodingdefault}{\zf@family}{\mddefault}{\updefault}
```

And now we input every single maths char. See File 12 for the source to `unicode-math.tex` which is used to create `unicode-math-table.tex`.

```
  \@input{unicode-math-table.tex}
```

Finally,

- Set up shapes for italic/upright or ordinary/var symbols as per package options.
- Remap symbols that don't take their natural mathcode
- Activate any symbols that need to be math-active
- Assign delimiter codes for symbols that need to grow
- Setup the maths alphabets (`\mathbf` etc.)

```
  \um_setup_nabla:
  \um_setup_partial:
  \um_remap_symbols:
```

```
  \um_setup_mathactives:
  \um_setup_delcodes:
  \um_setup_alphabets:
}
```

7.3.1 Functions for setting up symbols with mathcodes

\um_process_symbol_noparse:nnnn
\um_process_symbol_parse:nnnn

If the range font feature has been used, then only a subset of the unicode glyphs are to be defined. See section §8.3 for the code that enables this.

```
\cs_set:Nn \um_process_symbol_noparse:nnnn {
  \exp_args:Nc \um_set_mathsymbol:nNNn {sym\um_symfont_tl}#2#3{#1}
}
```

```
\cs_set:Nn \um_process_symbol_parse:nnnn {
  \um@parse@term{#1}{#2}{#3}{
    \um_process_symbol_noparse:nnnn{#1}{#2}{#3}{#4}
  }
}
```

\um_remap_symbols:
\um_remap_symbol_noparse:nnn
\um_remap_symbol_parse:nnn

This function is used to define the mathcodes for those chars which should be mapped to a different glyph than themselves.

```
\cs_new:Nn \um_remap_symbols: {
  \um_remap_symbol:nnn{`\-}{\mathbin}{"02212}% hyphen to minus
   \um_remap_symbol:nnn{`\*}{\mathbin}{"02217}% text asterisk to "centred asterisk"
  \bool_if:NF \g_um_literal_colon_bool {
   \um_remap_symbol:nnn{`\:}{\mathrel}{"02236}% colon to ratio (i.e., punct to rel)
  }
  \bool_if:NTF \g_um_literal_bool {
   \um_remap_symbol:nnn {\g_um_up_Nabla_usv}{\mathord}{\g_um_up_Nabla_usv}
   \um_remap_symbol:nnn {\g_um_it_Nabla_usv}{\mathord}{\g_um_it_Nabla_usv}
   \um_remap_symbol:nnn {\g_um_up_partial_usv}{\mathord}{\g_um_up_partial_usv}
   \um_remap_symbol:nnn {\g_um_it_partial_usv}{\mathord}{\g_um_it_partial_usv}
  }{
   \um_remap_symbol:nnn {\g_um_up_Nabla_usv,\g_um_it_Nabla_usv}{\mathord}{\g_um_Nabla_up_or_
   \um_remap_symbol:nnn {\g_um_up_partial_usv,\g_um_it_partial_usv}{\mathord}{\g_um_partial_
  }
```

Some of these in the bfliteral block may be redundant, but that's okay:

```
  \bool_if:NTF \g_um_bfliteral_bool {
   \um_remap_symbol:nnn {\g_um_bfup_Nabla_usv   }{\mathord}{\g_um_bfup_Nabla_usv}
   \um_remap_symbol:nnn {\g_um_bfit_Nabla_usv   }{\mathord}{\g_um_bfit_Nabla_usv}
   \um_remap_symbol:nnn {\g_um_bfsfup_Nabla_usv  }{\mathord}{\g_um_bfsfup_Nabla_usv}
   \um_remap_symbol:nnn {\g_um_bfsfit_Nabla_usv  }{\mathord}{\g_um_bfsfit_Nabla_usv}
   \um_remap_symbol:nnn {\g_um_bfup_partial_usv  }{\mathord}{\g_um_bfup_partial_usv}
   \um_remap_symbol:nnn {\g_um_bfit_partial_usv  }{\mathord}{\g_um_bfit_partial_usv}
   \um_remap_symbol:nnn {\g_um_bfsfup_partial_usv}{\mathord}{\g_um_bfsfup_partial_usv}
   \um_remap_symbol:nnn {\g_um_bfsfit_partial_usv}{\mathord}{\g_um_bfsfit_partial_usv}
  }{
   \um_remap_symbol:nnn {\g_um_bfup_Nabla_usv,\g_um_bfit_Nabla_usv}{\mathord}{\g_um_bfNabla_
   \um_remap_symbol:nnn {\g_um_bfsfup_Nabla_usv,\g_um_bfsfit_Nabla_usv}{\mathord}{\g_um_bfsf
   \um_remap_symbol:nnn {\g_um_bfup_partial_usv,\g_um_bfit_partial_usv}{\mathord}{\g_um_bfpa
   \um_remap_symbol:nnn {\g_um_bfsfup_partial_usv,\g_um_bfsfit_partial_usv}{\mathord}{\g_um_
  }
}
```

Where `\um_remap_symbol:nnn` is defined to be one of these two, depending on the range setup:

```
\cs_new:Nn \um_remap_symbol_parse:nnn {
  \um@parse@term {#3} {\@nil} {#2} {
    \um_remap_symbol_noparse:nnn {#1} {#2} {#3}
  }
}
\cs_new:Nn \um_remap_symbol_noparse:nnn {
  \clist_map_inline:nn {#1} {
    \um_set_mathcode:nnnn {##1} {#2} {\um_symfont_tl} {#3}
  }
}
```

7.3.2 Active math characters

There are more math active chars later in the subscript/superscript section. But they don't need to be able to be typeset directly.

`\um_setup_mathactives:`

```
\cs_new:Nn \um_setup_mathactives: {
  \um_make_mathactive:nNN {"2032} \sprime \mathord
}
```

`\um_make_mathactive:nNN`

: TODO : hook into range feature Makes `#1` a mathactive char, and gives cs `#2` the meaning of mathchar `#1` with class `#3`. You are responsible for giving active `#1` a particular meaning!

```
\cs_new:Nn \um_make_mathactive:nNN {
  \XeTeXmathchardef #2 = "\mathchar@type #3
                         \csname sym\um_symfont_tl\endcsname
                         #1 \scan_stop:
  \XeTeXmathcodenum #1 = "1FFFFF \scan_stop:
}
```

7.3.3 Delimiter codes

Some symbols that aren't mathopen/mathclose still need to have delimiter codes assigned. The list of vertical arrows may be incomplete. On the other hand, many fonts won't support them all being stretchy. And some of them are probably not meant to stretch, either. But adding them here doesn't hurt.

\um_setup_delcodes:

```
\cs_new:Nn \um_setup_delcodes: {
  \um_set_delcode:nn {`\/}   {\g_um_slash_delimiter_usv}
  \um_set_delcode:nn {"2044} {\g_um_slash_delimiter_usv} % fracslash
  \um_set_delcode:nn {"2215} {\g_um_slash_delimiter_usv} % divslash
  \um_set_delcode:n {"005C} % backslash
  \um_set_delcode:nn {`\<} {"27E8} % angle brackets with ascii notation
  \um_set_delcode:nn {`\>} {"27E9} % angle brackets with ascii notation
  \um_set_delcode:n {"2191} % up arrow
  \um_set_delcode:n {"2193} % down arrow
  \um_set_delcode:n {"2195} % updown arrow
  \um_set_delcode:n {"219F} % up arrow twohead
  \um_set_delcode:n {"21A1} % down arrow twohead
  \um_set_delcode:n {"21A5} % up arrow from bar
  \um_set_delcode:n {"21A7} % down arrow from bar
  \um_set_delcode:n {"21A8} % updown arrow from bar
  \um_set_delcode:n {"21BE} % up harpoon right
  \um_set_delcode:n {"21BF} % up harpoon left
  \um_set_delcode:n {"21C2} % down harpoon right
  \um_set_delcode:n {"21C3} % down harpoon left
  \um_set_delcode:n {"21C5} % arrows up down
  \um_set_delcode:n {"21F5} % arrows down up
  \um_set_delcode:n {"21C8} % arrows up up
  \um_set_delcode:n {"21CA} % arrows down down
  \um_set_delcode:n {"21D1} % double up arrow
  \um_set_delcode:n {"21D3} % double down arrow
  \um_set_delcode:n {"21D5} % double updown arrow
  \um_set_delcode:n {"21DE} % up arrow double stroke
  \um_set_delcode:n {"21DF} % down arrow double stroke
  \um_set_delcode:n {"21E1} % up arrow dashed
  \um_set_delcode:n {"21E3} % down arrow dashed
  \um_set_delcode:n {"21E7} % up white arrow
  \um_set_delcode:n {"21E9} % down white arrow
  \um_set_delcode:n {"21EA} % up white arrow from bar
  \um_set_delcode:n {"21F3} % updown white arrow
}
```

\um_set_delcode:nn
\um_set_delcode:n

: TODO : hook into range feature

```
\cs_new:Nn \um_set_delcode:nn {
  \XeTeXdelcode#1 = \csname sym\um_symfont_tl\endcsname #2
}
\cs_new:Nn \um_set_delcode:n {
  \XeTeXdelcode#1 = \csname sym\um_symfont_tl\endcsname #1
}
```

7.3.4 Maths alphabets' character mapping

7.3.5 Functions for setting up the maths alphabets

`\um_mathmap_noparse:Nnn`

`#1` : Maths alphabet, *e.g.*, `\mathbb`
`#2` : Input slot(s), *e.g.*, the slot for 'A' (comma separated)
`#3` : Output slot, *e.g.*, the slot for '𝔸'
Adds `\um_set_mathcode:nnnn` declarations to the specified maths alphabet's definition.

```
\cs_set:Nn \um_mathmap_noparse:Nnn {
  \clist_map_inline:nn {#2} {
    \tl_put_right:cx {um_setup_\cs_to_str:N #1:} {
    \exp_not:N\um_set_mathcode:nnnn{##1}{\exp_not:N\mathalpha}{\um_symfont_tl}{#3}
    }
  }
}
```

`\um_mathmap_parse:Nnn`

`#1` : Maths alphabet, *e.g.*, `\mathbb`
`#2` : Input slot(s), *e.g.*, the slot for 'A' (comma separated)
`#3` : Output slot, *e.g.*, the slot for '𝔸'
When `\um@parse@term` is executed, it populates the `\um@char@num@range` macro with slot numbers corresponding to the specified range. This range is used to conditionally add `\um_set_mathcode:nnnn` declaractions to the maths alphabet definition.

