The LinguisTiX bundle

निरंजन

May 20, 2025 (v0.1b)

https://ctan.org/pkg/linguistix
https://puszcza.gnu.org.ua/projects/linguistix
https://matrix.to/#/#linguistix:matrix.org

Abstract

There are quite a few LaTeX packages that support typesetting in linguistics, but most of them lack a modern LaTeX-like users syntax as well as a programming interface. The LinguisTiX bundle fills this gap. It contains several packages enhancing the general support for linguistics in LaTeX. This is a comprehensive documentation of the same comprising of three parts. The first one is the general users manual, the second one documents the programming interface of the bundle, whereas the last one is the documented implementation of all the packages.

Contents

Dedicated to Renuka who taught me rigour under the guise of linguistics...

1 Introduction

Linguistics is a discipline that studies the phenomenon of language and for this linguists analyse data from languages across the globe. In order to be able to present the data that is collected for this, linguists use several representational methods that lead to a fiasco when their typesetting is considered. In order to understand the complexity of the task at hand, first, let's have a look at some of the problem cases first. If you are an impatient user and are just willing to read the users manual, you may skip reading the current section and start with section 5 and the ones following it.

1.1 Phonetic symbols

Speech sounds are the building blocks of many human languages and the data collected from languages demands an unambiguous method of representation which is served by the International Phonetic Alphabet. For the longest time, the TIPA package (`https://ctan.org/pkg/tipa`) was the one that produced phonetic symbols in (LA)TEX. Visually, it matches the default Computer Modern design of (LA)TEX, but TIPA is not Unicode. It is set in a legacy encoding. With the recent developments, the New Computer Modern family supports all the IPA characters (even the ones that are missing in TIPA). They are created keeping in mind the principles of Knuth's Computer Modern. Additionally, the family also supports sans serif (recommended in presentations) and mono (recommended in coding context) families. It supports two weights, i.e., book and regular respectively. The book weight is slightly thicker than the regular weight, but the regular one matches the thickness of the Computer Modern design. Because of the increased thickness, the former looks better. The current document, for example, is typeset in the book weight of New Computer Modern. If you are like me, you probably don't like using non-LATEX-fonts. The good news is that the requirements of linguistics are very well fulfilled by the recent developments in the New Computer modern family and it *does* belong to the fraternity of LATEX-fonts.

Apart from this, there are some other advantages of the New Computer Modern fonts. The IPA distinguishes between [a] and [ɑ], but unfortunately, in Italic shape, the latter is a variant of the former. E.g., `[a\textit{a}]` produces '[a*a*]'. Whenever an author uses Italic shape for their transcription and use `a`, a wrong IPA symbol is printed with most fonts. This problem was kindly acknowledged by Antonis Tsolomitis, the developer of New Computer Modern. In the stylistic set dedicated for linguistics, the correct shape was added to the Italic shape by him. Thus, `\ipatext{a\textit{a}}` (a command from LinguisTiX-IPA) renders '[a*a*]'. The package enables New Computer Modern family with stylistic set 05 dedicated for IPA. It also adds the brackets or slashes around the argument as explained in section 7.

A similar problem is with the character `g`. E.g., `[g\textit{g}]` produces '[g*g*]'. Here, the situation is the other way round. The upright 'g' is not recognised by the IPA. The IPA charts generally have the upright version of the Italic shape. To see what this means, try `\ipatext{g\textit{g}}`. It produces [ɡ*g*] and not [g*g*].

In order to avail all of these features, I have set New Computer Modern as the default font-family of LinguisTiX. The bundle provides options to control these defaults. Users can use their preferred text and IPA fonts. There also is an option to use the regular weight of NewCM instead of the book weight.

2 Planned

I plan to develop this bundle further in order to support the typesetting of good quality examples with interlinear glossing. My model is to imitate the output of the expex package, but with a modern LaTeX-like syntax. I also plan to provide support for glossing. Currently the leipzig package is used, but it has some unresolved bugs. Some syntactic improvements are also possible, I believe.

3 Funding

I am a doctorate student without a fellowship (thanks to our education policies!) currently sustaining only with a full time job unrelated to linguistics that consumes most of my working hours. At times, it becomes difficult to continue the research, the job and the passion development projects. LinguisTiX needs funding in order to sustain. If you think you can support it, you can contact me on the email ID found on the front page.

4 Acknowledgements

This package relies the most on the New Computer Modern font family. I would like to express my gratitude to Antonis Tsolomitis who tirelessly worked on the set of IPA symbols and brought back the good old charm of TIPA's design in the modern Unicode world. I would like to thank Renuka and Avinash who taught me linguistics. They nourished my passion, helped me pursue my love for the subject as well as the computation that came along with it. I could have never imagined myself working on a package written in LaTeX3's syntax. Not so long ago, I used to find it very complicated. It's mostly Jonathan Spratte and Florent Rougon's help (and, at times, scolding :P) that helped me get used to it. I would also like to mention C.V. Radhakrishnan for being an important part of my journey in LaTeX. Lastly, to all the free software people who have created this friendly and supportive world for people by investing their precious time and resources!

Documentation

The bundle is comprised of several packages that are developed for different purposes. In order to load all the packages of the bundle, one can issue:

```
\usepackage{linguistix}
```

This is the easiest method for getting all of LinguisTiX in one go. But, if you don't need all the packages of the bundle, you may load the required packages separately. We will start with the elementary package that sets up things for other packages of the bundle.

5 LinguisTiX-base

LaTeX3-interface | Implementation

This package provides a single command that is used in all the other packages of the bundle. The command is:

`\linguistix {⟨key-value-list⟩}`

We have a single set of keys for the entire bundle. Each package appends keys to the same set. The argument of this central processor command is the comma-separated ⟨*key-value-list*⟩. So you can load any package of LinguisTiX and use the `\linguistix` command. The only exception to this is LinguisTiX-nfss. We will see how it is different in its section.

6 LinguisTiX-fonts

LaTeX3-interface | Implementation

This is a package that loads the New Computer Modern family for the entire document. The package sets fonts for both text and math. It has keys for customisation for both. Note that just loading this package does *not* provide any support for IPA. For that one needs LinguisTiX-ipa separately. Let's look at the keys provided for the text.

6.1 Text

Most keys of this package are prefixed with the `text` in order to distinguish them from the maths and IPA ones. There aren't any commands provided by the package. Most of the important features of the fontspec packakge are variablised with l3keys.

The 'old style numbers' have varying heights. Some numbers have ascenders and some have descenders (e.g., 6789). According to Bringhurst, 2004, this makes them easier to read in running text. Lining numbers, on the other hand have uniform heights. They go well with all capital text (rare). Thus, for the general text, I enable this setting by default in LinguisTiX-fonts.

Apart from that, the New Computer Modern font family provides an old-style shape for the number '1' (this exact shape!), but it is provided as a character variant. Different fonts may use these arbitrary slots for any character's alternation. Therefore this setting should not be loaded blindly. Let's have a look at the keys that can be employed to change these behaviours.

```
old style numbers = {⟨truth value⟩}                    true | false
old style one     = {⟨truth value⟩}                    true | false
```

If one wants to disable old style numbers, they may use the `old style numbers` key with the `false` value (default is `true`)[1]. Note that printing of old style numbers also depends on whether the font you select has old style numbers or not. The relevant settings are added by the package to the font automatically, but while selecting the font, make sure whether the old style table is present in the font or not.

Suppose one wants the alternative shape of number '1' from the New Computer Modern family, they may use the key `old style one` (default is false; adding `true` is optional).

Let's have a look at the three way distinction we get because of this.

0123456789	`Old style with default 1`
0123456789	`Old style with the old 1`
0123456789	`Lining`

[1]The possible and the default values of keys are given at the right side in the documentation and the defaults are highlighted in red.

`newcm`
`newcm sans`
`newcm mono`
`newcm regular`
`newcm regular sans`
`newcm regular mono`

These are some keys that come in handy for setting New Computer Modern defaults. All the necessary values are stored in these. The keys that have `regular` in their names refer to the 'regular' variants of New Computer Modern fonts. These variants match the colour and widths of the Latin Modern fonts. One may use these keys to override the changed defaults.