```
\cs_set:Nn \um_mathmap_parse:Nnn {
  \clist_map_inline:Nn \um@char@num@range {
    \ifnum##1=#3\relax
      \um_mathmap_noparse:Nnn {#1}{#2}{#3}
    \fi
  }
}
```

7.4 (Big) operators

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

However, the limits aren't set automatically; that is, we want to define, a la Plain TeX *etc.*, `\def\int{\intop\nolimits}`, so there needs to be a transformation from `\int` to `\intop` during the expansion of `\UnicodeMathSymbol` in the appropriate contexts.

Following is a table of every math operator (`\mathop`) defined in `unicode-math-table.tex`, from which a subset need to be flagged for `\nolimits` adjustments. The limits behaviour as specified by unicode-math are shown (with grey 'scripts).

USV	Ex.	Macro	Description
U+02140	$\Bbbsum_{0}^{1}$	`\Bbbsum`	DOUBLE-STRUCK N-ARY SUMMATION
U+0220F	$\prod_{0}^{1}$	`\prod`	PRODUCT OPERATOR
U+02210	$\coprod_{0}^{1}$	`\coprod`	COPRODUCT OPERATOR
U+02211	$\sum_{0}^{1}$	`\sum`	SUMMATION OPERATOR
U+0222B	$\int_{0}^{1}$	`\int`	INTEGRAL OPERATOR
U+0222C	$\iint_{0}^{1}$	`\iint`	DOUBLE INTEGRAL OPERATOR
U+0222D	$\iiint_{0}^{1}$	`\iiint`	TRIPLE INTEGRAL OPERATOR
U+0222E	$\oint_{0}^{1}$	`\oint`	CONTOUR INTEGRAL OPERATOR
U+0222F	$\oiint_{0}^{1}$	`\oiint`	DOUBLE CONTOUR INTEGRAL OPERATOR
U+02230	$\oiiint_{0}^{1}$	`\oiiint`	TRIPLE CONTOUR INTEGRAL OPERATOR
U+02231	$\intclockwise_{0}^{1}$	`\intclockwise`	CLOCKWISE INTEGRAL
U+02232	$\varointclockwise_{0}^{1}$	`\varointclockwise`	CONTOUR INTEGRAL, CLOCKWISE
U+02233	$\ointctrclockwise_{0}^{1}$	`\ointctrclockwise`	CONTOUR INTEGRAL, ANTICLOCKWISE
U+022C0	$\bigwedge_{0}^{1}$	`\bigwedge`	LOGICAL OR OPERATOR
U+022C1	$\bigvee_{0}^{1}$	`\bigvee`	LOGICAL AND OPERATOR
U+022C2	$\bigcap_{0}^{1}$	`\bigcap`	INTERSECTION OPERATOR
U+022C3	$\bigcup_{0}^{1}$	`\bigcup`	UNION OPERATOR
U+027D5	$\leftouterjoin_{0}^{1}$	`\leftouterjoin`	LEFT OUTER JOIN
U+027D6	$\rightouterjoin_{0}^{1}$	`\rightouterjoin`	RIGHT OUTER JOIN
U+027D7	$\fullouterjoin_{0}^{1}$	`\fullouterjoin`	FULL OUTER JOIN
U+027D8	$\bigbot_{0}^{1}$	`\bigbot`	LARGE UP TACK
U+027D9	$\bigtop_{0}^{1}$	`\bigtop`	LARGE DOWN TACK
U+029F8	$\xsol_{0}^{1}$	`\xsol`	BIG SOLIDUS
U+029F9	$\xbsol_{0}^{1}$	`\xbsol`	BIG REVERSE SOLIDUS
U+02A00	$\bigodot_{0}^{1}$	`\bigodot`	N-ARY CIRCLED DOT OPERATOR

U+02A01	$⨁_{0}^{1}$	`\bigoplus`	N-ARY CIRCLED PLUS OPERATOR
U+02A02	$⨂_{0}^{1}$	`\bigotimes`	N-ARY CIRCLED TIMES OPERATOR
U+02A03	$⨃_{0}^{1}$	`\bigcupdot`	N-ARY UNION OPERATOR WITH DOT
U+02A04	$⨄_{0}^{1}$	`\biguplus`	N-ARY UNION OPERATOR WITH PLUS
U+02A05	$⨅_{0}^{1}$	`\bigsqcap`	N-ARY SQUARE INTERSECTION OPERATOR
U+02A06	$⨆_{0}^{1}$	`\bigsqcup`	N-ARY SQUARE UNION OPERATOR
U+02A07	$⨇_{0}^{1}$	`\conjquant`	TWO LOGICAL AND OPERATOR
U+02A08	$⨈_{0}^{1}$	`\disjquant`	TWO LOGICAL OR OPERATOR
U+02A09	$⨉_{0}^{1}$	`\bigtimes`	N-ARY TIMES OPERATOR
U+02A0B	$⨋_{0}^{1}$	`\sumint`	SUMMATION WITH INTEGRAL
U+02A0C	$⨌_{0}^{1}$	`\iiiint`	QUADRUPLE INTEGRAL OPERATOR
U+02A0D	$⨍_{0}^{1}$	`\intbar`	FINITE PART INTEGRAL
U+02A0E	$⨎_{0}^{1}$	`\intBar`	INTEGRAL WITH DOUBLE STROKE
U+02A0F	$⨏_{0}^{1}$	`\fint`	INTEGRAL AVERAGE WITH SLASH
U+02A10	$⨐_{0}^{1}$	`\cirfnint`	CIRCULATION FUNCTION
U+02A11	$⨑_{0}^{1}$	`\awint`	ANTICLOCKWISE INTEGRATION
U+02A12	$⨒_{0}^{1}$	`\rppolint`	LINE INTEGRATION WITH RECTANGULAR PATH AROUND POLE
U+02A13	$⨓_{0}^{1}$	`\scpolint`	LINE INTEGRATION WITH SEMICIRCULAR PATH AROUND POLE
U+02A14	$⨔_{0}^{1}$	`\npolint`	LINE INTEGRATION NOT INCLUDING THE POLE
U+02A15	$⨕_{0}^{1}$	`\pointint`	INTEGRAL AROUND A POINT OPERATOR
U+02A16	$⨖_{0}^{1}$	`\sqint`	QUATERNION INTEGRAL OPERATOR
U+02A17	$⨗_{0}^{1}$	`\intlarhk`	INTEGRAL WITH LEFTWARDS ARROW WITH HOOK
U+02A18	$⨘_{0}^{1}$	`\intx`	INTEGRAL WITH TIMES SIGN
U+02A19	$⨙_{0}^{1}$	`\intcap`	INTEGRAL WITH INTERSECTION
U+02A1A	$⨚_{0}^{1}$	`\intcup`	INTEGRAL WITH UNION
U+02A1B	$⨛_{0}^{1}$	`\upint`	INTEGRAL WITH OVERBAR
U+02A1C	$⨜_{0}^{1}$	`\lowint`	INTEGRAL WITH UNDERBAR
U+02A1D	$⨝_{0}^{1}$	`\Join`	JOIN

U+02A1E	1 ⨞ 0	`\bigtriangleleft`	LARGE LEFT TRIANGLE OPERATOR
U+02A1F	1 ⨟ 0	`\zcmp`	Z NOTATION SCHEMA COMPOSITION
U+02A20	1 ⨠ 0	`\zpipe`	Z NOTATION SCHEMA PIPING
U+02A21	1 ⨡ 0	`\zproject`	Z NOTATION SCHEMA PROJECTION
U+02AFC	1 0	`\biginterleave`	LARGE TRIPLE VERTICAL BAR OPERATOR
U+02AFF	1 0	`\bigtalloblong`	N-ARY WHITE VERTICAL BAR

`\l_um_nolimits_tl` This macro is a sequence containing those maths operators that require a `\nolimits` suffix. This list is used when processing `unicode-math-table.tex` to define such commands automatically (see the macro `\um_set_mathsymbol:nNNn`). I've chosen essentially just the operators that look like integrals; hopefully a better mathematician can help me out here. I've a feeling that it's more useful *not* to include the multiple integrals such as ⨌, but that might be a matter of preference.

```
\tl_new:Nn \l_um_nolimits_tl {
  \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.

```
\DeclareDocumentCommand \addnolimits {m} {
  \tl_put_right:Nn \l_um_nolimits_tl {#1}
}
```

`\removenolimits` Can this macro be given a better name? It removes an item from the nolimits list.

```
\DeclareDocumentCommand \removenolimits {m} {
  \tl_remove_all_in:Nn \l_um_nolimits_tl {#1}
}
```

7.5 Radicals

The radical for square root is organised in `\um_set_mathsymbol:nNNn` on page **??**. I think it's the only radical ever. (Actually, there is also `\cuberoot` and `\fourthroot`, but they don't seem to behave as proper radicals.)

Also, what about right-to-left square roots?

`\um@radicals` We organise radicals in the same way as nolimits-operators; that is, in a comma-list.

```
\tl_new:Nn \l_um_radicals_tl {\sqrt}
```

$$\sqrt[2]{1+\sqrt[3]{1+x}}$$

```
\setmathfont{Cambria Math}
\[ \sqrt[2]{1+\sqrt[3]{1+x}} \]
```

7.6 Delimiters

`\left` We redefine the primitive to be preceded by `\mathopen`; this gives much better spacing in cases such as `\sin\left`.... Courtesy of Frank Mittelbach:

□□□□://□□□.□□□□□-□□□□□□□.□□□/□□□-□□□/□□□□□□□2□□□□?□□=□□□□□/3853&□□□□□□□□/3754

```
\let\left@primitive\left
\def\left{\mathopen{}\left@primitive}
```

No re-definition is made for `\right` because it's not necessary.

Here are all `\mathopen` characters:

USV	Ex.	Macro	Description
U+00028	(	`\lparen`	LEFT PARENTHESIS
U+0005B	[	`\lbrack`	LEFT SQUARE BRACKET
U+0007B	{	`\lbrace`	LEFT CURLY BRACKET
U+0221A	√	`\sqrt`	RADICAL
U+0221B	∛	`\cuberoot`	CUBE ROOT
U+0221C	∜	`\fourthroot`	FOURTH ROOT
U+02308	⌈	`\lceil`	LEFT CEILING
U+0230A	⌊	`\lfloor`	LEFT FLOOR
U+0231C	⌜	`\ulcorner`	UPPER LEFT CORNER
U+0231E	⌞	`\llcorner`	LOWER LEFT CORNER
U+02772		`\lbrbrak`	LIGHT LEFT TORTOISE SHELL BRACKET ORNAMENT
U+027C5	⟅	`\lbag`	LEFT S-SHAPED BAG DELIMITER
U+027CC	⟌	`\longdivision`	LONG DIVISION
U+027E6	⟦	`\lBrack`	MATHEMATICAL LEFT WHITE SQUARE BRACKET
U+027E8	⟨	`\langle`	MATHEMATICAL LEFT ANGLE BRACKET
U+027EA	⟪	`\lAngle`	MATHEMATICAL LEFT DOUBLE ANGLE BRACKET
U+027EC		`\Lbrbrak`	MATHEMATICAL LEFT WHITE TORTOISE SHELL BRACKET

U+02983	⦃	\lBrace	LEFT WHITE CURLY BRACKET
U+02985	⦅	\lParen	LEFT WHITE PARENTHESIS
U+02987	⦇	\llparenthesis	Z NOTATION LEFT IMAGE BRACKET
U+02989	⦉	\llangle	Z NOTATION LEFT BINDING BRACKET
U+0298B	⦋	\lbrackubar	LEFT SQUARE BRACKET WITH UNDERBAR
U+0298D	⦍	\lbrackultick	LEFT SQUARE BRACKET WITH TICK IN TOP CORNER
U+0298F	⦏	\lbrackltick	LEFT SQUARE BRACKET WITH TICK IN BOTTOM CORNER
U+02991	⦑	\langledot	LEFT ANGLE BRACKET WITH DOT
U+02993	⦓	\lparenless	LEFT ARC LESS-THAN BRACKET
U+02995	⦕	\Lparengtr	DOUBLE LEFT ARC GREATER-THAN BRACKET
U+02997	⦗	\lblkbrbrak	LEFT BLACK TORTOISE SHELL BRACKET
U+029D8	⧘	\lvzigzag	LEFT WIGGLY FENCE
U+029DA	⧚	\Lvzigzag	LEFT DOUBLE WIGGLY FENCE
U+029FC	⧼	\lcurvyangle	LEFT POINTING CURVED ANGLE BRACKET
U+03014		\lbrbrak	LEFT BROKEN BRACKET
U+03018		\Lbrbrak	LEFT WHITE TORTOISE SHELL BRACKET

And \mathclose:

USV	Ex.	Macro	Description
U+00029	)	\rparen	RIGHT PARENTHESIS
U+0005D	]	\rbrack	RIGHT SQUARE BRACKET
U+0007D	}	\rbrace	RIGHT CURLY BRACKET
U+02309	⌉	\rceil	RIGHT CEILING
U+0230B	⌋	\rfloor	RIGHT FLOOR
U+0231D	⌝	\urcorner	UPPER RIGHT CORNER
U+0231F	⌟	\lrcorner	LOWER RIGHT CORNER
U+02773		\rbrbrak	LIGHT RIGHT TORTOISE SHELL BRACKET ORNAMENT
U+027C6	⟆	\rbag	RIGHT S-SHAPED BAG DELIMITER
U+027E7	⟧	\rBrack	MATHEMATICAL RIGHT WHITE SQUARE BRACKET
U+027E9	⟩	\rangle	MATHEMATICAL RIGHT ANGLE BRACKET
U+027EB	⟫	\rAngle	MATHEMATICAL RIGHT DOUBLE ANGLE BRACKET
U+027ED		\Rbrbrak	MATHEMATICAL RIGHT WHITE TORTOISE SHELL BRACKET
U+02984	⦄	\rBrace	RIGHT WHITE CURLY BRACKET
U+02986	⦆	\rParen	RIGHT WHITE PARENTHESIS
U+02988	⦈	\rrparenthesis	Z NOTATION RIGHT IMAGE BRACKET
U+0298A	⦊	\rrangle	Z NOTATION RIGHT BINDING BRACKET
U+0298C	⦌	\rbrackubar	RIGHT SQUARE BRACKET WITH UNDERBAR

U+0298E	⦎	`\rbracklrtick`	RIGHT SQUARE BRACKET WITH TICK IN BOTTOM CORNER
U+02990	⦐	`\rbrackurtick`	RIGHT SQUARE BRACKET WITH TICK IN TOP CORNER
U+02992	⦒	`\rangledot`	RIGHT ANGLE BRACKET WITH DOT
U+02994	⦔	`\rparengtr`	RIGHT ARC GREATER-THAN BRACKET
U+02996	⦖	`\Rparenless`	DOUBLE RIGHT ARC LESS-THAN BRACKET
U+02998	⦘	`\rblkbrbrak`	RIGHT BLACK TORTOISE SHELL BRACKET
U+029D9	⧙	`\rvzigzag`	RIGHT WIGGLY FENCE
U+029DB	⧛	`\Rvzigzag`	RIGHT DOUBLE WIGGLY FENCE
U+029FD	⧽	`\rcurvyangle`	RIGHT POINTING CURVED ANGLE BRACKET
U+03015		`\rbrbrak`	RIGHT BROKEN BRACKET
U+03019		`\Rbrbrak`	RIGHT WHITE TORTOISE SHELL BRACKET

7.7 Maths accents

Maths accents should just work *if they are available in the font*.