6.2 Maths

LinguisTiX-fonts sets maths fonts also. In order to control the settings related to maths, the following keys can be used.

```
math                  = {⟨math font⟩}
math features         = {⟨math font features⟩}
math bold             = {⟨bold math font⟩}
math bold features    = {⟨bold math font features⟩}
```

The `math` and `math bold` keys set the respective fonts (i.e., regular and bold fonts for mathematics respectively). The keys suffixed with `features` set the font features of the same.

```
bourbaki's empty set = {⟨truth value⟩}                true | false
```

In (LA)TEX, the default shape of the 'empty set' symbol is: '$\emptyset$', but the symbol used by the Bourbaki group is still considered more correct and preferred by many (including me). New Computer Modern Math fonts provide it as a character variant that I activate by default. Thus `$\emptyset$` always renders: '$\varnothing$' and not: '$\emptyset$'. In order to change this behaviour, one may use this key and set it to false for getting the slashed-zero of original (LA)TEX. Hail plumbers union, *IYKYK! ;-)*

7 LinguisTiX-ipa

LATEX3-interface | Implementation

This package sets the fonts exclusively for the IPA. The commands provided for switching to the IPA control all serif, sans serif and typewriter families. This package can be loaded standalone for loading IPA fonts as well as some switch commands useful in running text. New Computer Modern provides a special stylistic set dedicated for linguistics. It is enabled for IPA fonts automatically with this package. Only the legally marked up IPA is affected by the customisation provided by this package. For switching to the IPA, LinguisTiX-ipa provides one command with a starred variant.

```
\ipatext  {⟨phonetic transcription⟩}
\ipatext* {⟨phonemic transcription⟩}
```

This is a command that resembles with the TIPA command `\textipa`. I have deliberately kept it distinct from it so that just in case somebody wants to use their old TIPA code in a Unicode document, the commands won't clash (I highly discourage doing this, though). The command comes with a starred variant. The behaviour of the unstarred command is to print the argument in brackets for phonetic transcription, e.g.: `\ipatext{aɪ pʰiː eɪ}` ⟶ [aɪ̯ pʰiː eɪ̯] whereas the starred version prints it in slashes for phonemic transcription, e.g.: `\ipatext*{aɪ pʰiː eɪ}` ⟶ /aɪ̯ pʰiː eɪ̯/.

Suppose someone just wants to load the font without the brackets or slashes, they cann use the following command for switching to the IPA without adding the aforementioned.

`\lngxipa` This also is a command that switches to the IPA-only features (default as well as user added). This command, of course, leaks and that's why *should* be delimited. E.g., the following code lines produce [aɪ̯ pʰiː eɪ̯] and /aɪ̯ pʰiː eɪ̯/ respectively:

```
{\lngxipa [aɪ̯ pʰiː eɪ̯]}
{\lngxipa /aɪ̯ pʰiː eɪ̯/}
```

```
ipa newcm
ipa newcm sans
ipa newcm mono
ipa newcm regular
ipa newcm regular sans
ipa newcm regular mono
```

All the IPA fonts are stored in variables as seen in table 1 and table 2. These keys reset the IPA-only fonts to New Computer Modern. They can be used even for resetting to New Computer Modern from another IPA font. In order to change or reset to the IPA defaults these keys can be used. They store the names of the New Computer Modern font family in the variables concerning IPA. The keys that contain `regular` in their name use the regular version of New Computer Modern that matches the colour of Latin Modern.

Let's now see the combined table of font keys provided by both LinguisTiX-fonts and LinguisTiX-ipa.

Family	LinguisTiX-fonts	LinguisTiX-ipa
Serif	`text upright`	`ipa upright`
	`text upright features`	`ipa upright features`
	`text bold upright`	`ipa bold upright`
	`text bold upright features`	`ipa bold upright features`
	`text italic`	`ipa italic`
	`text italic features`	`ipa italic features`
	`text bold italic`	`ipa bold italic`
	`text bold italic features`	`ipa bold italic features`
	`text slanted`	`ipa slanted`
	`text slanted features`	`ipa slanted features`
	`text bold slanted`	`ipa bold slanted`
	`text bold slanted features`	`ipa bold slanted features`
	`text swash`	`ipa swash`
	`text swash features`	`ipa swash features`
	`text bold swash`	`ipa bold swash`
	`text bold swash features`	`ipa bold swash features`
	`text small caps`	`ipa small caps`
	`text small caps features`	`ipa small caps features`
Sans serif	`text sans upright`	`ipa sans upright`
	`text sans upright features`	`ipa sans upright features`
	`text sans bold upright`	`ipa sans bold upright`
	`text sans bold upright features`	`ipa sans bold upright features`
	`text sans italic`	`ipa sans italic`
	`text sans italic features`	`ipa sans italic features`

Continued on the next page…

Family	LinguisTiX-fonts	LinguisTiX-ipa
	`text sans bold italic`	`ipa sans bold italic`
	`text sans bold italic features`	`ipa sans bold italic features`
	`text sans slanted`	`ipa sans slanted`
	`text sans slanted features`	`ipa sans slanted features`
	`text sans bold slanted`	`ipa sans bold slanted`
	`text sans bold slanted features`	`ipa sans bold slanted features`
	`text sans swash`	`ipa sans swash`
	`text sans swash features`	`ipa sans swash features`
	`text sans bold swash`	`ipa sans bold swash`
	`text sans bold swash features`	`ipa sans bold swash features`
	`text sans small caps`	`ipa sans small caps`
	`text sans small caps features`	`ipa sans small caps features`
Monospaced	`text mono upright`	`ipa mono upright`
	`text mono upright features`	`ipa mono upright features`
	`text mono bold upright`	`ipa mono bold upright`
	`text mono bold upright features`	`ipa mono bold upright features`
	`text mono italic`	`ipa mono italic`
	`text mono italic features`	`ipa mono italic features`
	`text mono bold italic`	`ipa mono bold italic`
	`text mono bold italic features`	`ipa mono bold italic features`
	`text mono slanted`	`ipa mono slanted`
	`text mono slanted features`	`ipa mono slanted features`
	`text mono bold slanted`	`ipa mono bold slanted`
	`text mono bold slanted features`	`ipa mono bold slanted features`
	`text mono swash`	`ipa mono swash`
	`text mono swash features`	`ipa mono swash features`
	`text mono bold swash`	`ipa mono bold swash`
	`text mono bold swash features`	`ipa mono bold swash features`
	`text mono small caps`	`ipa mono small caps`
	`text mono small caps features`	`ipa mono small caps features`
		End of the table...

Table 1: Font keys provided by LinguisTiX-fonts and LinguisTiX-ipa

8 LinguisTiX-Logos

LaTeX3-interface | **Implementation**

This is a small package that provides commands for printing logos of the LinguisTiX bundle. The logo is printed in New Computer Modern Uncial font. It uses purple colour for the 'X' in it and it is defined using l3color module. It provides one command that takes an optional argument. Obviously it is '`protected`'. It is as follows:

`\lngxlogo [⟨package name⟩]`

The logo of the ⟨*package name*⟩ from the LinguisTiX bundle is printed with this command, e.g., `\lngxlogo{fonts}` ⟶ LinguisTiXfonts.

Sometimes, the logos might be required to be used in an expandable way, but optional arguments are not supported in expandable commands. Thus we create separate commands for separate packages. Even these ones have the `lngx` prefix. It is followed by the package name, e.g., `fonts` or `ipa` and finally the suffix `logo`. In the context of hyperref, their behaviour is different than in the context of normal text. The commands and their are as follows:

`\lngxpkg`	⋆	LinguisTiX
`\lngxbaselogo`	⋆	LinguisTiX-base
`\lngxfontslogo`	⋆	LinguisTiX-fonts
`\lngxipalogo`	⋆	LinguisTiX-ipa
`\lngxlogoslogo`	⋆	LinguisTiX-Logos
`\lngxnfsslogo`	⋆	LinguisTiX-nfss

9 LinguisTiX-nfss

LaTeX3-interface | Implementation

This is an extension package to the existing NFSS scheme of LaTeX. The NFSS mainly works on the four facets of the text.

1. Encoding
2. Family
3. Shape
4. Series

These facets are reset to default by the `\normalfont` and `\selectfont` commands. These commands work on some internals that are reset with every usage of some commands that set them, e.g., `\rmfamily`, `\bfseries`. There isn't any way to control this unless some internals are touched and there might be incidences where one does want to control them, e.g., try compiling the following code in LuaLaTeX.

```
\documentclass{article}

\begin{document}
\makeatletter
\fontencoding{OT1}\sffamily\itshape\bfseries
\selectfont
\f@encoding\ | \f@family\ | \f@series\ | \f@shape\quad
\normalfont
\f@encoding\ | \f@family\ | \f@series\ | \f@shape
\end{document}
```

As can be seen in the output, the first line shows the text in OT1 encoding, sans family, bold series and Italic shape. After `\normalfont`, every aspect of the text is reset to the default one. The default encoding is TU. We can see TU instead of OT1 after `\normalfont`. So is the case with family (default: `\rmfamily`), series (default: `\mdseries`) and shape (default: `\upshape`). This usually is okay, but sometimes it doesn't fit the requirement. E.g., the following might be used with the intention of switching from the IPA font to the text font, but as can be seen, it doesn't really change anything.

```
\documentclass{article}
\usepackage{linguistix-fonts}
\usepackage{linguistix-ipa}
\linguistix{%
  text upright           = {KpRoman-Regular.otf},%
  text upright features  = {Color={green}},%
  ipa upright            = {KpSans-Regular.otf},%
  ipa upright features   = {Color={red}}%
}

\begin{document}
test \lngxipa test \normalfont test
\end{document}
```

The reason for this is the way `\lngxipa` is defined. It resets `\rmdefault`, `\sfdefault` and `\ttdefault` and uses `\normalfont` to initialise this new super font family (see: https://tex.stackexchange.com/a/729805). Setting a 'super' font family effectively changes the behaviour of `\normalfont` permanently. By the way, this is not just something that LinguisTiX has to deal with. This situation may arise whenever one wants to have a font family command that sets all serif, sans serif and monospaced font families. LinguisTiX-nfss is useful in such cases. It introduces the concept of 'super' font family. It shouldn't be confused with LaTeX 2ε's 'meta' font family. It refers to `rm`, `sf` or `tt` in the kernel. Note that, as of now, LaTeX 2ε does *not* provide any public interface to save 'meta' family, as well as, the current encoding, series and shape. This package provides control over these facets. Let's have a look at the macros it provides.

```
\IfEncodingTF     ⋆ {⟨encoding⟩} {⟨true code⟩} {⟨false code⟩}
\IfEncodingT      ⋆ {⟨encoding⟩} {⟨true code⟩}
\IfEncodingF      ⋆ {⟨encoding⟩} {⟨false code⟩}
\CurrentEncoding  ⋆
```

If the current encoding matches with the given ⟨*encoding*⟩, it selects the true branch; false otherwise. The `\CurrentEncoding` macro expands to the current encoding.

```
\IfMetaFamilyTF     ⋆ {⟨meta family⟩} {⟨true code⟩} {⟨false code⟩}
\IfMetaFamilyT      ⋆ {⟨meta family⟩} {⟨true code⟩}
\IfMetaFamilyF      ⋆ {⟨meta family⟩} {⟨false code⟩}
\CurrentMetaFamily  ⋆
```

If the current meta family matches with the given ⟨*meta family*⟩, it selects the true branch; false otherwise. The `\CurrentMetaFamily` macro expands to the current meta family.

```
\IfSuperFamilyTF     ⋆ {⟨super family⟩} {⟨true code⟩} {⟨false code⟩}
\IfSuperFamilyT      ⋆ {⟨super family⟩} {⟨true code⟩}
\IfSuperFamilyF      ⋆ {⟨super family⟩} {⟨false code⟩}
\CurrentSuperFamily  ⋆
```

If the current super family matches with the given ⟨*super family*⟩, it selects the true branch; false otherwise. The `\CurrentSuperFamily` macro expands to the current super family.

```
\IfSeriesTF    ⋆ {⟨series⟩} {⟨true code⟩} {⟨false code⟩}
\IfSeriesT     ⋆ {⟨series⟩} {⟨true code⟩}
\IfSeriesF     ⋆ {⟨series⟩} {⟨false code⟩}
\CurrentSeries ⋆
```

If the current series matches with the given ⟨*series*⟩, it selects the true branch and false otherwise. The `\CurrentSeries` macro expands to the current series.

```
\IfShapeTF    ⋆ {⟨shape⟩} {⟨true code⟩} {⟨false code⟩}
\IfShapeT     ⋆ {⟨shape⟩} {⟨true code⟩}
\IfShapeF     ⋆ {⟨shape⟩} {⟨false code⟩}
\CurrentShape ⋆
```

If the current series matches with the given ⟨*shape*⟩, it selects the true branch and false otherwise. The `\CurrentShape` macro expands to the current shape.

```
\superfontfamily {⟨family id⟩} {⟨rm={⟨rm nfss⟩},sf={⟨sf nfss⟩},tt={⟨tt nfss⟩}⟩}
```

Every super font family has a ⟨*family id*⟩, even the default one (i.e., `default`). This command creates a super family with the given ⟨*family id*⟩s. The ⟨*meta family keys*⟩ argument accepts a list of specific keys, `rm`, `sf` and `tt`. They take the NFSS family names of these meta families as arguments. One may define a font with, say, `\newfontfamily`, pass the `NFSSkeys={⟨key⟩}` option to it and use the ⟨*key*⟩ in the suitable ⟨*meta family key*⟩. Note that using all these keys is *not* mandatory. A super family may have ≤ 3 keys.

```
\softsuperfontfamily     {⟨id⟩}{⟨encoding,family,series,shape⟩}
\softersuperfontfamily   {⟨id⟩}
\softestsuperfontfamily  {⟨id⟩}
```

These commands loads the super font family with the given ⟨*id*⟩. The attributes listed in the second argument are the only choices available. The required super font family is loaded and the listed attributes are reset to the ones that were active before. All the four are not required. The number of attributes may be ≤ 4. The `\softernormalfont` command excludes encoding and reactivates all the other attributes, whereas the `\softestnormalfont` command reactivates all of them.

```
\softnormalfont    {⟨encoding,family,series,shape⟩}
\softernormalfont
\softestnormalfont
```

Similar to `\softsuperfontfamily` and friends, these commands switch back to the default super font family, but reactivate the previously active font attributes. The argument to `\softnormalfont` takes the list of the required font attributes. It can have ≤ 4 values. Now try the following example:

```
\documentclass{article}
\usepackage{linguistix}
\linguistix{%
  text upright features  = {Color={green}},%
  ipa upright features   = {Color={red}}%
}

\begin{document}
test \lngxipa test \softernormalfont test\par
\makeatletter
\sffamily\itshape\bfseries
```

```
\f@family\ | \f@series\ | \f@shape\quad
\softnormalfont{series}
\f@family\ | \f@series\ | \f@shape
\end{document}
```

Better? :-)

LaTeX3 interface for programmers

In this section, we take a look at the public LaTeX3 commands of the bundle. These can be considered stable and can be used in production code.

LinguisTiX-base

Documentation | Implementation

`\lngx_set_keys:n`

`\lngx_keys_set:n ⟨keyval list⟩`

This is the base command for `\linguistix`. It takes a comma separated list of ⟨*keyval list*⟩ and parses it.

LinguisTiX-fonts

Documentation | Implementation

`\g_lngx_old_style_bool`
`\g_lngx_old_style_one_bool`
`\g_lngx_bourbaki_bool`

These are the two booleans that are used to check if the old style numbers, the old style one (i.e., ‘1’) and Bourbaki’s emtpy set symbol (i.e., ‘∅’) is asked by the user.

`\lngx_set_main_font:nn`
`\lngx_set_main_font:ee`
`\lngx_set_sans_font:nn`
`\lngx_set_sans_font:ee`
`\lngx_set_mono_font:nn`
`\lngx_set_mono_font:ee`
`\lngx_set_math_font:nn`
`\lngx_set_math_font:ee`

```
\lngx_set_main_font:nn {⟨features⟩} {⟨font⟩}
\lngx_set_sans_font:nn {⟨features⟩} {⟨font⟩}
\lngx_set_mono_font:nn {⟨features⟩} {⟨font⟩}
\lngx_set_math_font:nn {⟨features⟩} {⟨font⟩}
```

These commands take two arguments, expand them if the `:ee` variant is used. These are wrapper commands around the font-setting commands of fontspec and unicode-math, i.e., `\setmainfont`, `\setsansfont`, `\setmonofont` and `\setmathfont`. The ⟨*features*⟩ are passed to the optional argument and the ⟨*font*⟩ is passed to the mandatory argument of the respective command from the aforementioned list.

LinguisTiX-ipa

Documentation | Implementation

This package provides a few wrapper functions around fontspec’s commands.

`\lngx_set_main_ipa_font:nn`
`\lngx_set_main_ipa_font:ee`
`\lngx_main_ipa:`
`lngx_ipa_rm_nfss`

```
\lngx_set_main_ipa_font:nn {⟨features⟩} {⟨font⟩}
```

These functions set the IPA fonts for the serif variants. The ⟨*font*⟩ is set with ⟨*features*⟩ for the serif IPA. The command to switch to this family is `\lngx_main_ipa:`. It can be accessed with the NFSS family `lngx_ipa_rm_nfss`.

`\lngx_set_sans_ipa_font:nn`
`\lngx_set_sans_ipa_font:ee`
`\lngx_sans_ipa:`
`lngx_ipa_sf_nfss`

`\lngx_set_sans_ipa_font:nn {⟨features⟩} {⟨font⟩}`

These functions set the IPA fonts for the sans variants. The ⟨*font*⟩ is set with ⟨*features*⟩ for the sans IPA. The command to switch to this family is `\lngx_sans_ipa:`. It can be accessed with the NFSS family `lngx_ipa_sf_nfss`.