USV	Ex.	Macro	Description
U+00300	x̀	`\grave`	GRAVE ACCENT
U+00301	x́	`\acute`	ACUTE ACCENT
U+00302	x̂	`\hat`	CIRCUMFLEX ACCENT
U+00303	x̃	`\tilde`	TILDE
U+00304	x̄	`\bar`	MACRON
U+00305	x̅	`\overbar`	OVERBAR EMBELLISHMENT
U+00306	x̆	`\breve`	BREVE
U+00307	ẋ	`\dot`	DOT ABOVE
U+00308	ẍ	`\ddot`	DIERESIS
U+00309	x̉	`\ovhook`	COMBINING HOOK ABOVE
U+0030A	x̊	`\ocirc`	RING
U+0030C	x̌	`\check`	CARON
U+00310	x̐	`\candra`	CANDRABINDU (NON-SPACING)
U+00312	x̒	`\oturnedcomma`	COMBINING TURNED COMMA ABOVE
U+00313	x̓	`\osmooth`	GREEK PSILI (SMOOTH BREATHING) (NON-SPACING)
U+00314	x̔	`\orough`	GREEK DASIA (ROUGH BREATHING) (NON-SPACING)
U+00315	x̕	`\ocommatopright`	COMBINING COMMA ABOVE RIGHT
U+0031A	x̚	`\droang`	LEFT ANGLE ABOVE (NON-SPACING)
U+00330	x̰	`\wideutilde`	UNDER TILDE ACCENT (MULTIPLE CHARACTERS AND NON-SPACING)
U+00331	x̱	`\underbar`	COMBINING MACRON BELOW
U+00338	x̸	`\not`	COMBINING LONG SOLIDUS OVERLAY

U+020D0	x⃐	\leftharpoonaccent	COMBINING LEFT HARPOON ABOVE
U+020D1	x⃑	\rightharpoonaccent	COMBINING RIGHT HARPOON ABOVE
U+020D2	x⃒	\vertoverlay	COMBINING LONG VERTICAL LINE OVERLAY
U+020D6	x⃖	\overleftarrow	COMBINING LEFT ARROW ABOVE
U+020D7	x⃗	\vec	COMBINING RIGHT ARROW ABOVE
U+020DB	x⃛	\dddot	COMBINING THREE DOTS ABOVE
U+020DC	x⃜	\ddddot	COMBINING FOUR DOTS ABOVE
U+020E1	x⃡	\overleftrightarrow	COMBINING LEFT RIGHT ARROW ABOVE
U+020E7	⍰	\annuity	COMBINING ANNUITY SYMBOL
U+020E8	x⃨	\threeunderdot	COMBINING TRIPLE UNDERDOT
U+020E9	x⃩	\widebridgeabove	COMBINING WIDE BRIDGE ABOVE
U+020EC	⍰	\underrightharpoondown	COMBINING RIGHTWARDS HARPOON WITH BARB DOWNWARDS
U+020ED	⍰	\underleftharpoondown	COMBINING LEFTWARDS HARPOON WITH BARB DOWNWARDS
U+020EE	⍰	\underleftarrow	COMBINING LEFT ARROW BELOW
U+020EF	⍰	\underrightarrow	COMBINING RIGHT ARROW BELOW
U+020F0	⍰	\asteraccent	COMBINING ASTERISK ABOVE

8 Font features

\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}[]{
    \bool_if:NTF \l_um_fontspec_feature_bool {
      #2
    }{
      \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\setmathfont}
    }
  }
}
```

8.1 OpenType maths font features

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

8.2 Script and scriptscript font options

```
\define@cmdkey[um]{options}[um@]{script-features}{}
\define@cmdkey[um]{options}[um@]{sscript-features}{}
\define@cmdkey[um]{options}[um@]{script-font}{}
\define@cmdkey[um]{options}[um@]{sscript-font}{}
```

8.3 Range processing

The ‘ALL’ branch here is deprecated and happens automatically.

```
\seq_new:N \g_um_mathalph_seq
\seq_new:N \l_um_mathalph_seq
\seq_new:N \l_um_char_range_seq
\define@choicekey+[um]{options}{range}[\@tempa\@tempb]{ALL}{
  \ifcase\@tempb\relax
    \bool_set_true:N \l_um_init_bool
  \fi
}{
  \bool_set_false:N \l_um_init_bool
  \seq_clear:N \l_um_char_range_seq
  \seq_clear:N \l_um_mathalph_seq
  \clist_map_inline:nn {#1} {
    \um_if_mathalph_decl:nTF {##1} {
    \seq_put_right:Nx \l_um_mathalph_seq { {\exp_not:V\l_um_tmpa_tl} {\exp_not:V\l_um_tmpb_tl
    }{
      \seq_put_right:Nn \l_um_char_range_seq {##1}
    }
  }
}
\prg_new_conditional:Nnn \um_if_mathalph_decl:n {TF} {
  \tl_set:Nn \l_um_tmpa_tl {#1}
  \tl_set:Nn \l_um_tmpb_tl {}
  \tl_set:Nn \l_um_tmpc_tl {}
  \tl_if_in:NnT \l_um_tmpa_tl {->} {
    \exp_after:wN \um_split_arrow:w \l_um_tmpa_tl \q_nil
  }
  \tl_if_in:NnT \l_um_tmpa_tl {/} {
    \exp_after:wN \um_split_slash:w \l_um_tmpa_tl \q_nil
  }
  \seq_if_in:NVTF \g_um_mathalph_seq \l_um_tmpa_tl {
    \prg_return_true:
  }{
    \prg_return_false:
  }
}
```

```
\cs_set:Npn \um_split_arrow:w #1->#2 \q_nil {
  \tl_set:Nn \l_um_tmpa_tl {#1}
  \tl_set:Nn \l_um_tmpc_tl {#2}
}
\cs_set:Npn \um_split_slash:w #1/#2 \q_nil {
  \tl_set:Nn \l_um_tmpa_tl {#1}
  \tl_set:Nn \l_um_tmpb_tl {#2}
}
```

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 if any of its arguments are contained in \l_um_char_range_seq. 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 13.

Table 13: Ranges accepted by \um@parse@range.

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

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

```
\newcommand\um@parse@term[4]{
  \seq_map_variable:NNn \l_um_char_range_seq \@ii {
    \unless\ifx\@ii\@empty
      \@tempswafalse
```

Match to either the character macro (\alpha) or the math type (\mathbin):

```
      \expandafter\um@firstchar\expandafter{\@ii}
      \ifx\@tempa\um@backslash
        \expandafter\ifx\@ii#2\relax
          \@tempswatrue
        \else
          \expandafter\ifx\@ii#3\relax
            \@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! It also populates the `\um@char@num@range` macro, which is used when defining `\mathbf` (*etc.*) `\mathchar` remappings.

```
      \if@tempswa
        \ifx\um@char@num@range\@empty
          \g@addto@macro\um@char@num@range{#1}
        \else
          \g@addto@macro\um@char@num@range{,#1}
        \fi
        #4%
      \fi
    \fi
  }
}
\def\um@firstof#1#2\@nil{#1}
\edef\um@backslash{\expandafter\um@firstof\string\string\@nil}
\def\um@firstchar#1{\edef\@tempa{\expandafter\um@firstof\string#1\@nil}}
```

`\um@parse@range` Weird syntax. As shown previously in table 13, 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-{}`

```
  \expandafter\ifx\expandafter\@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	$r \le y$
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	$x \le r \le y$
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
}
```

\um_map_char:nn
\um_map_chars_xxvi:nn
\um_map_chars_xxiii:nn

#1 : Number of iterations
#2 : Starting input char(s)
#3 : Starting output char
Loops through character ranges setting \mathcode.

```
\cs_set:Nn \um_map_chars_range:nnn {
 \clist_map_inline:nn {#2} {
  \prg_stepwise_inline:nnnn {0}{1}{#1} {
   \um_map_char_internal:nn {##1+####1}{#3+####1}
  }
 }
}
\cs_new:Nn \um_map_char_noparse:nn {
 \um_set_mathcode:nnnn
  {\numexpr #1 \relax}{\mathalpha}{\um_symfont_tl}{\numexpr #2 \relax}
}
\cs_new:Nn \um_map_char_parse:nn {
 \um@parse@term {#1} {\@nil} {\mathalpha} {
  \um_map_char_noparse:nn {#1}{#2}
 }
}
```

```
\cs_set:Nn \um_map_chars_xxvi:nn {
  \um_map_chars_range:nnn {25}{#1}{#2}
}
\cs_set:Nn \um_map_chars_xxiii:nn {
  \um_map_chars_range:nnn {24}{#1}{#2}
}
\cs_set:Nn \um_map_chars_Latin:nn {
  \clist_map_inline:nn {#1} {
    \um_map_chars_xxvi:cc {g_um_ ##1 _Latin_usv}{g_um_ #2 _Latin_usv}
  }
}
\cs_set:Nn \um_map_chars_latin:nn {
  \clist_map_inline:nn {#1} {
    \um_map_chars_xxvi:cc {g_um_ ##1 _latin_usv}{g_um_ #2 _latin_usv}
  }
}
\cs_set:Nn \um_map_chars_greek:nn {
  \clist_map_inline:nn {#1} {
    \um_map_chars_xxiii:cc {g_um_ ##1 _greek_usv}{g_um_ #2 _greek_usv}
    \um_map_char:cc {g_um_ ##1 _varepsilon_usv}{g_um_ #2 _varepsilon_usv}
    \um_map_char:cc {g_um_ ##1 _vartheta_usv  }{g_um_ #2 _vartheta_usv  }
    \um_map_char:cc {g_um_ ##1 _varkappa_usv  }{g_um_ #2 _varkappa_usv  }
    \um_map_char:cc {g_um_ ##1 _varphi_usv    }{g_um_ #2 _varphi_usv    }
    \um_map_char:cc {g_um_ ##1 _varrho_usv    }{g_um_ #2 _varrho_usv    }
    \um_map_char:cc {g_um_ ##1 _varpi_usv     }{g_um_ #2 _varpi_usv     }
  }
}
\cs_set:Nn \um_map_chars_Greek:nn {
  \clist_map_inline:nn {#1} {
    \um_map_chars_xxiii:cc {g_um_ ##1 _Greek_usv}{g_um_ #2 _Greek_usv}
    \um_map_char:cc {g_um_ ##1 _varTheta_usv}{g_um_ #2 _varTheta_usv}
  }
}
\cs_set:Nn \um_map_chars_numbers:nn {
  \um_map_chars_range:nnn {9}{#1}{#2}
}
\cs_set:Nn \um_map_char:nn {
  \um_map_chars_range:nnn {0}{#1}{#2}
}
\cs_generate_variant:Nn \um_map_char:nn {cc}
\cs_generate_variant:Nn \um_map_chars_xxiii:nn {cc}
\cs_generate_variant:Nn \um_map_chars_xxvi:nn {cc}
```

`\um_set_mathalphabet_char:Nnn`

#1 : Maths alphabet
#2 : Input char(s)
#3 : Output char

Loops through character ranges setting \mathcode.

```
\cs_set:Npn \exp_args:Nnff {\::n\::f\::f\:::}
\cs_new:Nn \um_set_mathalphabet_char:Nnn {
  \clist_map_variable:nNn {#2} \l_um_input_num {
    \exp_args:Nnff \um_mathmap:Nnn {#1}
      {\number\numexpr\l_um_input_num\relax} {\number\numexpr#3\relax}
  }
}
```

\um_set_mathalph_range:Nnn

[⟨*Number of iterations*⟩] #1 : Maths alphabet
#2 : Starting input char(s)
#3 : Starting output char
Loops through character ranges setting \mathcode.

```
\cs_new:Nn \um_set_mathalph_range:nNnn {
  \clist_map_variable:nNn {#3} \l_um_input_num {
    \errorcontextlines=999
    \prg_stepwise_variable:nnnNn {0}{1}{#1} \l_um_inc_num {
      \exp_args:Nnff \um_mathmap:Nnn {#2}
        {\number\numexpr \l_um_inc_num + \l_um_input_num \relax}
        {\number\numexpr \l_um_inc_num + #4 \relax}
    }
  }
}
\cs_new:Nn \um_set_mathalphabet_x:Nnn {
  \um_set_mathalph_range:nNnn {9}{#1}{#2}{#3}
}
\cs_new:Nn \um_set_mathalphabet_xxvi:Nnn {
  \um_set_mathalph_range:nNnn {25}{#1}{#2}{#3}
}
\cs_new:Nn \um_set_mathalphabet_xxiii:Nnn {
  \um_set_mathalph_range:nNnn {24}{#1}{#2}{#3}
}

\cs_new:Nn \um_set_mathalphabet_pos:Nnnn {
  \clist_map_inline:nn {#3} {
    \um_set_mathalphabet_char:Ncc  #1 {g_um_##1_#2_usv}{g_um_#4_#2_usv}
  }
}
\cs_new:Nn \um_set_mathalphabet_numbers:Nnn {
  \clist_map_inline:nn {#2} {
    \um_set_mathalphabet_x:Ncc  #1 {g_um_##1_num_usv}{g_um_#3_num_usv}
  }
}
\cs_new:Nn \um_set_mathalphabet_Latin:Nnn {
  \clist_map_inline:nn {#2} {
   \um_set_mathalphabet_xxvi:Ncc #1 {g_um_##1_Latin_usv}{g_um_#3_Latin_usv}
  }
}
\cs_new:Nn \um_set_mathalphabet_latin:Nnn {
  \clist_map_inline:nn {#2} {
   \um_set_mathalphabet_xxvi:Ncc #1 {g_um_##1_latin_usv}{g_um_#3_latin_usv}
    \um_set_mathalphabet_char:Ncc #1 {g_um_##1_h_usv}    {g_um_#3_h_usv}
  }
}
\cs_new:Nn \um_set_mathalphabet_Greek:Nnn {
  \clist_map_inline:nn {#2} {
   \um_set_mathalphabet_xxiii:Ncc #1 {g_um_##1_Greek_usv}  {g_um_#3_Greek_usv}
   \um_set_mathalphabet_char:Ncc  #1 {g_um_##1_varTheta_usv}{g_um_#3_varTheta_usv}
  }
}
\cs_new:Nn \um_set_mathalphabet_greek:Nnn {
  \clist_map_inline:nn {#2} {
   \um_set_mathalphabet_xxiii:Ncc #1 {g_um_##1_greek_usv}    {g_um_#3_greek_usv}
   \um_set_mathalphabet_char:Ncc  #1 {g_um_##1_varepsilon_usv}{g_um_#3_varepsilon_usv}
   \um_set_mathalphabet_char:Ncc  #1 {g_um_##1_vartheta_usv} {g_um_#3_vartheta_usv}
   \um_set_mathalphabet_char:Ncc  #1 {g_um_##1_varkappa_usv} {g_um_#3_varkappa_usv}
   \um_set_mathalphabet_char:Ncc  #1 {g_um_##1_varphi_usv}   {g_um_#3_varphi_usv}
   \um_set_mathalphabet_char:Ncc  #1 {g_um_##1_varrho_usv}   {g_um_#3_varrho_usv}
   \um_set_mathalphabet_char:Ncc  #1 {g_um_##1_varpi_usv}    {g_um_#3_varpi_usv}
  }
}
\cs_generate_variant:Nn \um_set_mathalphabet_char:Nnn {Ncc}
\cs_generate_variant:Nn \um_set_mathalphabet_xxiii:Nnn {Ncc}
\cs_generate_variant:Nn \um_set_mathalphabet_xxvi:Nnn {Ncc}
\cs_generate_variant:Nn \um_set_mathalphabet_x:Nnn {Ncc}
```

8.4 Resolving Greek symbol name control sequences

\um_resolve_greek:

This macro defines \Alpha…\omega as their corresponding unicode (mathematical italic) character. Remember that the mapping to upright or italic happens with the mathcode definitions, whereas these macros just stand for the literal unicode characters.

```
\AtBeginDocument{\um_resolve_greek:}
\cs_new:Nn \um_resolve_greek: {
  \clist_map_inline:nn {
    Alpha,Beta,Gamma,Delta,Epsilon,Zeta,Eta,Theta,Iota,Kappa,Lambda,
    alpha,beta,gamma,delta,        zeta,eta,theta,ioto,kappa,lambda,
    Mu,Nu,Xi,Omicron,Pi,Rho,Sigma,Tau,Upsilon,Phi,Chi,Psi,Omega,
    mu,nu,xi,omicron,pi,rho,sigma,tau,upsilon,    chi,psi,omega,
    varTheta,
    varsigma,vartheta,varkappa,varrho,varpi
  }{
    \tl_set:cx {##1} { \exp_not:c { mit ##1 } }
  }
  \tl_set:Nn \epsilon {
    \bool_if:NTF \g_um_texgreek_bool \mitvarepsilon \mitepsilon
  }
  \tl_set:Nn \phi {
    \bool_if:NTF \g_um_texgreek_bool \mitvarphi \mitphi
  }
  \tl_set:Nn \varepsilon {
    \bool_if:NTF \g_um_texgreek_bool \mitepsilon \mitvarepsilon
  }
  \tl_set:Nn \varphi {
    \bool_if:NTF \g_um_texgreek_bool \mitphi \mitvarphi
  }
}
```

9 Maths alphabets mapping definitions

Algorithm for setting alphabet fonts. By default, when `range` is empty, we are in *implicit* mode. If `range` contains the name of the math alphabet, we are in *explicit* mode and do things slightly differently.

Implicit mode:

- Try and set all of the alphabet shapes.
- Check for the first glyph of each alphabet to detect if the font supports each alphabet shape.
- For alphabets that do exist, overwrite whatever's already there.
- For alphabets that are not supported, *do nothing*. (This includes leaving the old alphabet definition in place.)

Explicit mode:

- Only set the alphabets specified.
- Check for the first glyph of the alphabet to detect if the font contains the alphabet shape in the unicode math plane.
- For unicode math alphabets, overwrite whatever's already there.
- Otherwise, use the ASCII letters instead.

9.0.1 Macros

This is every math alphabet known to unicode-math:

\g_um_mathalph_seq

```
\seq_clear:N \g_um_mathalph_seq
\tl_map_inline:nn {
  \mathup\mathit
  \mathbb\mathscr\mathfrak\mathtt
  \mathsf\mathsfup\mathsfit
  \mathbf\mathbfup\mathbfit
  \mathbfscr\mathbffrak
  \mathbfsf\mathbfsfup\mathbfsfit
}{
  \seq_put_right:Nn \g_um_mathalph_seq {#1}
}
```

\um_setup_alphabets:

```

\tl_new:Nn \g_um_mathup_alph_clist {latin,Latin,greek,Greek,num}
\tl_new:Nn \g_um_mathit_alph_clist {latin,Latin,greek,Greek}
\tl_new:Nn \g_um_mathscr_alph_clist    {latin,Latin}
\tl_new:Nn \g_um_mathfrak_alph_clist   {latin,Latin}
\tl_new:Nn \g_um_mathbfscr_alph_clist  {latin,Latin}
\tl_new:Nn \g_um_mathbffrak_alph_clist {latin,Latin}
\tl_new:Nn \g_um_mathbb_alph_clist     {latin,Latin,num}
\tl_new:Nn \g_um_mathtt_alph_clist     {latin,Latin,num}
\tl_new:Nn \g_um_mathsf_alph_clist     {latin,Latin,num}
\tl_new:Nn \g_um_mathsfup_alph_clist   {latin,Latin,num}
\tl_new:Nn \g_um_mathsfit_alph_clist   {latin,Latin}
\tl_new:Nn \g_um_mathbf_alph_clist     {latin,Latin,greek,Greek,num}
\tl_new:Nn \g_um_mathbfup_alph_clist   {latin,Latin,greek,Greek,num}
\tl_new:Nn \g_um_mathbfit_alph_clist   {latin,Latin,greek,Greek,num}
\tl_new:Nn \g_um_mathbfsf_alph_clist   {latin,Latin,greek,Greek,num}
\tl_new:Nn \g_um_mathbfsfup_alph_clist {latin,Latin,greek,Greek,num}
\tl_new:Nn \g_um_mathbfsfit_alph_clist {latin,Latin,greek,Greek}

\tl_new:Nn \g_um_mathup_latin_usv {`\a-`\z}
\tl_new:Nn \g_um_mathup_Latin_usv {`\A-`\Z}
\tl_new:Nn \g_um_mathup_greek_usv {"3B1-"3C9,"3F5,"3D1,"3F0,"3D5,"3F1,"3D6,"3DD}
\tl_new:Nn \g_um_mathup_Greek_usv {"391-"3A9,"3F4,"3DC}
\tl_new:Nn \g_um_mathup_num_usv   {`\0-`\9}

\tl_new:Nn \g_um_mathit_latin_usv {"1D44E-"1D467,\g_um_it_h_usv}
\tl_new:Nn \g_um_mathit_Latin_usv {"1D434-"1D44C}
\tl_new:Nn \g_um_mathit_greek_usv {"1D6FC-"1D714,"1D716-1D71B}
\tl_new:Nn \g_um_mathit_Greek_usv {"1D6E2-"1D6FA}
```

```
\seq_new:N \l_um_missing_alph_seq
\cs_new:Nn \um_setup_alphabets: {
  \seq_clear:N \l_um_missing_alph_seq
  \seq_if_empty:NTF \l_um_mathalph_seq {
    \um_setup_math_alphabet:NV \mathup     \g_um_mathup_alph_clist
    \um_setup_math_alphabet:NV \mathit     \g_um_mathit_alph_clist
    \um_setup_math_alphabet:NV \mathbb     \g_um_mathbb_alph_clist
    \um_setup_math_alphabet:NV \mathscr    \g_um_mathscr_alph_clist
    \um_setup_math_alphabet:NV \mathfrak   \g_um_mathfrak_alph_clist
    \um_setup_math_alphabet:NV \mathsf     \g_um_mathsf_alph_clist
    \um_setup_math_alphabet:NV \mathsfup   \g_um_mathsfup_alph_clist
    \um_setup_math_alphabet:NV \mathsfit   \g_um_mathsfit_alph_clist
    \um_setup_math_alphabet:NV \mathtt     \g_um_mathtt_alph_clist
    \um_setup_math_alphabet:NV \mathbf     \g_um_mathbf_alph_clist
    \um_setup_math_alphabet:NV \mathbfup   \g_um_mathbfup_alph_clist
    \um_setup_math_alphabet:NV \mathbfit   \g_um_mathbfit_alph_clist
    \um_setup_math_alphabet:NV \mathbfscr  \g_um_mathbfscr_alph_clist
    \um_setup_math_alphabet:NV \mathbffrak \g_um_mathbffrak_alph_clist
    \um_setup_math_alphabet:NV \mathbfsf   \g_um_mathbfsf_alph_clist
    \um_setup_math_alphabet:NV \mathbfsfup \g_um_mathbfsfup_alph_clist
    \um_setup_math_alphabet:NV \mathbfsfit \g_um_mathbfsfit_alph_clist
    \um_setup_math_mapping:n   {up    }
    \um_setup_math_mapping:n   {it    }
    \um_setup_math_mapping:n   {bb    }
    \um_maybe_init_alphabet:n  {bbit  }
    \um_setup_math_mapping:n   {bbit  }
    \um_setup_math_mapping:n   {bfup  }
    \um_setup_math_mapping:n   {bfit  }
    \um_setup_math_mapping:n   {bfsfup}
    \um_setup_math_mapping:n   {bfsfit}
    \seq_if_empty:NF \l_um_missing_alph_seq {
      \typeout{
        Package~unicode-math~Warning:~
        missing~math~alphabets~in~font~ \fontname\l_um_font
      }
      \seq_map_inline:Nn \l_um_missing_alph_seq {
        \typeout{\space\space\space\space##1}
      }
    }
  }{
    \cs_set_eq:NN \um_mathmap:Nnn \um_mathmap_noparse:Nnn
    \seq_map_inline:Nn \l_um_mathalph_seq {
      \tl_set:No \l_um_tmpa_tl { \use_i:nnn   ##1 }
      \tl_set:No \l_um_tmpb_tl { \use_ii:nnn  ##1 }
      \tl_set:No \l_um_tmpc_tl { \use_iii:nnn ##1 }
      \tl_if_empty:NF \l_um_tmpc_tl {
     \PackageWarning{unicode-math}{alphabet~remapping~not~yet~implemented}
      }
      \tl_if_empty:NT \l_um_tmpb_tl {
        \cs_set_eq:NN \um_maybe_init_alphabet:n \um_init_alphabet:n
     \tl_set:Nv \l_um_tmpb_tl { g_um_ \exp_after:wN \cs_to_str:N \l_um_tmpa_tl _alph_clist }
      }
      \um_setup_math_alphabet:VV \l_um_tmpa_tl \l_um_tmpb_tl
    }
  }
}
```

`\um_setup_math_alphabet:Nn`

`#1 :` Math font family name (e.g., `\mathbb`)
`#2 :` Math alphabets, comma separated of {latin,Latin,greek,Greek,num}

First check that at least one of the alphabets for the font shape is defined, and then loop through them defining the individual ranges.

```
\cs_new:Nn \um_setup_math_alphabet:Nn {
  \tl_set:Nx \l_um_tmpa_tl {\cs_to_str:N #1}
  \tl_set:Nx \l_um_tmpb_tl {\exp_after:wN \use_none:nnnn \l_um_tmpa_tl}
  \clist_map_inline:nn {#2} {
    \um_glyph_if_exist:cT {g_um_ \l_um_tmpb_tl _##1_usv}{
      \exp_args:NV \um_maybe_init_alphabet:n \l_um_tmpb_tl
      \clist_map_break:
    }
  }
  \clist_map_inline:nn {#2} {
    \um_glyph_if_exist:cTF {g_um_ \l_um_tmpb_tl _##1_usv}{
      \use:c {um_config_ \l_um_tmpa_tl _##1:}
    }{
      \seq_put_right:Nx \l_um_missing_alph_seq {
        \@backslashchar
        \l_um_tmpa_tl\space(\tl_use:c{g_um_math_alphabet_name_##1_tl})
      }
    }
  }
}
\cs_generate_variant:Nn \um_setup_math_alphabet:Nn {NV,VV}

\tl_set:Nn \g_um_math_alphabet_name_latin_tl {Latin,~lowercase}
\tl_set:Nn \g_um_math_alphabet_name_Latin_tl {Latin,~uppercase}
\tl_set:Nn \g_um_math_alphabet_name_greek_tl {Greek,~lowercase}
\tl_set:Nn \g_um_math_alphabet_name_Greek_tl {Greek,~uppercase}
\tl_set:Nn \g_um_math_alphabet_name_num_tl   {Numerals}
\cs_new:Nn \um_setup_math_mapping:n {
  \cs_if_exist:cT {um_setup_math#1:} {
    \use:c {um_config_math#1_misc:}
  }
}
```

```
\cs_set:Nn \um_init_alphabet:n {
  \wlog{unicode-math:~Initialiasing~\@backslashchar math#1}
  \um_prepare_alph:n {#1}
  \cs_set_eq:cN {um_setup_math#1:} \prg_do_nothing:
}
```

`\um_glyph_if_exist:nTF` : TODO: Generalise for arbitrary fonts! `\um@font` is not always the one used for a specific glyph!!

```
\prg_new_conditional:Nnn \um_glyph_if_exist:n {p,TF,T,F} {
 \etex_iffontchar:D \l_um_font #1 \scan_stop: \prg_return_true: \else: \prg_return_false: \fi:
}
\cs_generate_variant:Nn \um_glyph_if_exist_p:n {c}
\cs_generate_variant:Nn \um_glyph_if_exist:nTF {c}
\cs_generate_variant:Nn \um_glyph_if_exist:nT  {c}
\cs_generate_variant:Nn \um_glyph_if_exist:nF  {c}
```

`\um_prepare_alph:n` If `\mathXY` hasn't been (re-)declared yet, then define it in terms of unicode-math defintions. Use `\bgroup`/`\egroup` so s'scripts scan the whole thing.

```
\cs_new:Nn \um_prepare_alph:n {
  \cs_if_exist:cF {um_math#1:n} {
    \cs_set:cpn {um_math#1:n} ##1 {
      \use:c {um_setup_math#1:} ##1 \egroup
    }
    \cs_set_protected:cpn {math#1} {
      \bgroup
      \mode_if_math:F {
        \egroup\expandafter
        \non@alpherr\expandafter{\csname math#1\endcsname\space}
      }
      \use:c {um_math#1:n}
    }
  }
}
```