`\lngx_set_mono_ipa_font:nn`
`\lngx_set_mono_ipa_font:ee`
`\lngx_mono_ipa:`
`lngx_ipa_tt_nfss`

`\lngx_set_mono_ipa_font:nn {⟨features⟩} {⟨font⟩}`

These functions set the IPA fonts for the mono variants. The ⟨*font*⟩ is set with ⟨*features*⟩ for the mono IPA. The command to switch to this family is `\lngx_mono_ipa:`. It can be accessed with the NFSS family `lngx_ipa_nfss_nfss`.

`\lngx_ipa:`
`lngx_ipa`

The `\lngx_ipa:` command loads the super family `lngx_ipa` (see the documentation of LinguisTiX-NFSS. The `\lngx_ipa:` function has a user-side command `\lngxipa` too.

Variables for fonts and features

Now we look at the table that summarises the `tls` that are used by the package for saving serif, sans serif and monospaced fonts and their features. Note that this table also lists the `tls` used by the LinguisTiX-IPA package.

Serif	Sans serif	Monospaced
\g_lngx_text_upright_tl	\g_lngx_text_sans_upright_tl	\g_lngx_text_mono_upright_tl
\g_lngx_ipa_upright_tl	\g_lngx_ipa_sans_upright_tl	\g_lngx_ipa_mono_upright_tl
\g_lngx_text_upright_features_tl	\g_lngx_text_sans_upright_features_tl	\g_lngx_text_mono_upright_features_tl
\g_lngx_ipa_upright_features_tl	\g_lngx_ipa_sans_upright_features_tl	\g_lngx_ipa_mono_upright_features_tl
\g_lngx_text_bold_upright_tl	\g_lngx_text_sans_bold_upright_tl	\g_lngx_text_mono_bold_upright_tl
\g_lngx_ipa_bold_upright_tl	\g_lngx_ipa_sans_bold_upright_tl	\g_lngx_ipa_mono_bold_upright_tl
\g_lngx_text_bold_upright_features_tl	\g_lngx_text_sans_bold_upright_features_tl	\g_lngx_text_mono_bold_upright_features_tl
\g_lngx_ipa_bold_upright_features_tl	\g_lngx_ipa_sans_bold_upright_features_tl	\g_lngx_ipa_mono_bold_upright_features_tl
\g_lngx_text_italic_tl	\g_lngx_text_sans_italic_tl	\g_lngx_text_mono_italic_tl
\g_lngx_ipa_italic_tl	\g_lngx_ipa_sans_italic_tl	\g_lngx_ipa_mono_italic_tl
\g_lngx_text_italic_features_tl	\g_lngx_text_sans_italic_features_tl	\g_lngx_text_mono_italic_features_tl
\g_lngx_ipa_italic_features_tl	\g_lngx_ipa_sans_italic_features_tl	\g_lngx_ipa_mono_italic_features_tl
\g_lngx_text_bold_italic_tl	\g_lngx_text_sans_bold_italic_tl	\g_lngx_text_mono_bold_italic_tl
\g_lngx_ipa_bold_italic_tl	\g_lngx_ipa_sans_bold_italic_tl	\g_lngx_ipa_mono_bold_italic_tl
\g_lngx_text_bold_italic_features_tl	\g_lngx_text_sans_bold_italic_features_tl	\g_lngx_text_mono_bold_italic_features_tl
\g_lngx_ipa_bold_italic_features_tl	\g_lngx_ipa_sans_bold_italic_features_tl	\g_lngx_ipa_mono_bold_italic_features_tl
\g_lngx_text_slanted_tl	\g_lngx_text_sans_slanted_tl	\g_lngx_text_mono_slanted_tl
\g_lngx_ipa_slanted_tl	\g_lngx_ipa_sans_slanted_tl	\g_lngx_ipa_mono_slanted_tl
\g_lngx_text_slanted_features_tl	\g_lngx_text_sans_slanted_features_tl	\g_lngx_text_mono_slanted_features_tl
\g_lngx_ipa_slanted_features_tl	\g_lngx_ipa_sans_slanted_features_tl	\g_lngx_ipa_mono_slanted_features_tl
\g_lngx_text_bold_slanted_tl	\g_lngx_text_sans_bold_slanted_tl	\g_lngx_text_mono_bold_slanted_tl
\g_lngx_ipa_bold_slanted_tl	\g_lngx_ipa_sans_bold_slanted_tl	\g_lngx_ipa_mono_bold_slanted_tl
\g_lngx_text_bold_slanted_features_tl	\g_lngx_text_sans_bold_slanted_features_tl	\g_lngx_text_mono_bold_slanted_features_tl
\g_lngx_ipa_bold_slanted_features_tl	\g_lngx_ipa_sans_bold_slanted_features_tl	\g_lngx_ipa_mono_bold_slanted_features_tl
\g_lngx_text_swash_tl	\g_lngx_text_sans_swash_tl	\g_lngx_text_mono_swash_tl

Continued on the next page…

Serif	Sans serif	Monospaced
\g_lngx_ipa_swash_tl	\g_lngx_ipa_sans_swash_tl	\g_lngx_ipa_mono_swash_tl
\g_lngx_text_swash_features_tl	\g_lngx_text_sans_swash_features_tl	\g_lngx_text_mono_swash_features_tl
\g_lngx_ipa_swash_features_tl	\g_lngx_ipa_sans_swash_features_tl	\g_lngx_ipa_mono_swash_features_tl
\g_lngx_text_bold_swash_tl	\g_lngx_text_sans_bold_swash_tl	\g_lngx_text_mono_bold_swash_tl
\g_lngx_ipa_bold_swash_tl	\g_lngx_ipa_sans_bold_swash_tl	\g_lngx_ipa_mono_bold_swash_tl
\g_lngx_text_bold_swash_features_tl	\g_lngx_text_sans_bold_swash_features_tl	\g_lngx_text_mono_bold_swash_features_tl
\g_lngx_ipa_bold_swash_features_tl	\g_lngx_ipa_sans_bold_swash_features_tl	\g_lngx_ipa_mono_bold_swash_features_tl
\g_lngx_text_small_caps_tl	\g_lngx_text_sans_small_caps_tl	\g_lngx_text_mono_small_caps_tl
\g_lngx_ipa_small_caps_tl	\g_lngx_ipa_sans_small_caps_tl	\g_lngx_ipa_mono_small_caps_tl
\g_lngx_text_small_caps_features_tl	\g_lngx_text_sans_small_caps_features_tl	\g_lngx_text_mono_small_caps_features_tl
\g_lngx_ipa_small_caps_features_tl	\g_lngx_ipa_sans_small_caps_features_tl	\g_lngx_ipa_mono_small_caps_features_tl

End of the table...

Table 2: Variables for fonts and font features provided by LinguisTiX-fonts and LinguisTiX-ipa

LinguisTiX-Logos

Documentation | Implementation

There are only two LaTeX3 functions provided by this package.

`\lngx_logo_font:` This function switches to the New Computer Modern Uncial font family.

`lngx_purple_color` I don't like the default purple colour of the xcolor package (i.e., 🟥). Thus I have created a new colour using l3color module. It can be accessed using this variable. The color looks like: 🟪.

LinguisTiX-nfss

Documentation | Implementation

This subsection discusses the programming interface LinguisTiX-nfss provides.

`\c_lngx_default_rmdefault_tl` ⋆
`\c_lngx_default_sfdefault_tl` ⋆
`\c_lngx_default_ttdefault_tl` ⋆

These `tl`s expand to the default values of the fonts set at the `begindocument/end` hook. These are not supposed to be changed and hence they are set with the `c` prefix.

`\l_lngx_current_encoding_tl` ⋆
`\l_lngx_current_meta_family_tl` ⋆
`\l_lngx_current_super_family_tl` ⋆
`\l_lngx_current_series_tl` ⋆
`\l_lngx_current_shape_tl` ⋆

These `tl`s expand to the current values of encoding, meta family, super family, series and shape respectively. Note that these are updated time to time by the commands that change them (package-internal or LaTeX-internal).

```
\lngx_if_encoding_p:n      ⋆ {⟨encoding⟩}
\lngx_if_encoding:nTF      ⋆ {⟨encoding⟩}{⟨true code⟩}{⟨false code⟩}
\lngx_if_meta_family_p:n   ⋆ {⟨meta font family⟩}
\lngx_if_meta_family:nTF   ⋆ {⟨meta font family⟩}{⟨true code⟩}{⟨false code⟩}
\lngx_if_super_family_p:n  ⋆ {⟨super font family⟩}
\lngx_if_super_family:nTF  ⋆ {⟨super font family⟩}{⟨true code⟩}{⟨false code⟩}
\lngx_if_series_p:n        ⋆ {⟨series⟩}
\lngx_if_series:nTF        ⋆ {⟨series⟩}{⟨true code⟩}{⟨false code⟩}
\lngx_if_shape_p:n         ⋆ {⟨shape⟩}
\lngx_if_shape:nTF         ⋆ {⟨shape⟩}{⟨true code⟩}{⟨false code⟩}
```

```
\lngx_if_meta_family_rm_p: ⋆
\lngx_if_meta_family_rm:TF ⋆ {⟨true code⟩}{⟨false code⟩}
\lngx_if_meta_family_sf_p: ⋆
\lngx_if_meta_family_sf:TF ⋆ {⟨true code⟩}{⟨false code⟩}
\lngx_if_meta_family_tt_p: ⋆
\lngx_if_meta_family_tt:TF ⋆ {⟨true code⟩}{⟨false code⟩}
```

These conditionals select the true branch if the `rm`, `sf`, `tt` families (respectively) are active, false otherwise.

```
\lngx_if_series_md_p: ⋆
\lngx_if_series_md:TF ⋆ {⟨true code⟩}{⟨false code⟩}
\lngx_if_series_bf_p: ⋆
\lngx_if_series_bf:TF ⋆ {⟨true code⟩}{⟨false code⟩}
```

These conditionals select the true branch if the `md`, `bf` series (respectively) are active, false otherwise.

```
\lngx_if_shape_up_p:  ⋆
\lngx_if_shape_up:TF  ⋆ {⟨true code⟩}{⟨false code⟩}
\lngx_if_shape_it_p:  ⋆
\lngx_if_shape_it:TF  ⋆ {⟨true code⟩}{⟨false code⟩}
\lngx_if_shape_sc_p:  ⋆
\lngx_if_shape_sc:TF  ⋆ {⟨true code⟩}{⟨false code⟩}
\lngx_if_shape_ssc_p: ⋆
\lngx_if_shape_ssc:TF ⋆ {⟨true code⟩}{⟨false code⟩}
\lngx_if_shape_sl_p:  ⋆
\lngx_if_shape_sl:TF  ⋆ {⟨true code⟩}{⟨false code⟩}
\lngx_if_shape_sw_p:  ⋆
\lngx_if_shape_sw:TF  ⋆ {⟨true code⟩}{⟨false code⟩}
\lngx_if_shape_ulc_p: ⋆
\lngx_if_shape_ulc:TF ⋆ {⟨true code⟩}{⟨false code⟩}
```

These conditionals select the true branch if the `up`, `it`, `sc`, `ssc`, `sl`, `sw`, `ulc` shapes (respectively) are active, false otherwise.

```
\lngx_super_font_family:nn {⟨family id⟩} {⟨rm={⟨rm nfss⟩},sf={⟨sf nfss⟩},tt={⟨tt nfss⟩}⟩}
```

This function takes an ⟨*id*⟩ and sets the `rm`, `sf`, `tt` values as requested by the user and creates a super font family.

```
\lngx_soft_super_font_family:nn   {⟨id⟩}{⟨encoding,family,series,shape⟩}
\lngx_softer_super_font_family:n  {⟨id⟩}
\lngx_softest_super_font_family:n {⟨id⟩}
```

The `\lngx_soft_super_font_family:nn` sets super family marked by the ⟨*id*⟩ and reactivates the currently active font attributes listed in the second argument. The other two do the same, but without the list. the `softer` one omits the encoding and the `softest` one reactivate all of them.

```
\lngx_soft_normal_font:n  {⟨id⟩}
\lngx_softer_normal_font:
\lngx_softest_normal_font:
```

Quite similar to the soft super family functions, these ones set the default font family and reactivate the font attributes. The `soft` one sets the attributes listed in the argument. The `softer` one omits encoding and reactivates the rest and the `softest` one reactivates all.

Implementation

In this section the code of this bundle is documented. Each package in the bundle is documented in a separate subsection.

LinguisTiX

Provide the package with its basic information.

```
⟨*package⟩
\ProvidesExplPackage{linguistix}
                    {2025-05-20}
                    {v0.1b}
                    {%
                      The ‘LinguisTiX’ bundle: Enhanced
                      support for linguistics.%
                    }
```

When one loads LinguisTiX, all the packages of the bundle are loaded automatically. That's the only content of the umbrella package LinguisTiX. All the packages are loaded conditionally (i.e., only if not loaded already).

```

\IfPackageLoadedF { linguistix-base } {
  \RequirePackage { linguistix-base }
}
\IfPackageLoadedF { linguistix-fonts } {
  \RequirePackage { linguistix-fonts }
}
\IfPackageLoadedF { linguistix-ipa } {
  \RequirePackage { linguistix-ipa }
}
\IfPackageLoadedF { linguistix-logos } {
  \RequirePackage { linguistix-logos }
}
\IfPackageLoadedF { linguistix-nfss } {
  \RequirePackage { linguistix-nfss }
}
⟨/package⟩
```

Set the essentials of the package.

```
⟨*base⟩
\ProvidesExplPackage{linguistix-base}
                    {2025-05-20}
                    {v0.1b}
                    {%
                      The base package of the 'LinguisTiX'
                      bundle.%
                    }
```

`\lngx_set_keys:n` I use the l3keys module of LaTeX3 for creating the key-values used in this bundle. In order to get a singleton parser for all the packages of the bundle, I have create this parsing command that is used throughout the bundle.

```

\cs_new_protected:Npn \lngx_set_keys:n #1 {
  \keys_set:nn { lngx _ keys } { #1 }
}
```

(*End of definition for* `\lngx_set_keys:n`*. This function is documented on page 12.*)

`\linguistix` I equate this command with a user-side macro here and end the LinguisTiX-base package.

```

\cs_gset_eq:NN \linguistix \lngx_set_keys:n
⟨/base⟩
```

(*End of definition for* `\linguistix`*. This function is documented on page 5.*)

Package essentials first.

```
⟨*font⟩
\ProvidesExplPackage{linguistix-fonts}
                    {2025-05-20}
                    {v0.1b}
                    {%
                      The font-assistant package of the
                      ‘LinguisTiX’ bundle.%
                    }
```

I load LinguisTiX-base and unicode-math (if they are not already loaded).

```

\IfPackageLoadedF { linguistix-base } {
  \RequirePackage { linguistix-base }
}

\IfPackageLoadedF { unicode-math } {
  \RequirePackage { unicode-math }
}
```

old style numbers
\g_lngx_old_style_bool
old style one
\g_lngx_old_style_one_bool
bourbaki's empty set
\g_lngx_bourbaki_bool

I use the .bool_gset:N key-type of l3keys for developing these boolean keys.

```

\keys_define:nn { lngx _ keys } {
  old~ style~ numbers
  .bool_gset:N           = {
    \g_lngx_old_style_bool
  },
  old~ style~ one
  .bool_gset:N           = {
    \g_lngx_old_style_one_bool
  },
  bourbaki's~ empty~ set
  .bool_gset:N           = {
    \g_lngx_bourbaki_bool
  }
}
```

(*End of definition for* `old style numbers` *and others. These functions are documented on page 5.*)

`text upright`
`text upright features`
`text bold upright`
`text bold upright features`
`text italic`
`text italic features`
`text bold italic`
`text bold italic features`
`text slanted`
`text slanted features`
`text bold slanted`
`text bold slanted features`
`text swash`
`text swash features`
`text bold swash`
`text bold swash features`
`text small caps`
`text small caps features`
`\g_lngx_text_upright_tl`
`\g_lngx_text_upright_features_tl`
`\g_lngx_text_bold_upright_tl`
`\g_lngx_text_bold_upright_features_tl`
`\g_lngx_text_italic_tl`
`\g_lngx_text_italic_features_tl`
`\g_lngx_text_bold_italic_tl`
`\g_lngx_text_bold_italic_features_tl`
`\g_lngx_text_slanted_tl`
`\g_lngx_text_slanted_features_tl`
`\g_lngx_text_bold_slanted_tl`
`\g_lngx_text_bold_slanted_features_tl`
`\g_lngx_text_swash_tl`
`\g_lngx_text_swash_features_tl`
`\g_lngx_text_bold_swash_tl`
`\g_lngx_text_bold_swash_features_tl`
`\g_lngx_text_small_caps_tl`
`\g_lngx_text_small_caps_features_tl`

I save the names of the fonts in `tl` variables. This section creates the keys for serif text fonts. All these keys have a common pattern of code. For the convenience of maintenance, I have created a comma-separated-list and used the elements of this list inside the common code. (See: https://topanswers.xyz/tex?q=8074#a7689.)

```

\clist_map_inline:nn {
  upright,
  bold~ upright,
  italic,
  bold~ italic,
  slanted,
  bold~ slanted,
  swash,
  bold~ swash,
  small~ caps
} {
```

The key-names can contain spaces, but the variables can't. I set a temporary variable and convert the spaces into underscores. Note that `#1` means the elements of the `clist` here.

```
  \tl_set:Nn \l_tmpa_tl { #1 }
  \tl_replace_all:Nnn \l_tmpa_tl { ~ } { _ }
  \tl_gclear_new:c {
    g _ lngx _ text _ \l_tmpa_tl _ features _ tl
  }
```