9.1 Alphabets

9.1.1 Upright: `\mathup`

```
\cs_new:Npn \um_config_mathup_num: {
  \um_map_chars_numbers:nn {`\0}{`\0}
  \um_set_mathalphabet_numbers:Nnn \mathup {up}{up}
}
\cs_new:Npn \um_config_mathup_Latin: {
  \bool_if:NTF \g_um_literal_bool {
    \um_map_chars_Latin:nn {up} {up}
  }{
    \bool_if:NT \g_um_upLatin_bool {
      \um_map_chars_Latin:nn {up,it} {up}
    }
  }
  \um_set_mathalphabet_Latin:Nnn \mathup {up,it}{up}
}
\cs_new:Npn \um_config_mathup_latin: {
  \bool_if:NTF \g_um_literal_bool {
    \um_map_chars_latin:nn {up} {up}
  }{
    \bool_if:NT \g_um_uplatin_bool {
      \um_map_chars_latin:nn {up,it} {up}
    \um_map_char:nn {\g_um_up_h_usv,\g_um_it_h_usv}{\g_um_up_h_usv}% KEEP
    }
  }
  \um_set_mathalphabet_latin:Nnn \mathup {up,it}{up}
}
\cs_new:Npn \um_config_mathup_Greek: {
  \bool_if:NTF \g_um_literal_bool {
    \um_map_chars_Greek:nn {up}{up}
  }{
    \bool_if:NT \g_um_upGreek_bool {
      \um_map_chars_Greek:nn {up,it}{up}
    }
  }
  \um_set_mathalphabet_Greek:Nnn \mathup {up,it}{up}
}
\cs_new:Npn \um_config_mathup_greek: {
  \bool_if:NTF \g_um_literal_bool {
    \um_map_chars_greek:nn {up} {up}
  }{
    \bool_if:NT \g_um_upgreek_bool {
      \um_map_chars_greek:nn {up,it} {up}
    }
  }
  \um_set_mathalphabet_greek:Nnn \mathup {up,it} {up}
}
\cs_new:Npn \um_config_mathup_misc: {
  \um_set_mathalphabet_pos:Nnnn \mathup {partial} {up,it}{up}
  \um_set_mathalphabet_pos:Nnnn \mathup {Nabla}   {up,it}{up}
}
```

9.1.2 Italic: \mathit

```
\cs_new:Npn \um_config_mathit_Latin: {
  \bool_if:NTF \g_um_literal_bool {
    \um_map_chars_Latin:nn {it} {it}
  }{
    \bool_if:NF \g_um_upLatin_bool {
      \um_map_chars_Latin:nn {up,it} {it}
    }
  }
  \um_set_mathalphabet_Latin:Nnn \mathit {up,it}{it}
}
\cs_new:Npn \um_config_mathit_latin: {
  \bool_if:NTF \g_um_literal_bool {
    \um_map_chars_latin:nn {it} {it}
    \um_map_char:nn {\g_um_it_h_usv}{\g_um_it_h_usv}% KEEP
  }{
    \bool_if:NF \g_um_uplatin_bool {
      \um_map_chars_latin:nn {up,it} {it}
     \um_map_char:nn {\g_um_up_h_usv,\g_um_it_h_usv}{\g_um_it_h_usv}% KEEP
    }
  }
  \um_set_mathalphabet_latin:Nnn \mathit {up,it}{it}
}
\cs_new:Npn \um_config_mathit_Greek: {
  \bool_if:NTF \g_um_literal_bool {
    \um_map_chars_Greek:nn {it}{it}
  }{
    \bool_if:NF \g_um_upGreek_bool {
      \um_map_chars_Greek:nn {up,it}{it}
    }
  }
  \um_set_mathalphabet_Greek:Nnn \mathit {up,it}{it}
}
\cs_new:Npn \um_config_mathit_greek: {
  \bool_if:NTF \g_um_literal_bool {
    \um_map_chars_greek:nn {it} {it}
  }{
    \bool_if:NF \g_um_upgreek_bool {
      \um_map_chars_greek:nn {it,up} {it}
    }
  }
  \um_set_mathalphabet_greek:Nnn \mathit {up,it} {it}
}
\cs_new:Npn \um_config_mathit_misc: {
  \um_set_mathalphabet_pos:Nnnn \mathit {partial} {up,it}{it}
  \um_set_mathalphabet_pos:Nnnn \mathit {Nabla}   {up,it}{it}
}
```

9.1.3 Blackboard or double-struck: \mathbb and \mathbbit

```
\cs_new:Npn \um_config_mathbb_latin: {
  \um_set_mathalphabet_latin:Nnn \mathbb {up,it}{bb}
}
\cs_new:Npn \um_config_mathbb_Latin: {
  \um_set_mathalphabet_Latin:Nnn \mathbb {up,it}{bb}
  \um_set_mathalphabet_char:Nnn{\mathbb}{`\C,"1D436}{"2102}
  \um_set_mathalphabet_char:Nnn{\mathbb}{`\H,"1D43B}{"210D}
  \um_set_mathalphabet_char:Nnn{\mathbb}{`\N,"1D441}{"2115}
  \um_set_mathalphabet_char:Nnn{\mathbb}{`\P,"1D443}{"2119}
  \um_set_mathalphabet_char:Nnn{\mathbb}{`\Q,"1D444}{"211A}
  \um_set_mathalphabet_char:Nnn{\mathbb}{`\R,"1D445}{"211D}
  \um_set_mathalphabet_char:Nnn{\mathbb}{`\Z,"1D44D} {"2124}
}
\cs_new:Npn \um_config_mathbb_num: {
  \um_set_mathalphabet_numbers:Nnn \mathbb {up}{bb}
}
\cs_new:Npn \um_config_mathbb_misc: {
  \um_set_mathalphabet_char:Nnn \mathbb {"03A0,"1D6F1}{"213F} % Pi
  \um_set_mathalphabet_char:Nnn \mathbb {"03C0,"1D70B}{"213C} % pi
  \um_set_mathalphabet_char:Nnn \mathbb {"0393,"1D6E4}{"213E} % Gamma
  \um_set_mathalphabet_char:Nnn \mathbb {"03B3,"1D6FE}{"213D} % gamma
  \um_set_mathalphabet_char:Nnn \mathbb {"2211}{"2140} % summation
}
\cs_new:Npn \um_config_mathbbit_misc: {
  \um_set_mathalphabet_char:Nnn \mathbbit {`\D,"1D437}{"2145}
  \um_set_mathalphabet_char:Nnn \mathbbit {`\d,"1D451}{"2146}
  \um_set_mathalphabet_char:Nnn \mathbbit {`\e,"1D452}{"2147}
  \um_set_mathalphabet_char:Nnn \mathbbit {`\i,"1D456}{"2148}
  \um_set_mathalphabet_char:Nnn \mathbbit {`\j,"1D457}{"2149}
}
```

9.1.4 Script or caligraphic: \mathscr and \mathcal

```
\cs_new:Npn \um_config_mathscr_Latin: {
  \um_set_mathalphabet_Latin:Nnn \mathscr {up,it}{scr}
  \um_set_mathalphabet_char:Nnn  \mathscr {`\B,"1D435}{"212C}
  \um_set_mathalphabet_char:Nnn  \mathscr {`\E,"1D438}{"2130}
  \um_set_mathalphabet_char:Nnn  \mathscr {`\F,"1D439}{"2131}
  \um_set_mathalphabet_char:Nnn  \mathscr {`\H,"1D43B}{"210B}
  \um_set_mathalphabet_char:Nnn  \mathscr {`\I,"1D43C}{"2110}
  \um_set_mathalphabet_char:Nnn  \mathscr {`\L,"1D43F}{"2112}
  \um_set_mathalphabet_char:Nnn  \mathscr {`\M,"1D440}{"2133}
  \um_set_mathalphabet_char:Nnn  \mathscr {`\R,"1D445}{"211B}
}
\cs_new:Npn \um_config_mathscr_latin: {
  \um_set_mathalphabet_latin:Nnn \mathscr {up,it}{scr}
  \um_set_mathalphabet_char:Nnn \mathscr {`\e,"1D452}{"212F}
  \um_set_mathalphabet_char:Nnn \mathscr {`\g,"1D454}{"210A}
  \um_set_mathalphabet_char:Nnn \mathscr {`\o,"1D45C}{"2134}
}
```

9.1.5 Fractur or fraktur or blackletter: \mathfrak

```
\cs_new:Npn \um_config_mathfrak_Latin: {
  \um_set_mathalphabet_Latin:Nnn \mathfrak {up,it}{frak}
  \um_set_mathalphabet_char:Nnn  \mathfrak {`\C,"1D436}{"212D}
  \um_set_mathalphabet_char:Nnn  \mathfrak {`\H,"1D43B}{"210C}
  \um_set_mathalphabet_char:Nnn  \mathfrak {`\I,"1D43C}{"2111}
  \um_set_mathalphabet_char:Nnn  \mathfrak {`\R,"1D445}{"211C}
  \um_set_mathalphabet_char:Nnn  \mathfrak {`\Z,"1D44D}{"2128}
}
\cs_new:Npn \um_config_mathfrak_latin: {
  \um_set_mathalphabet_latin:Nnn \mathfrak {up,it}{frak}
}
```

9.1.6 Sans serif upright: \mathsfup

```
\cs_new:Npn \um_config_mathsfup_num: {
  \um_set_mathalphabet_numbers:Nnn \mathsf   {up}{sf}
  \um_set_mathalphabet_numbers:Nnn \mathsfup {up}{sf}
}
\cs_new:Npn \um_config_mathsfup_Latin: {
  \bool_if:NTF \g_um_sfliteral_bool {
    \um_map_chars_Latin:nn {sfup} {sfup}
    \um_set_mathalphabet_Latin:Nnn \mathsf {up}{sfup}
  }{
    \bool_if:NT \g_um_upsans_bool {
      \um_map_chars_Latin:nn {sfup,sfit} {sfup}
      \um_set_mathalphabet_Latin:Nnn \mathsf {up,it}{sfup}
    }
  }
  \um_set_mathalphabet_Latin:Nnn \mathsfup {up,it}{sfup}
}
\cs_new:Npn \um_config_mathsfup_latin: {
  \bool_if:NTF \g_um_sfliteral_bool {
    \um_map_chars_latin:nn {sfup} {sfup}
    \um_set_mathalphabet_latin:Nnn \mathsf {up}{sfup}
  }{
    \bool_if:NT \g_um_upsans_bool {
      \um_map_chars_latin:nn {sfup,sfit} {sfup}
      \um_set_mathalphabet_latin:Nnn \mathsf {up,it}{sfup}
    }
  }
  \um_set_mathalphabet_latin:Nnn \mathsfup {up,it}{sfup}
}
```

9.1.7 Sans serif italic: \mathsfit

```
\cs_new:Npn \um_config_mathsfit_Latin: {
  \bool_if:NTF \g_um_sfliteral_bool {
    \um_map_chars_Latin:nn {sfit} {sfit}
    \um_set_mathalphabet_Latin:Nnn \mathsf {it}{sfit}
  }{
    \bool_if:NF \g_um_upsans_bool {
      \um_map_chars_Latin:nn {sfup,sfit} {sfit}
      \um_set_mathalphabet_Latin:Nnn \mathsf {up,it}{sfit}
    }
  }
  \um_set_mathalphabet_Latin:Nnn \mathsfit {up,it}{sfit}
}
\cs_new:Npn \um_config_mathsfit_latin: {
  \bool_if:NTF \g_um_sfliteral_bool {
    \um_map_chars_latin:nn {sfit} {sfit}
    \um_set_mathalphabet_latin:Nnn \mathsf {it}{sfit}
  }{
    \bool_if:NF \g_um_upsans_bool {
      \um_map_chars_latin:nn {sfup,sfit} {sfit}
      \um_set_mathalphabet_latin:Nnn \mathsf {up,it}{sfit}
    }
  }
  \um_set_mathalphabet_latin:Nnn \mathsfit {up,it}{sfit}
}
```

9.1.8 Typewriter or monospaced: \mathtt

```
\cs_new:Npn \um_config_mathtt_num: {
  \um_set_mathalphabet_numbers:Nnn \mathtt {up}{tt}
}
\cs_new:Npn \um_config_mathtt_Latin: {
  \um_set_mathalphabet_Latin:Nnn \mathtt {up,it}{tt}
}
\cs_new:Npn \um_config_mathtt_latin: {
  \um_set_mathalphabet_latin:Nnn \mathtt {up,it}{tt}
}
```

9.1.9 Bold Italic: \mathbfit

```
\cs_new:Npn \um_config_mathbfit_Latin: {
  \bool_if:NF \g_um_bfupLatin_bool {
    \um_map_chars_Latin:nn {bfup,bfit} {bfup}
  }
  \um_set_mathalphabet_Latin:Nnn \mathbfit {up,it}{bfit}
  \bool_if:NTF \g_um_bfliteral_bool {
    \um_map_chars_Latin:nn {bfit} {bfit}
    \um_set_mathalphabet_Latin:Nnn \mathbf {it}{bfit}
  }{
    \bool_if:NF \g_um_bfupLatin_bool {
      \um_map_chars_Latin:nn {bfup,bfit} {bfit}
      \um_set_mathalphabet_Latin:Nnn \mathbf {up,it}{bfit}
    }
  }
}
\cs_new:Npn \um_config_mathbfit_latin: {
  \bool_if:NF \g_um_bfuplatin_bool {
    \um_map_chars_latin:nn {bfup,bfit} {bfit}
  }
  \um_set_mathalphabet_latin:Nnn \mathbfit {up,it}{bfit}
  \bool_if:NTF \g_um_bfliteral_bool {
    \um_map_chars_latin:nn {bfit} {bfit}
    \um_set_mathalphabet_latin:Nnn \mathbf {it}{bfit}
  }{
    \bool_if:NF \g_um_bfuplatin_bool {
      \um_map_chars_latin:nn {bfup,bfit} {bfit}
      \um_set_mathalphabet_latin:Nnn \mathbf {up,it}{bfit}
    }
  }
}
\cs_new:Npn \um_config_mathbfit_Greek: {
  \um_set_mathalphabet_Greek:Nnn \mathbfit {up,it}{bfit}
  \bool_if:NTF \g_um_bfliteral_bool {
    \um_map_chars_Greek:nn {bfit}{bfit}
    \um_set_mathalphabet_Greek:Nnn \mathbf {it}{bfit}
  }{
    \bool_if:NF \g_um_bfupGreek_bool {
      \um_map_chars_Greek:nn {bfup,bfit}{bfit}
      \um_set_mathalphabet_Greek:Nnn \mathbf {up,it}{bfit}
    }
  }
}
\cs_new:Npn \um_config_mathbfit_greek: {
  \um_set_mathalphabet_greek:Nnn \mathbfit {up,it} {bfit}
  \bool_if:NTF \g_um_bfliteral_bool {
    \um_map_chars_greek:nn {bfit} {bfit}
    \um_set_mathalphabet_greek:Nnn \mathbfit {it} {bfit}
  }{
    \bool_if:NF \g_um_bfupgreek_bool {
      \um_map_chars_greek:nn {bfit,bfup} {bfit}
    }
    \bool_if:NF \g_um_bfupgreek_bool {
      \um_set_mathalphabet_greek:Nnn \mathbfit {up,it} {bfit}
    }
  }
}
\cs_new:Npn \um_config_mathbfit_misc: {
  \um_set_mathalphabet_pos:Nnnn  \mathbfit {partial} {up,it}{bfit}
  \um_set_mathalphabet_pos:Nnnn  \mathbfit {Nabla}   {up,it}{bfit}
  \bool_if:NTF \g_um_bfliteral_bool {
    \um_set_mathalphabet_pos:Nnnn  \mathbf {partial} {it}{bfit}
    \um_set_mathalphabet_pos:Nnnn  \mathbf {Nabla}   {it}{bfit}
  }{
    \bool_if:NF \g_um_upNabla_bool {
      \um_set_mathalphabet_pos:Nnnn  \mathbf {Nabla}   {up,it}{bfit}
    }
    \bool_if:NF \g_um_uppartial_bool {
      \um_set_mathalphabet_pos:Nnnn  \mathbf {partial} {up,it}{bfit}
    }
  }
}
```