All the keys here are prefixed with the word `text` in order to distinguish them from the keys provided by the LinguisTiX-ipa package. The argument of these keys should be expanded for which I use `.tl_gset_e:c` type of l3keys.

```
  \keys_define:nn { lngx _ keys } {
    text~ #1
    .tl_gset_e:c          = {
      g _ lngx _ text _ \l_tmpa_tl _ tl
    },
```

Each of these keys is followed by its respective `features` key which is supposed to take an appending argument. The `.tl`-type keys don't support this. I create this key with the `.code:n` type. Like before, first I set a temporary variable for space-to-underscore conversion, use it with the `\tl_put_right:ce` call for appending.

```
    text~ #1~ features
    .code:n               = {
      \tl_set:Nn \l_tmpb_tl { #1 }
      \tl_replace_all:Nnn \l_tmpb_tl { ~ } { _ }
      \tl_put_right:ce {
        g _ lngx _ text _ \l_tmpb_tl _ features _ tl
      } { ##1 , }
```

Lastly, we clear the temporary `tl`s.

```
      \tl_clear:N \l_tmpb_tl
    }
  }
  \tl_clear:N \l_tmpa_tl
}
```

(*End of definition for* ***`text upright`*** *and others. These functions are documented on page 7.*)

text sans upright
text sans upright features
text sans bold upright
text sans bold upright features
text sans italic
text sans italic features
text sans bold italic
text sans bold italic features
text sans slanted
text sans slanted features
text sans bold slanted
text sans bold slanted features
text sans swash
text sans swash features
text sans bold swash
text sans bold swash features
text sans small caps
text sans small caps features
\g_lngx_text_sans_upright_tl
\g_lngx_text_sans_upright_features_tl
\g_lngx_text_sans_bold_upright_tl
\g_lngx_text_sans_bold_upright_features_tl
\g_lngx_text_sans_italic_tl
\g_lngx_text_sans_italic_features_tl
\g_lngx_text_sans_bold_italic_tl
\g_lngx_text_sans_bold_italic_features_tl
\g_lngx_text_sans_slanted_tl
\g_lngx_text_sans_slanted_features_tl
\g_lngx_text_sans_bold_slanted_tl
\g_lngx_text_sans_bold_slanted_features_tl
\g_lngx_text_sans_swash_tl
\g_lngx_text_sans_swash_features_tl
\g_lngx_text_sans_bold_swash_tl
\g_lngx_text_sans_bold_swash_features_tl
\g_lngx_text_sans_small_caps_tl
\g_lngx_text_sans_small_caps_features_tl

With this same mechanism, the keys for sans serif fonts are developed.

```

\clist_map_inline:nn {
  upright,
  bold~ upright,
  italic,
  bold~ italic,
  slanted,
  bold~ slanted,
  swash,
  bold~ swash,
  small~ caps
} {
  \tl_set:Nn \l_tmpa_tl { #1 }
  \tl_replace_all:Nnn \l_tmpa_tl { ~ } { _ }
  \tl_gclear_new:c {
    g _ lngx _ text _ sans _ \l_tmpa_tl _ features _ tl
  }
  \keys_define:nn { lngx _ keys } {
    text~ sans~ #1
    .tl_gset_e:c            = {
      g _ lngx _ text _ sans _ \l_tmpa_tl _ tl
    },
    text~ sans~ #1~ features
    .code:n                 = {
      \tl_set:Nn \l_tmpb_tl { #1 }
      \tl_replace_all:Nnn \l_tmpb_tl { ~ } { _ }
      \tl_put_right:ce {
        g _ lngx _ text _ sans _ \l_tmpb_tl _ features _ tl
      } { ##1 , }
      \tl_clear:N \l_tmpb_tl
    }
  }
  \tl_clear:N \l_tmpa_tl
}
```

(*End of definition for* **`text sans upright`** *and others. These functions are documented on page 7.*)

text mono upright
text mono upright features
text mono bold upright
text mono bold upright features
text mono italic
text mono italic features
text mono bold italic
text mono bold italic features
text mono slanted
text mono slanted features
text mono bold slanted
text mono bold slanted features
text mono swash
text mono swash features
text mono bold swash
text mono bold swash features
text mono small caps
text mono small caps features
\g_lngx_text_mono_upright_tl
\g_lngx_text_mono_upright_features_tl
\g_lngx_text_mono_bold_upright_tl
\g_lngx_text_mono_bold_upright_features_tl
\g_lngx_text_mono_italic_tl
\g_lngx_text_mono_italic_features_tl
\g_lngx_text_mono_bold_italic_tl
\g_lngx_text_mono_bold_italic_features_tl
\g_lngx_text_mono_slanted_tl
\g_lngx_text_mono_slanted_features_tl
\g_lngx_text_mono_bold_slanted_tl
\g_lngx_text_mono_bold_slanted_features_tl
\g_lngx_text_mono_swash_tl
\g_lngx_text_mono_swash_features_tl
\g_lngx_text_mono_bold_swash_tl
\g_lngx_text_mono_bold_swash_features_tl
\g_lngx_text_mono_small_caps_tl
\g_lngx_text_mono_small_caps_features_tl

Here, with the same setup, I develop the keys for monospaced fonts.

```

\clist_map_inline:nn {
  upright,
  bold~ upright,
  italic,
  bold~ italic,
  slanted,
  bold~ slanted,
  swash,
  bold~ swash,
  small~ caps
} {
  \tl_set:Nn \l_tmpa_tl { #1 }
  \tl_replace_all:Nnn \l_tmpa_tl { ~ } { _ }
  \tl_gclear_new:c {
    g _ lngx _ text _ mono _ \l_tmpa_tl _ features _ tl
  }
  \keys_define:nn { lngx _ keys } {
    text~ mono~ #1
    .tl_gset_e:c            = {
      g _ lngx _ text _ mono _ \l_tmpa_tl _ tl
    },
    text~ mono~ #1~ features
    .code:n                 = {
      \tl_set:Nn \l_tmpb_tl { #1 }
      \tl_replace_all:Nnn \l_tmpb_tl { ~ } { _ }
      \tl_put_right:ce {
        g _ lngx _ text _ mono _ \l_tmpb_tl _ features _ tl
      } { ##1 , }
      \tl_clear:N \l_tmpb_tl
    }
  }
  \tl_clear:N \l_tmpa_tl
}
```

(*End of definition for* **`text mono upright`** *and others. These functions are documented on page 8.*)

math
math features
math bold
math bold features

The following are the keys set for math. They use the same mechanism as before.

```

\keys_define:nn { lngx _ keys } {
  math
  .tl_gset_e:c           = {
    g _ lngx _ math _ tl
  },
  math~ features
  .tl_gset_e:c           = {
    g _ lngx _ math _ features _ tl
  },
  math~ bold
  .tl_gset_e:c           = {
    g _ lngx _ math _ bold _ tl
  },
  math~ bold~ features
  .code:n                = {
    \tl_put_right:ce {
      g _ lngx _ math _ bold _ features _ tl
    } { #1 }
  }
}
```

(*End of definition for* `math` *and others. These functions are documented on page 6.*)

newcm

This key is of type `.meta:n`. It sets certain other keys that enable the New Computer Modern fonts in all serif, serif and monospaced families.

```

\keys_define:nn { lngx _ keys } {
  newcm
  .meta:n                = {
    text~
    upright                = {
      NewCM10-Book.otf
    },
    text~
    bold~ upright          = {
      NewCM10-Bold.otf
    },
    text~
    italic                 = {
      NewCM10-BookItalic.otf
    },
    text~
    bold~ italic           = {
      NewCM10-BoldItalic.otf
    },
    math                   = {
      NewCMMath-Book.otf
    },
    math~ bold             = {
      NewCMMath-Bold.otf
    },
    text~
    sans~ upright          = {
      NewCMSans10-Book.otf
    },
    text~
    sans~ bold~ upright    = {
      NewCMSans10-Bold.otf
    },
    text~
    sans~ italic           = {
      NewCMSans10-BookOblique.otf
    },
    text~
    sans~ bold~ italic     = {
      NewCMSans10-BoldOblique.otf
    },
    text~
    mono~ upright          = {
      NewCMMono10-Book.otf
    },
    text~
    mono~ bold~ upright    = {
      NewCMMono10-Bold.otf
    },
    text~
    mono~ italic           = {
      NewCMMono10-BookItalic.otf
    },
    text~
    mono~ bold~ italic     = {
      NewCMMono10-BoldOblique.otf
    }
  }
}
```

(*End of definition for* `newcm`. *This function is documented on page 6.*)