9.1.10 Bold Upright: \mathbfup

```
\cs_new:Npn \um_config_mathbfup_num: {
  \um_set_mathalphabet_numbers:Nnn \mathbf   {up}{bfup}
  \um_set_mathalphabet_numbers:Nnn \mathbfup {up}{bfup}
}
\cs_new:Npn \um_config_mathbfup_Latin: {
  \bool_if:NT \g_um_bfupLatin_bool {
    \um_map_chars_Latin:nn {bfup,bfit} {bfit}
  }
  \um_set_mathalphabet_Latin:Nnn \mathbfup {up,it}{bfup}
  \bool_if:NTF \g_um_bfliteral_bool {
    \um_map_chars_Latin:nn {bfup} {bfup}
    \um_set_mathalphabet_Latin:Nnn \mathbf {up}{bfup}
  }{
    \bool_if:NT \g_um_bfupLatin_bool {
      \um_map_chars_Latin:nn {bfup,bfit} {bfup}
      \um_set_mathalphabet_Latin:Nnn \mathbf {up,it}{bfup}
    }
  }
}
\cs_new:Npn \um_config_mathbfup_latin: {
  \bool_if:NT \g_um_bfuplatin_bool {
    \um_map_chars_latin:nn {bfup,bfit} {bfup}
  }
  \um_set_mathalphabet_latin:Nnn \mathbfup {up,it}{bfup}
  \bool_if:NTF \g_um_bfliteral_bool {
    \um_map_chars_latin:nn {bfup} {bfup}
    \um_set_mathalphabet_latin:Nnn \mathbf {up}{bfup}
  }{
    \bool_if:NT \g_um_bfuplatin_bool {
      \um_map_chars_latin:nn {bfup,bfit} {bfup}
      \um_set_mathalphabet_latin:Nnn \mathbf {up,it}{bfup}
    }
  }
}
\cs_new:Npn \um_config_mathbfup_Greek: {
  \um_set_mathalphabet_Greek:Nnn \mathbfup {up,it}{bfup}
  \bool_if:NTF \g_um_bfliteral_bool {
    \um_map_chars_Greek:nn {bfup}{bfup}
    \um_set_mathalphabet_Greek:Nnn \mathbf {up}{bfup}
  }{
    \bool_if:NF \g_um_bfupGreek_bool {
      \um_map_chars_Greek:nn {bfup,bfit}{bfup}
      \um_set_mathalphabet_Greek:Nnn \mathbf {up,it}{bfup}
    }
  }
}
\cs_new:Npn \um_config_mathbfup_greek: {
  \um_set_mathalphabet_greek:Nnn \mathbfup {up,it} {bfup}
  \bool_if:NTF \g_um_bfliteral_bool {
    \um_map_chars_greek:nn {bfup} {bfup}
    \um_set_mathalphabet_greek:Nnn \mathbf {up} {bfup}
  }{
    \bool_if:NT \g_um_bfupgreek_bool {
      \um_map_chars_greek:nn {bfup,bfit} {bfup}
    }
    \bool_if:NT \g_um_bfupgreek_bool {
      \um_set_mathalphabet_greek:Nnn \mathbf {up,it} {bfup}
    }
  }
}
\cs_new:Npn \um_config_mathbfup_misc: {
  \um_set_mathalphabet_pos:Nnnn  \mathbfup {partial} {up,it}{bfup}
  \um_set_mathalphabet_pos:Nnnn  \mathbfup {Nabla}   {up,it}{bfup}
  \um_set_mathalphabet_pos:Nnnn  \mathbfup {digamma} {up}{bfup}
  \um_set_mathalphabet_pos:Nnnn  \mathbfup {Digamma} {up}{bfup}
  \um_set_mathalphabet_pos:Nnnn  \mathbf   {digamma} {up}{bfup}
  \um_set_mathalphabet_pos:Nnnn  \mathbf   {Digamma} {up}{bfup}
  \bool_if:NTF \g_um_bfliteral_bool {
    \um_set_mathalphabet_pos:Nnnn  \mathbf {partial} {up}{bfup}
    \um_set_mathalphabet_pos:Nnnn  \mathbf {Nabla}   {up}{bfup}
  }{
    \bool_if:NT \g_um_upNabla_bool {
      \um_set_mathalphabet_pos:Nnnn  \mathbf {Nabla}   {up,it}{bfup}
    }
    \bool_if:NT \g_um_uppartial_bool {
      \um_set_mathalphabet_pos:Nnnn  \mathbf {partial} {up,it}{bfup}
    }
  }
}
```

9.1.11 Bold fractur or fraktur or blackletter: \mathbffrak

```
\cs_new:Npn \um_config_mathbffrak_Latin: {
  \um_set_mathalphabet_Latin:Nnn \mathbffrak {up,it}{bffrak}
}
\cs_new:Npn \um_config_mathbffrak_latin: {
  \um_set_mathalphabet_latin:Nnn \mathbffrak {up,it}{bffrak}
}
```

9.1.12 Bold script or calligraphic: \mathbfscr

```
\cs_new:Npn \um_config_mathbfscr_Latin: {
  \um_set_mathalphabet_Latin:Nnn \mathbfscr {up,it}{bfscr}
}
\cs_new:Npn \um_config_mathbfscr_latin: {
  \um_set_mathalphabet_latin:Nnn \mathbfscr {up,it}{bfscr}
}
```

9.1.13 Bold upright sans serif: \mathbfsfup

```
\cs_new:Npn \um_config_mathbfsfup_num: {
  \um_set_mathalphabet_numbers:Nnn \mathbfsf   {up}{bfsfup}
  \um_set_mathalphabet_numbers:Nnn \mathbfsfup {up}{bfsfup}
}
\cs_new:Npn \um_config_mathbfsfup_Latin: {
  \bool_if:NTF \g_um_sfliteral_bool {
    \um_map_chars_Latin:nn {bfsfup} {bfsfup}
    \um_set_mathalphabet_Latin:Nnn \mathbfsf {up}{bfsfup}
  }{
    \bool_if:NT \g_um_upsans_bool {
      \um_map_chars_Latin:nn {bfsfup,bfsfit} {bfsfup}
      \um_set_mathalphabet_Latin:Nnn \mathbfsf {up,it}{bfsfup}
    }
  }
  \um_set_mathalphabet_Latin:Nnn \mathbfsfup {up,it}{bfsfup}
}
\cs_new:Npn \um_config_mathbfsfup_latin: {
  \bool_if:NTF \g_um_sfliteral_bool {
    \um_map_chars_latin:nn {bfsfup} {bfsfup}
    \um_set_mathalphabet_latin:Nnn \mathbfsf {up}{bfsfup}
  }{
    \bool_if:NT \g_um_upsans_bool {
      \um_map_chars_latin:nn {bfsfup,bfsfit} {bfsfup}
      \um_set_mathalphabet_latin:Nnn \mathbfsf {up,it}{bfsfup}
    }
  }
  \um_set_mathalphabet_latin:Nnn \mathbfsfup {up,it}{bfsfup}
}
\cs_new:Npn \um_config_mathbfsfup_Greek: {
  \bool_if:NTF \g_um_sfliteral_bool {
    \um_map_chars_Greek:nn {bfsfup}{bfsfup}
    \um_set_mathalphabet_Greek:Nnn \mathbfsf {up}{bfsfup}
  }{
    \bool_if:NT \g_um_upsans_bool {
      \um_map_chars_Greek:nn {bfsfup,bfsfit}{bfsfup}
      \um_set_mathalphabet_Greek:Nnn \mathbfsf {up,it}{bfsfup}
    }
  }
  \um_set_mathalphabet_Greek:Nnn \mathbfsfup {up,it}{bfsfup}
}
\cs_new:Npn \um_config_mathbfsfup_greek: {
  \bool_if:NTF \g_um_sfliteral_bool {
    \um_map_chars_greek:nn {bfsfup} {bfsfup}
    \um_set_mathalphabet_greek:Nnn \mathbfsf {up} {bfsfup}
  }{
    \bool_if:NT \g_um_upsans_bool {
      \um_map_chars_greek:nn {bfsfup,bfsfit} {bfsfup}
      \um_set_mathalphabet_greek:Nnn \mathbfsf {up,it} {bfsfup}
    }
  }
  \um_set_mathalphabet_greek:Nnn \mathbfsfup {up,it} {bfsfup}
}
\cs_new:Npn \um_config_mathbfsfup_misc: {
  \um_set_mathalphabet_pos:Nnnn  \mathbfsfup {partial} {up,it}{bfsfup}
  \um_set_mathalphabet_pos:Nnnn  \mathbfsfup {Nabla}   {up,it}{bfsfup}
  \bool_if:NTF \g_um_sfliteral_bool {
    \um_set_mathalphabet_pos:Nnnn  \mathbfsf {partial} {up}{bfsfup}
    \um_set_mathalphabet_pos:Nnnn  \mathbfsf {Nabla}   {up}{bfsfup}
  }{
    \bool_if:NT \g_um_upNabla_bool {
      \um_set_mathalphabet_pos:Nnnn  \mathbfsf {Nabla}   {up,it}{bfsfup}
    }
    \bool_if:NT \g_um_uppartial_bool {
      \um_set_mathalphabet_pos:Nnnn  \mathbfsf {partial} {up,it}{bfsfup}
    }
  }
}
```

9.1.14 Bold italic sans serif: \mathbfsfit

```
\cs_new:Npn \um_config_mathbfsfit_Latin: {
  \bool_if:NTF \g_um_sfliteral_bool {
    \um_map_chars_Latin:nn {bfsfit} {bfsfit}
    \um_set_mathalphabet_Latin:Nnn \mathbfsf {it}{bfsfit}
  }{
    \bool_if:NF \g_um_upsans_bool {
      \um_map_chars_Latin:nn {bfsfup,bfsfit} {bfsfit}
      \um_set_mathalphabet_Latin:Nnn \mathbfsf {up,it}{bfsfit}
    }
  }
  \um_set_mathalphabet_Latin:Nnn \mathbfsfit {up,it}{bfsfit}
}
\cs_new:Npn \um_config_mathbfsfit_latin: {
  \bool_if:NTF \g_um_sfliteral_bool {
    \um_map_chars_latin:nn {bfsfit} {bfsfit}
    \um_set_mathalphabet_latin:Nnn \mathbfsf {it}{bfsfit}
  }{
    \bool_if:NF \g_um_upsans_bool {
      \um_map_chars_latin:nn {bfsfup,bfsfit} {bfsfit}
      \um_set_mathalphabet_latin:Nnn \mathbfsf {up,it}{bfsfit}
    }
  }
  \um_set_mathalphabet_latin:Nnn \mathbfsfit {up,it}{bfsfit}
}
\cs_new:Npn \um_config_mathbfsfit_Greek: {
  \bool_if:NTF \g_um_sfliteral_bool {
    \um_map_chars_Greek:nn {bfsfit}{bfsfit}
    \um_set_mathalphabet_Greek:Nnn \mathbfsf {it}{bfsfit}
  }{
    \bool_if:NF \g_um_upsans_bool {
      \um_map_chars_Greek:nn {bfsfup,bfsfit}{bfsfit}
      \um_set_mathalphabet_Greek:Nnn \mathbfsf {up,it}{bfsfit}
    }
  }
  \um_set_mathalphabet_Greek:Nnn \mathbfsfit {up,it}{bfsfit}
}
\cs_new:Npn \um_config_mathbfsfit_greek: {
  \bool_if:NTF \g_um_sfliteral_bool {
    \um_map_chars_greek:nn {bfsfit} {bfsfit}
    \um_set_mathalphabet_greek:Nnn \mathbfsf {it} {bfsfit}
  }{
    \bool_if:NF \g_um_upsans_bool {
      \um_map_chars_greek:nn {bfsfup,bfsfit} {bfsfit}
      \um_set_mathalphabet_greek:Nnn \mathbfsf {up,it} {bfsfit}
    }
  }
  \um_set_mathalphabet_greek:Nnn \mathbfsfit {up,it} {bfsfit}
}
\cs_new:Npn \um_config_mathbfsfit_misc: {
  \um_set_mathalphabet_pos:Nnnn  \mathbfsfit {partial} {up,it}{bfsfit}
  \um_set_mathalphabet_pos:Nnnn  \mathbfsfit {Nabla}   {up,it}{bfsfit}
  \bool_if:NTF \g_um_sfliteral_bool {
    \um_set_mathalphabet_pos:Nnnn  \mathbfsf {partial} {it}{bfsfit}
    \um_set_mathalphabet_pos:Nnnn  \mathbfsf {Nabla}   {it}{bfsfit}
  }{
    \bool_if:NF \g_um_upNabla_bool {
      \um_set_mathalphabet_pos:Nnnn  \mathbfsf {Nabla}   {up,it}{bfsfit}
    }
    \bool_if:NF \g_um_uppartial_bool {
      \um_set_mathalphabet_pos:Nnnn  \mathbfsf {partial} {up,it}{bfsfit}
    }
  }
}
```

10 Definitions of the math symbols

Here we define every unicode math codepoint an equivalent macro name. The two are equivalent, in a `\let\xyz=^^^^1234` kind of way.