`newcm sans` This is a `.meta:n` key that sets the default fonts to the sans family.

```

\keys_define:nn { lngx _ keys } {
  newcm~ sans
  .meta:n                = {
    text~
    upright                = {
      NewCMSans10-Book.otf
    },
    text~
    bold upright           = {
      NewCMSans10-Bold.otf
    },
    text~
    italic                 = {
      NewCMSans10-BookOblique.otf
    },
    text~
    bold~ italic           = {
      NewCMSans10-BoldOblique.otf
    }
  }
}
```

(*End of definition for* `newcm sans`. *This function is documented on page 6.*)

`newcm mono` This is a `.meta:n` key that sets the default fonts to the monospaced family.

```

\keys_define:nn { lngx _ keys } {
  newcm~ mono
  .meta:n                = {
    text~
    upright                = {
      NewCMMono10-Book.otf
    },
    text~
    bold upright           = {
      NewCMMono10-Bold.otf
    },
    text~
    italic                 = {
      NewCMMono10-BookItalic.otf
    },
    text~
    bold~ italic           = {
      NewCMMono10-BoldOblique.otf
    }
  }
}
```

(*End of definition for* `newcm mono`. *This function is documented on page 6.*)

`newcm regular` This is a `.meta:n` key that sets the default fonts to the regular variant of the New Computer Modern family.

```

\keys_define:nn { lngx _ keys } {
  newcm~ regular
  .meta:n                = {
    text~
    upright                = {
      NewCM10-Regular.otf
    },
    text~
    bold~ upright          = {
      NewCM10-Bold.otf
    },
    text~
    italic                 = {
      NewCM10-Italic.otf
    },
    text~
    bold~ italic           = {
      NewCM10-BoldItalic.otf
    },
    math                  = {
      NewCMMath-Regular.otf
    },
    math~ bold            = {
      NewCMMath-Bold.otf
    },
    text~
    sans~ upright         = {
      NewCMSans10-Regular.otf
    },
    text~
    sans~ bold~ upright   = {
      NewCMSans10-Bold.otf
    },
    text~
    sans~ italic          = {
      NewCMSans10-Oblique.otf
    },
    text~
    sans~ bold~ italic    = {
      NewCMSans10-BoldOblique.otf
    },
    text~
    mono~ upright         = {
      NewCMMono10-Regular.otf
    },
    text~
    mono~ bold~ upright   = {
      NewCMMono10-Bold.otf
    },
    text~
    mono~ italic          = {
      NewCMMono10-Italic.otf
    },
    text~
    mono~ bold~ italic    = {
      NewCMMono10-Bold.otf
    }
  }
}
```

(*End of definition for* `newcm regular`*. This function is documented on page 6.*)

`newcm regular sans` This is a `.meta:n` key that sets the default fonts to the regular sans variant of the New Computer Modern family.

```

\keys_define:nn { lngx _ keys } {
  newcm~ regular~ sans
  .meta:n                = {
    text~
    upright                = {
      NewCMSans10-Regular.otf
    },
    text~
    bold~ upright          = {
      NewCMSans10-Bold.otf
    },
    text~
    italic                 = {
      NewCMSans10-Oblique.otf
    },
    text~
    bold~ italic           = {
      NewCMSans10-BoldOblique.otf
    }
  }
}
```

(*End of definition for* `newcm regular sans`*. This function is documented on page 6.*)

`newcm regular mono` This is a `.meta:n` key that sets the default fonts to the regular monospaced variant of the New Computer Modern family.

```

\keys_define:nn { lngx _ keys } {
  newcm~ regular~ mono
  .meta:n                = {
    text~
    upright                = {
      NewCMMono10-Regular.otf
    },
    text~
    bold~ upright          = {
      NewCMMono10-Bold.otf
    },
    text~
    italic                 = {
      NewCMMono10-Italic.otf
    },
    text~
    bold~ italic           = {
      NewCMMono10-Bold.otf
    }
  }
}
```

(*End of definition for* `newcm regular mono`*. This function is documented on page 6.*)

By default, we load the `newcm` key that loads all the New Computer Modern fonts in its book variant.

```

\lngx_set_keys:n {
  newcm,
```

Then we load the `bourbaki's empty set` boolean. This gets read later while setting the math font.

```
  bourbaki's~ empty~ set,
```

Lastly we load the `old style numbers` boolean.

```
  old~ style~ numbers
}
```

We need HarfBuzz renderer whenever LuaLaTeX is used. For that we add the required feature to the feature-lists of all the fonts.

```

\sys_if_engine_luatex:T {
  \lngx_set_keys:n {
    text~
    upright~ features      = {
      Renderer               = { HarfBuzz }
    },
    text~ sans~
    upright~ features      = {
      Renderer               = { HarfBuzz }
    },
    text~ mono~
    upright~ features      = {
      Renderer               = { HarfBuzz }
    }
  }
}
```

`\lngx_set_main_font:nn`
`\lngx_set_sans_font:nn`
`\lngx_set_mono_font:nn`
`\lngx_set_math_font:nn`

Since I use many conditionals and values while setting the fonts, here, I develop a few wrappers around the font commands. The `\cs_generate_variant:Nn` line comes in handy to generate the argument-expanding versions of the default wrapper-commands.

```

\cs_new_protected:Npn \lngx_set_main_font:nn #1#2 {
  \setmainfont [ #1 ] { #2 }
}

\cs_new_protected:Npn \lngx_set_sans_font:nn #1#2 {
  \setsansfont [ #1 ] { #2 }
}

\cs_new_protected:Npn \lngx_set_mono_font:nn #1#2 {
  \setmonofont [ #1 ] { #2 }
}

\cs_new_protected:Npn \lngx_set_math_font:nn #1#2 {
  \setmathfont [ #1 ] { #2 }
}

\cs_generate_variant:Nn \lngx_set_main_font:nn { ee }
\cs_generate_variant:Nn \lngx_set_sans_font:nn { ee }
\cs_generate_variant:Nn \lngx_set_mono_font:nn { ee }
\cs_generate_variant:Nn \lngx_set_math_font:nn { ee }
```

(*End of definition for* `\lngx_set_main_font:nn` *and others. These functions are documented on page 12.*)

Now I start the pre-`begindocument` hook. New Computer Modern comes in two sizes for some shapes, 8 and 10. They matter for micro-typographic perfection. I have a little complicated checking for providing support for the entire New Computer Modern family.

First I check if the font that is set to be the main font is New Computer Modern or not. For that, searching for the keyword NewCM suffices.

```

\hook_gput_code:nnn { begindocument / before } { . } {
  \tl_if_in:cnT {
    g _ lngx _ text _ upright _ tl
  } { NewCM } {
```

The Book weight of New Computer Modern consistently has Book in all its font-file-names. I test over that to distinguish it from the regular weight. In the true branch of it, I add the size features as required by fontspec for setting size-specific fonts.

```
    \tl_if_in:cnTF {
      g _ lngx _ text _ upright _ tl
    } { Book } {
      \lngx_set_keys:n {
        text~
        upright~ features      = {
          SizeFeatures           = {
            {
              Size                   = {-8},
              Font                   = {
                NewCM08-Book.otf
              }
            },
            {
              Size                   = {8-},
              Font                   = {
                NewCM10-Book.otf
              }
            }
          }
        }
      }
```

In the false branch, the same settings are used for the regular variant.

```
    } {
      \lngx_set_keys:n {
        text~
        upright~ features      = {
          SizeFeatures           = {
            {
              Size                   = {-8},
              Font                   = {
                NewCM08-Regular.otf
              }
            },
            {
              Size                   = {8-},
              Font                   = {
                NewCM10-Regular.otf
              }
            }
          }
        }
      }
    }
  }
```

When the `newcm sans` key is loaded, sans fonts are set as main fonts. All the sans variants have `NewCMSans` in their file-names. I repeat the same check for this case. This is on purpose loaded later, so that the features loaded by the previous snippet are overridden by this one in case the main font is sans[2].

```
  \tl_if_in:cnT {
    g _ lngx _ text _ upright _ tl
  } { NewCMSans } {
    \tl_if_in:cnTF {
      g _ lngx _ text _ upright _ tl
    } { Book } {
      \lngx_set_keys:n {
        text~
        upright~ features      = {
          SizeFeatures           = {
            {
              Size                   = {-8},
              Font                   = {
                NewCMSans08-Book.otf
              }
            },
            {
              Size                   = {8-},
              Font                   = {
                NewCMSans10-Book.otf
              }
            }
          }
        }
      }
    } {
      \lngx_set_keys:n {
        text~
        upright~ features      = {
          SizeFeatures           = {
            {
              Size                   = {-8},
              Font                   = {
                NewCMSans08-Regular.otf
              }
            },
            {
              Size                   = {8-},
              Font                   = {
                NewCMSans10-Regular.otf
              }
            }
          }
        }
```