`\um@scancharlet`
`\um@scanactivedef`

We need to do some trickery to transform the `\UnicodeMathSymbol` argument `"ABCDEF` into the XeTeX 'caret input' form `^^^^^abcdef`. It is *very important* that the argument has five characters. Otherwise we need to change the number of ^ chars.

To do this, turn ^ into a regular 'other' character and define the macro to perform the lowercasing and `\let`. `\scantokens` changes the carets back into their original meaning after the group has ended and ^'s catcode returns to normal.

```
\begingroup
  \char_make_other:N \^
  \cs_gset:Npn \um@scancharlet#1="#2\@nil {
    \lowercase{
      \scantokens{\global\let#1=^^^^^#2}
    }
  }
```

Making ^ the right catcode isn't strictly necessary right now but it helps to future proof us with, e.g., breqn.

```
  \gdef\um@scanactivedef"#1\@nil#2{
    \lowercase{
      \tl_rescan:nn{
        \ExplSyntaxOn
        \char_make_math_superscript:N\^
      }{
        \global\def^^^^^#1{#2}
      }
    }
  }
\endgroup
```

Now give `\UnicodeMathSymbol` a definition in terms of `\um@scancharlet` and we're good to go. Make sure `#` is an 'other' so that we don't get confused with `\mathoctothorpe`.

```
\begingroup
  \def\UnicodeMathSymbol#1#2#3#4{
    \um@scancharlet#2=#1\@nil
  }
  \char_make_other:N \#
  \@input{unicode-math-table.tex}
\endgroup
```

Fix `\backslash`:

```
\group_begin:
  \lccode`\*=`\\
  \char_make_escape:N \|
  \char_make_other:N \\
  |lowercase{
|group_end:|let|backslash=*}
```

11 Epilogue

Lots of little things to tidy up.

11.0.15 Primes

We need a new 'prime' algorithm. Unicode math has four pre-drawn prime glyphs.

U+2032: PRIME (`\sprime`): x'

U+2033: DOUBLE PRIME (`\dprime`): x''

U+2034: TRIPLE PRIME (`\trprime`): x'''

U+2057: QUADRUPLE PRIME (`\qprime`): x''''

As you can see, they're all drawn at the correct height without being superscripted. However, in a correctly behaviour OpenType font with the MATH table, we also see different behaviour after the `ssty` feature is applied:

U+2032: PRIME in the 'scriptstyle' font: $x′$

The shrinking and offsetting is done as it is turned into a superscript. This means, luckily, that by default things work nicely for single primes.

However, it would be nice to use the pre-composed primes above if they exist in the font; consider x''' vs. $x‴$. Our algorithm is

- Prime encountered; pcount=1.
- Scan ahead; if prime: pcount:=pcount+1; repeat.
- If not prime, stop scanning.
- If pcount=1, `\sprime`, end.
- If pcount=2, check `\dprime`; if it exists, use it, end; if not, goto last step.
- Ditto pcount=3 & `\trprime`.
- Ditto pcount=4 & `\qprime`.
- If pcount>4 or the glyph doesn't exist, insert pcount `\primes` with `\primekern` between each.

```
\muskip_new:N \g_um_primekern_muskip
\muskip_gset:Nn \g_um_primekern_muskip { -\thinmuskip/2 }% arbitrary
\num_new:N \l_um_primecount_num
```

```
\cs_new:Nn \um_nprimes:n {
  ^{
     \sprime
     \prg_replicate:nn {#1-1} { \mskip \g_um_primekern_muskip \sprime }
   }
}
\cs_new:Nn \um_nprimes_select:n {
  \prg_case_int:nnn {#1}{
    {1} { ^{\sprime} }
    {2} {
      \um_glyph_if_exist:nTF {"2033} { ^{\dprime} } {\um_nprimes:n {#1}}
    }
    {3} {
      \um_glyph_if_exist:nTF {"2034} {^{\trprime} } {\um_nprimes:n {#1}}
    }
    {4} {
      \um_glyph_if_exist:nTF {"2057} { ^{\qprime} } {\um_nprimes:n {#1}}
    }
  }{
    \um_nprimes:n {#1}
  }
}
```

Scanning is more annoying than you'd think because we want to support all three of `\prime`, `'`, and the unicode prime. And `\ifx` doesn't work with mathactive chars.

```
\cs_new:Nn \um_scanprime: {
  \num_zero:N \l_um_primecount_num
  \um_scanprime_collect:
}
\cs_new:Nn \um_scanprime_collect: {
  \num_incr:N \l_um_primecount_num
  \peek_meaning_remove:NTF ' {
    \um_scanprime_collect:
  }{
    \peek_meaning_remove:NTF \um_scanprime: {
      \um_scanprime_collect:
    }{
      \peek_meaning_remove:NTF ^^^^2032 {
        \um_scanprime_collect:
      }{
        \um_nprimes_select:n {\l_um_primecount_num}
      }
    }
  }
}
```

```
\cs_set_eq:NN \prime \um_scanprime:
\group_begin:
  \char_make_active:N \'
  \char_make_active:n {"2032}
  \cs_gset_eq:NN ' \um_scanprime:
  \cs_gset_eq:NN ^^^^2032 \um_scanprime:
\group_end:
```

11.0.16 Unicode radicals

Undo the damage made to `\sqrt`:

```
\DeclareRobustCommand\sqrt{\@ifnextchar[\@sqrt\sqrtsign}
```

`\r@@t` `#1` : A mathstyle (for `\mathpalette`)
`#2` : Leading superscript for the sqrt sign
A re-implementation of LaTeX's hard-coded n-root sign using the appropriate `\fontdimens`.

```
\def\r@@t#1#2{
  \setbox\z@\hbox{$\m@th #1\sqrtsign{#2}$}
  \um@scaled@apply{#1}{\kern}{\fontdimen63\l_um_font}
  \raise \dimexpr(
      \um_fontdimen_to_percent:nn{65}{\l_um_font}\ht\z@-
      \um_fontdimen_to_percent:nn{65}{\l_um_font}\dp\z@
    )\relax
    \copy \rootbox
  \um@scaled@apply{#1}{\kern}{\fontdimen64\l_um_font}
  \box \z@
}
```

11.0.17 Unicode sub- and super-scripts

The idea here is to enter a scanning state after a superscript or subscript is encountered. If subsequent superscripts or subscripts (resp.) are found, they are lumped together. Each sub/super has a corresponding regular size glyph which is used by XeTeX to typeset the results; this means that the actual subscript/superscript glyphs are never seen in the output document — they are only used as input characters.

Open question: should the superscript-like 'modifiers' (U+1D2C: MODIFIER CAPITAL LETTER A and on) be included here?

First, the setup of each mathactive char:

```
\prop_new:N \g_um_supers_prop
\prop_new:N \g_um_subs_prop

\group_begin:

% Populate a property list with superscript characters; their meaning as their key,
% for reasons that will become apparent soon, and their replacement as each key's value.
% Then make the superscript active and bind it to the scanning function.
%
% \cs{scantokens} makes this process much simpler since we can activate the char
% and assign its meaning in one step.
\cs_set:Nn \um_setup_active_superscript:nn {
  \prop_gput:Nxn \g_um_supers_prop   {\meaning #1} {#2}
  \char_make_active:n {`#1}
  \global\XeTeXmathcodenum `#1 = "1FFFFF \scan_stop:
  \scantokens{
    \cs_gset:Npn #1 {
      \tl_set:Nn \l_um_ss_chain_tl {#2}
      \cs_set_eq:NN \um_sub_or_super:n \sp
      \tl_set:Nn \l_um_tmpa_tl {supers}
      \um_scan_sscript:
    }
  }
}

```

```
\um_setup_active_superscript:nn {^^^^2070} {0}
\um_setup_active_superscript:nn {^^^^00b9} {1}
\um_setup_active_superscript:nn {^^^^00b2} {2}
\um_setup_active_superscript:nn {^^^^00b3} {3}
\um_setup_active_superscript:nn {^^^^2074} {4}
\um_setup_active_superscript:nn {^^^^2075} {5}
\um_setup_active_superscript:nn {^^^^2076} {6}
\um_setup_active_superscript:nn {^^^^2077} {7}
\um_setup_active_superscript:nn {^^^^2078} {8}
\um_setup_active_superscript:nn {^^^^2079} {9}
\um_setup_active_superscript:nn {^^^^207a} {+}
\um_setup_active_superscript:nn {^^^^207b} {-}
\um_setup_active_superscript:nn {^^^^207c} {=}
\um_setup_active_superscript:nn {^^^^207d} {(}
\um_setup_active_superscript:nn {^^^^207e} {)}
\um_setup_active_superscript:nn {^^^^2071} {i}
\um_setup_active_superscript:nn {^^^^207f} {n}

% Ditto above.
\cs_set:Nn \um_setup_active_subscript:nn {
  \prop_gput:Nxn \g_um_subs_prop   {\meaning #1} {#2}
  \char_make_active:n {`#1}
  \global\XeTeXmathcodenum `#1 = "1FFFFF \scan_stop:
  \scantokens{
    \cs_gset:Npn #1 {
      \tl_set:Nn \l_um_ss_chain_tl {#2}
      \cs_set_eq:NN \um_sub_or_super:n \sb
      \tl_set:Nn \l_um_tmpa_tl {subs}
      \um_scan_sscript:
    }
  }
}

\um_setup_active_subscript:nn {^^^^2080} {0}
\um_setup_active_subscript:nn {^^^^2081} {1}
\um_setup_active_subscript:nn {^^^^2082} {2}
\um_setup_active_subscript:nn {^^^^2083} {3}
\um_setup_active_subscript:nn {^^^^2084} {4}
\um_setup_active_subscript:nn {^^^^2085} {5}
\um_setup_active_subscript:nn {^^^^2086} {6}
\um_setup_active_subscript:nn {^^^^2087} {7}
\um_setup_active_subscript:nn {^^^^2088} {8}
\um_setup_active_subscript:nn {^^^^2089} {9}
\um_setup_active_subscript:nn {^^^^208a} {+}
\um_setup_active_subscript:nn {^^^^208b} {-}
\um_setup_active_subscript:nn {^^^^208c} {=}
```

```
\um_setup_active_subscript:nn {^^^^208d} {(}
\um_setup_active_subscript:nn {^^^^208e} {)}
\um_setup_active_subscript:nn {^^^^2090} {a}
\um_setup_active_subscript:nn {^^^^2091} {e}
\um_setup_active_subscript:nn {^^^^1d62} {i}
\um_setup_active_subscript:nn {^^^^2092} {o}
\um_setup_active_subscript:nn {^^^^1d63} {r}
\um_setup_active_subscript:nn {^^^^1d64} {u}
\um_setup_active_subscript:nn {^^^^1d65} {v}
\um_setup_active_subscript:nn {^^^^2093} {x}
\um_setup_active_subscript:nn {^^^^1d66} {\beta}
\um_setup_active_subscript:nn {^^^^1d67} {\gamma}
\um_setup_active_subscript:nn {^^^^1d68} {\rho}
\um_setup_active_subscript:nn {^^^^1d69} {\phi}
\um_setup_active_subscript:nn {^^^^1d6a} {\chi}

\group_end:

% The scanning command, evident in its purpose:
\cs_new:Nn \um_scan_sscript: {
  \um_scan_sscript:TF {
    \um_scan_sscript:
  }{
    \um_sub_or_super:n {\l_um_ss_chain_tl}
  }
}

% The main theme here is stolen from the source to the various \cs{peek_} functions.
% Consider this function as simply boilerplate:
\cs_new:Nn \um_scan_sscript:TF {
  \tl_set:Nx \l_peek_true_aux_tl { \exp_not:n{ #1 } }
  \tl_set_eq:NN \l_peek_true_tl \c_peek_true_remove_next_tl
  \tl_set:Nx \l_peek_false_tl {\exp_not:n{\group_align_safe_end: #2}}
  \group_align_safe_begin:
    \peek_after:NN \um_peek_execute_branches_ss:
}

% We do not skip spaces when scanning ahead, and we explicitly wish to
% bail out on encountering a space or a brace.
\cs_new:Npn \um_peek_execute_branches_ss: {
  \bool_if:nTF {
    \token_if_eq_catcode_p:NN \l_peek_token \c_group_begin_token ||
    \token_if_eq_catcode_p:NN \l_peek_token \c_group_end_token ||
    \token_if_eq_meaning_p:NN \l_peek_token \c_space_token
  }
  { \l_peek_false_tl  }
  { \um_peek_execute_branches_ss_aux: }
}

% This is the actual comparison code.
% Because the peeking has already tokenised the next token,
% it's too late to extract its charcode directly. Instead,
% we look at its meaning, which remains a `character' even
% though it is itself math-active. If the character is ever
% made fully active, this will break our assumptions!
%
% If the char's meaning exists as a property list key, we
% build up a chain of sub-/superscripts and iterate. (If not, exit and
% typeset what we've already collected.)
\cs_new:Nn \um_peek_execute_branches_ss_aux: {
  \prop_if_in:cxTF
    {g_um_\l_um_tmpa_tl _prop}
    {\meaning\l_peek_token}
    {
      \prop_get:cxN
        {g_um_\l_um_tmpa_tl _prop}
        {\meaning\l_peek_token}
        \l_um_tmpb_tl
      \tl_put_right:NV \l_um_ss_chain_tl \l_um_tmpb_tl
      \l_peek_true_tl
    }
    {\l_peek_false_tl}
}
```

11.0.18 Synonyms and all the rest

We need to change LaTeX's idea of the font used to typeset things like \sin and \cos:

```
\def\operator@font{\um_setup_mathup:}
```

```
\def\to{\rightarrow}
\def\overrightarrow{\vec}
\def\le{\leq}
\def\ge{\geq}
\def\neq{\ne}
\def\triangle{\mathord{\bigtriangleup}}
\def\bigcirc{\mdlgwhtcircle}
\def\circ{\vysmwhtcircle}
\def\mathyen{\yen}
\def\mathsterling{\sterling}
```

Define \colon as a mathpunct ':'. This is wrong: it should be U+003A: COLON instead!

```
\@ifpackageloaded{amsmath}{
  % define their own colon, perhaps I should just steal it.
}{
  \cs_set_protected:Npn \colon {
    \bool_if:NTF \g_um_literal_colon_bool {:} { \mathpunct{:} }
  }
}
```

\mathcal

```
\def\mathcal{\mathscr}
```

\mathrm

```
\def\mathrm{\mathup}
\let\mathfence\mathord
```

11.0.19 Compatibility

Note that amsmath will always be loaded before unicode-math. (Conflicts occur if you try it the other way around.)

- Since the mathcode of `\- is greater than eight bits, this piece of \AtBeginDocument code from amsmath dies if we try and set the maths font in the preamble:

```
\bool_new:N \g_um_amsmath_bool
\@ifpackageloaded{amsmath}{
  \bool_set_true:N \g_um_amsmath_bool
}{
  \bool_set_false:N \g_um_amsmath_bool
}
\bool_if:NT \g_um_amsmath_bool {
  \tl_remove_in:Nn \@begindocumenthook {
    \mathchardef\std@minus\mathcode`\-\relax
    \mathchardef\std@equal\mathcode`\=\relax
  }
}
```

- This code is to improve the output of analphabetic symbols in text of operator names (\sin, \cos, etc.). Just comment out the offending lines for now:

```
\@ifpackageloaded{amsopn}{
  \cs_set:Npn \newmcodes@ {
    \mathcode`\'39
    \mathcode`\*42
    \mathcode`\."613A%
%  \ifnum\mathcode`\-=45 \else
%    \mathchardef\std@minus\mathcode`\-\relax
%  \fi
    \mathcode`\-45
    \mathcode`\/47
    \mathcode`\:"603A\relax
  }
}{}
```

- \mathinner items:

```
    \cs_set:Npn \mathellipsis {\mathinner{\unicodeellipsis}}
    \cs_set:Npn \cdots {\mathinner{\unicodecdots}}
    \bool_if:NT \g_um_amsmath_bool {
      \cs_set_eq:NN \@cdots \cdots
      \cs_set_eq:NN \dotsb@ \cdots
    }
```

Octothorpe is an odd one:

```
\AtBeginDocument{
  \def\#{\mode_if_math:TF{\mathoctothorpe}{\char`\#}}
  \def\widehat{\hat}
  \def\widetilde{\tilde}
}
```

\digamma \Digamma I might end up just changing these in the table.

```
\def\digamma{\updigamma}
\def\Digamma{\upDigamma}
```

Overriding amsmath definitions:

```
\AtBeginDocument{
  \def\@cdots{\mathinner{\cdots}}
}
```

Interaction with beamer:

```
\@ifclassloaded{beamer}{
  \ifbeamer@suppressreplacements\else
    \PackageWarningNoLine{unicode-math}{
      Disabling~ beamer's~ math~ setup.^^J
      Please~ load~ beamer~ with~ the~ [professionalfonts]~ class~ option
    }
    \beamer@suppressreplacementstrue
  \fi
}{}
```

The end.

```
\ExplSyntaxOff
```

12 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 check `http://www.ams.org/STIX/` for more up-to-date info.

This table is converted into a form suitable for reading by XeTeX, and then hand-edited by the author; the result is `unicode-math-table.tex`.

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. Performance for now seems to be acceptable without such measures.

```
#!/bin/sh

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

If the USV isn't repeated (TODO: check this is valid!) and the entry isn't one of the weird ones in the big block at the end of the STIX table (TODO: check that out!)…

```
 {if (usv != substr($0,2,5) && 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 the USV has a macro name, which isn't `\text...`, and isn't a single character macro (e.g., `\#`, `\S`, …), and has a class, and it isn't reserved (*i.e.*, doubled up with a previously assigned glyph):

```
     if (texname      ~ /[\\]/ &&
         substr(texname,0,5) != "\\text"    &&
         substr(texname,0,4) != "\\ipa"    &&
         substr(texname,0,5) != "\\tone"    &&
         substr(texname,3,1) != " "    &&
         class       != " "    &&
         description !~ /<reserved>/ )
```

Print the actual entry corresponding to the unicode character:

```
     print "\\UnicodeMathSymbol{\"" \
           usv "}{" \
           texname "}{" \
           class "}{" \
           description "}%";
    }}' - |
```

Now replace the STIX class abbreviations with their TeX macro names.

```
sed -e ' s/{N}/{\\mathord}/   ' \
```

A ‘fence’ defined by the STIX table is something like `\vert`; in XeTeX this is just a `\mathord` that will grow with the magic of `\XeTeXmathchardef`.

```
      -e ' s/{F}/{\\mathord}/   ' \
      -e ' s/{A}/{\\mathalpha}/ ' \
      -e ' s/{D}/{\\mathaccent}/ ' \
      -e ' s/{P}/{\\mathpunct}/ ' \
      -e ' s/{B}/{\\mathbin}/   ' \
      -e ' s/{R}/{\\mathrel}/   ' \
      -e ' s/{L}/{\\mathop}/    ' \
      -e ' s/{O}/{\\mathopen}/  ' \
      -e ' s/{C}/{\\mathclose}/ ' \
```

Fixing up a couple of things in the STIX table.

```
      -e ' s/\^/\\string^/   ' > unicode-math.tex
```

A Documenting maths support in the NFSS

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

Maths symbol fonts Fonts for symbols: ∝, ≤, →

`\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 XƎTEX. 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. This is not the case in XƎTEX.

Accents are not included yet.

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@}}}
```

B XƎTEX math font dimensions

These are the extended `\fontdimen`s available for suitable fonts in XƎTEX. Note that LuaTEX takes an alternative route, and this package will eventually provide a wrapper interface to the two (I hope).

`\fontdimen`	Dimension name	Description
10	ScriptPercentScaleDown	Percentage of scaling down for script level 1. Suggested value: 80%.
11	ScriptScriptPercentScaleDown	Percentage of scaling down for script level 2 (ScriptScript). Suggested value: 60%.

\fontdimen	Dimension name	Description
12	DelimitedSubFormulaMinHeight	Minimum height required for a delimited expression to be treated as a subformula. Suggested value: normal line height × 1.5.
13	DisplayOperatorMinHeight	Minimum height of n-ary operators (such as integral and summation) for formulas in display mode.
14	MathLeading	White space to be left between math formulas to ensure proper line spacing. For example, for applications that treat line gap as a part of line ascender, formulas with ink going above (os2.sTypoAscender + os2.sTypoLineGap – MathLeading) or with ink going below os2.sTypoDescender will result in increasing line height.
15	AxisHeight	Axis height of the font.
16	AccentBaseHeight	Maximum (ink) height of accent base that does not require raising the accents. Suggested: x-height of the font (os2.sxHeight) plus any possible overshots.
17	FlattenedAccentBaseHeight	Maximum (ink) height of accent base that does not require flattening the accents. Suggested: cap height of the font (os2.sCapHeight).
18	SubscriptShiftDown	The standard shift down applied to subscript elements. Positive for moving in the downward direction. Suggested: os2.ySubscriptYOffset.
19	SubscriptTopMax	Maximum allowed height of the (ink) top of subscripts that does not require moving subscripts further down. Suggested: /5 x-height.
20	SubscriptBaselineDropMin	Minimum allowed drop of the baseline of subscripts relative to the (ink) bottom of the base. Checked for bases that are treated as a box or extended shape. Positive for subscript baseline dropped below the base bottom.
21	SuperscriptShiftUp	Standard shift up applied to superscript elements. Suggested: os2.ySuperscriptYOffset.

\fontdimen	Dimension name	Description
22	SuperscriptShiftUpCramped	Standard shift of superscripts relative to the base, in cramped style.
23	SuperscriptBottomMin	Minimum allowed height of the (ink) bottom of superscripts that does not require moving subscripts further up. Suggested: ¼ x-height.
24	SuperscriptBaselineDropMax	Maximum allowed drop of the baseline of superscripts relative to the (ink) top of the base. Checked for bases that are treated as a box or extended shape. Positive for superscript baseline below the base top.
25	SubSuperscriptGapMin	Minimum gap between the superscript and subscript ink. Suggested: 4×default rule thickness.
26	SuperscriptBottomMaxWithSubscript	The maximum level to which the (ink) bottom of superscript can be pushed to increase the gap between superscript and subscript, before subscript starts being moved down. Suggested: /5 x-height.
27	SpaceAfterScript	Extra white space to be added after each subscript and superscript. Suggested: 0.5pt for a 12 pt font.
28	UpperLimitGapMin	Minimum gap between the (ink) bottom of the upper limit, and the (ink) top of the base operator.
29	UpperLimitBaselineRiseMin	Minimum distance between baseline of upper limit and (ink) top of the base operator.
30	LowerLimitGapMin	Minimum gap between (ink) top of the lower limit, and (ink) bottom of the base operator.
31	LowerLimitBaselineDropMin	Minimum distance between baseline of the lower limit and (ink) bottom of the base operator.
32	StackTopShiftUp	Standard shift up applied to the top element of a stack.
33	StackTopDisplayStyleShiftUp	Standard shift up applied to the top element of a stack in display style.
34	StackBottomShiftDown	Standard shift down applied to the bottom element of a stack. Positive for moving in the downward direction.

\fontdimen	Dimension name	Description
35	StackBottomDisplayStyleShiftDown	Standard shift down applied to the bottom element of a stack in display style. Positive for moving in the downward direction.
36	StackGapMin	Minimum gap between (ink) bottom of the top element of a stack, and the (ink) top of the bottom element. Suggested: 3×default rule thickness.
37	StackDisplayStyleGapMin	Minimum gap between (ink) bottom of the top element of a stack, and the (ink) top of the bottom element in display style. Suggested: 7×default rule thickness.
38	StretchStackTopShiftUp	Standard shift up applied to the top element of the stretch stack.
39	StretchStackBottomShiftDown	Standard shift down applied to the bottom element of the stretch stack. Positive for moving in the downward direction.
40	StretchStackGapAboveMin	Minimum gap between the ink of the stretched element, and the (ink) bottom of the element above. Suggested: UpperLimitGapMin
41	StretchStackGapBelowMin	Minimum gap between the ink of the stretched element, and the (ink) top of the element below. Suggested: LowerLimitGapMin.
42	FractionNumeratorShiftUp	Standard shift up applied to the numerator.
43	FractionNumeratorDisplayStyleShiftUp	Standard shift up applied to the numerator in display style. Suggested: StackTopDisplayStyleShiftUp.
44	FractionDenominatorShiftDown	Standard shift down applied to the denominator. Positive for moving in the downward direction.
45	FractionDenominatorDisplayStyleShiftDown	Standard shift down applied to the denominator in display style. Positive for moving in the downward direction. Suggested: StackBottomDisplayStyleShiftDown.

`\fontdimen`	Dimension name	Description
46	FractionNumeratorGapMin	Minimum tolerated gap between the (ink) bottom of the numerator and the ink of the fraction bar. Suggested: default rule thickness
47	FractionNumDisplayStyleGapMin	Minimum tolerated gap between the (ink) bottom of the numerator and the ink of the fraction bar in display style. Suggested: 3×default rule thickness.
48	FractionRuleThickness	Thickness of the fraction bar. Suggested: default rule thickness.
49	FractionDenominatorGapMin	Minimum tolerated gap between the (ink) top of the denominator and the ink of the fraction bar. Suggested: default rule thickness
50	FractionDenomDisplayStyleGapMin	Minimum tolerated gap between the (ink) top of the denominator and the ink of the fraction bar in display style. Suggested: 3×default rule thickness.
51	SkewedFractionHorizontalGap	Horizontal distance between the top and bottom elements of a skewed fraction.
52	SkewedFractionVerticalGap	Vertical distance between the ink of the top and bottom elements of a skewed fraction.
53	OverbarVerticalGap	Distance between the overbar and the (ink) top of he base. Suggested: 3×default rule thickness.
54	OverbarRuleThickness	Thickness of overbar. Suggested: default rule thickness.
55	OverbarExtraAscender	Extra white space reserved above the overbar. Suggested: default rule thickness.
56	UnderbarVerticalGap	Distance between underbar and (ink) bottom of the base. Suggested: 3×default rule thickness.
57	UnderbarRuleThickness	Thickness of underbar. Suggested: default rule thickness.
58	UnderbarExtraDescender	Extra white space reserved below the underbar. Always positive. Suggested: default rule thickness.

`\fontdimen`	Dimension name	Description
59	RadicalVerticalGap	Space between the (ink) top of the expression and the bar over it. Suggested: 1¼ default rule thickness.
60	RadicalDisplayStyleVerticalGap	Space between the (ink) top of the expression and the bar over it. Suggested: default rule thickness + ¼ x-height.
61	RadicalRuleThickness	Thickness of the radical rule. This is the thickness of the rule in designed or constructed radical signs. Suggested: default rule thickness.
62	RadicalExtraAscender	Extra white space reserved above the radical. Suggested: RadicalRuleThickness.
63	RadicalKernBeforeDegree	Extra horizontal kern before the degree of a radical, if such is present. Suggested: 5/18 of em.
64	RadicalKernAfterDegree	Negative kern after the degree of a radical, if such is present. Suggested: −10/18 of em.
65	RadicalDegreeBottomRaisePercent	Height of the bottom of the radical degree, if such is present, in proportion to the ascender of the radical sign. Suggested: 60%.

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.