[2]The test for `NewCM` matches with fonts that have `NewCMSans` too and this is the fastest test I could think of. Suggestions for alternative methods are highly welcome.

```
      }
    }
  }
```
Italic fonts also have this size variant, so here we repeat the same checks for Italic.
```
  \tl_if_in:cnT {
    g _ lngx _ text _ italic _ tl
  } { NewCM } {
    \tl_if_in:cnTF {
      g _ lngx _ text _ italic _ tl
    } { Book } {
      \lngx_set_keys:n {
        text~
        italic~ features       = {
          SizeFeatures           = {
            {
              Size                   = {-8},
              Font                   = {
                NewCM08-BookItalic.otf
              }
            },
            {
              Size                   = {8-},
              Font                   = {
                NewCM10-BookItalic.otf
              }
            }
          }
        }
      }
    } {
      \lngx_set_keys:n {
        text~
        italic~ features       = {
          SizeFeatures           = {
            {
              Size                   = {-8},
              Font                   = {
                NewCM08-Italic.otf
              }
            },
            {
              Size                   = {8-},
              Font                   = {
                NewCM08-Italic.otf
              }
            }
          }
        }
      }
    }
  }
  \tl_if_in:cnT {
    g _ lngx _ text _ italic _ tl
  } { NewCMSans } {
    \tl_if_in:cnTF {
      g _ lngx _ text _ italic _ tl
    } { Book } {
      \lngx_set_keys:n {
        text~
        italic~ features       = {
          SizeFeatures           = {
            {
              Size                   = {-8},
              Font                   = {
                NewCMSans08-BookOblique.otf
              }
            },
            {
              Size                   = {8-},
              Font                   = {
                NewCMSans10-BookOblique.otf
              }
            }
          }
        }
      }
    } {
      \lngx_set_keys:n {
        text~
        italic~ features       = {
          SizeFeatures           = {
            {
              Size                   = {-8},
              Font                   = {
                NewCMSans08-Oblique.otf
              }
            },
            {
              Size                   = {8-},
              Font                   = {
                NewCMSans08-Oblique.otf
              }
            }
          }
        }
      }
    }
  }
```

By default, I have set sans fonts from this family in a different set of variables. I repeat the same checks again for those variables. These coexist with the serif variables.

```
  \tl_if_in:cnT {
    g _ lngx _ text _ sans _ upright _ tl
  } { NewCMSans } {
    \tl_if_in:cnTF {
      g _ lngx _ text _ upright _ tl
    } { Book } {
      \lngx_set_keys:n {
```

```
      text~ sans~
      upright~ features      = {
        SizeFeatures           = {
          {
            Size                   = {-8},
            Font                   = {
              NewCMSans08-Book.otf
            }
          },
          {
            Size                   = {8-},
            Font                   = {
              NewCMSans10-Book.otf
            }
          }
        }
      }
    }
  } {
    \lngx_set_keys:n {
      text~ sans~
      upright~ features      = {
        SizeFeatures           = {
          {
            Size                   = {-8},
            Font                   = {
              NewCMSans08-Regular.otf
            }
          },
          {
            Size                   = {8-},
            Font                   = {
              NewCMSans10-Regular.otf
            }
          }
        }
      }
    }
  }
}
\tl_if_in:cnT {
  g _ lngx _ text _ sans _ italic _ tl
} { NewCMSans } {
  \tl_if_in:cnTF {
    g _ lngx _ text _ italic _ tl
  } { Book } {
    \lngx_set_keys:n {
      text~ sans~
      italic~ features       = {
        SizeFeatures           = {
          {
            Size                   = {-8},
            Font                   = {
              NewCMSans08-BookOblique.otf
            }
          },
          {
            Size                   = {8-},
            Font                   = {
              NewCMSans10-BookOblique.otf
            }
          }
        }
      }
    }
  } {
    \lngx_set_keys:n {
      text~ sans~
      italic~ features       = {
        SizeFeatures           = {
          {
            Size                   = {-8},
            Font                   = {
              NewCMSans08-Oblique.otf
            }
          },
          {
            Size                   = {8-},
            Font                   = {
              NewCMSans10-Oblique.otf
            }
          }
        }
      }
    }
  }
}
```

Now I load the fonts and features. I am using variables that need to be loaded at the end so that all the intermediate user-given changes are also read and considered. Every sub-font (e.g., bold font, Italic font) is stored in a `tl`. Here I save the features as required by fontspec in LinguisTiX feature keys.

```
\lngx_set_keys:n {
  text~
  upright~ features      = {
    UprightFont          = {
      \g_lngx_text_upright_tl
    },
    UprightFeatures        = {
      \g_lngx_text_upright_features_tl
    },
    ItalicFont           = {
      \g_lngx_text_italic_tl
    },
    ItalicFeatures         = {
      \g_lngx_text_italic_features_tl
    },
    BoldFont             = {
      \g_lngx_text_bold_upright_tl
    },
    BoldFeatures           = {
      \g_lngx_text_bold_upright_features_tl
    },
    BoldItalicFont       = {
      \g_lngx_text_bold_italic_tl
    },
    BoldItalicFeatures     = {
      \g_lngx_text_bold_italic_features_tl
    },
```

The New Computer Modern fonts don't have the following shapes, but other fonts may have them, so I load the variables conditionally (i.e., only if they are not empty).

```
    \tl_if_empty:cF {
      g _ lngx _ text _ slanted _ tl
    } {
      SlantedFont            = {
        \g_lngx_text_slanted_tl
      },
      \tl_if_empty:cF {
        g _ lngx _ text _ slanted _ features _ tl
      } {
        SlantedFeatures        = {
          \g_lngx_text_slanted_features_tl
        },
      }
    }
    \tl_if_empty:cF {
      g _ lngx _ text _ bold _ slanted _ tl
    } {
      BoldSlantedFont        = {
        \g_lngx_text_bold_slanted_tl
      },
      BoldSlantedFeatures    = {
        \g_lngx_text_bold_slanted_features_tl
      },
    }
    \tl_if_empty:cF {
      g _ lngx _ text _ swash _ tl
    } {
      SwashFont              = {
        \g_lngx_text_swash_tl
      },
      SwashFeatures          = {
        \g_lngx_text_swash_features_tl
      },
    }
    \tl_if_empty:cF {
      g _ lngx _ text _ bold _ swash _ tl
    } {
      BoldSwashFont          = {
        \g_lngx_text_bold_swash_tl
      },
      BoldSwashFeatures      = {
        \g_lngx_text_bold_swash_features_tl
      },
    }
    \tl_if_empty:cF {
      g _ lngx _ text _ small _ caps _ tl
    } {
      SmallCapsFont          = {
        \g_lngx_text_small_caps_tl
      },
      SmallCapsFeatures      = {
        \g_lngx_text_small_caps_features_tl
      }
    }
  },
```

Exactly like serif fonts, I develop the feature-set for sans and mono fonts.

```
  text~ sans~
  upright~ features      = {
    UprightFont            = {
      \g_lngx_text_sans_upright_tl
    },
    UprightFeatures        = {
      \g_lngx_text_sans_upright_features_tl
    },
    BoldFont               = {
      \g_lngx_text_sans_bold_upright_tl
    },
    BoldFeatures           = {
      \g_lngx_text_sans_bold_upright_features_tl
    },
    ItalicFont             = {
      \g_lngx_text_sans_italic_tl
    },
    ItalicFeatures         = {
      \g_lngx_text_sans_italic_features_tl
    },
    BoldItalicFont         = {
      \g_lngx_text_sans_bold_italic_tl
    },
    BoldItalicFeatures     = {
      \g_lngx_text_sans_bold_italic_features_tl
    },
    \tl_if_empty:cF {
      g _ lngx _ text _ sans _ slanted _ tl
    } {
      SlantedFont            = {
        \g_lngx_text_sans_slanted_tl
      },
      SlantedFeatures        = {
        \g_lngx_text_sans_slanted_features_tl
      },
    }
    \tl_if_empty:cF {
      g _ lngx _ text _ sans _ bold _ slanted _ tl
    } {
      BoldSlantedFont        = {
        \g_lngx_text_sans_bold_slanted_tl
      },
      BoldSlantedFeatures    = {
        \g_lngx_text_sans_bold_slanted_features_tl
      },
    }
    \tl_if_empty:cF {
      g _ lngx _ text _ sans _ swash _ tl
    } {
      SwashFont              = {
        \g_lngx_text_sans_swash_tl
      },
      SwashFeatures          = {
        \g_lngx_text_sans_swash_features_tl
      },
    }
    \tl_if_empty:cF {
      g _ lngx _ text _ sans _ bold _ swash _ tl
    } {
      BoldSwashFont          = {
        \g_lngx_text_sans_bold_swash_tl
      },
      BoldSwashFeatures      = {
        \g_lngx_text_sans_bold_swash_features_tl
      },
    }
    \tl_if_empty:cF {
      g _ lngx _ text _ sans _ small _ caps _ tl
    } {
      SmallCapsFont          = {
        \g_lngx_text_sans_small_caps_tl
      },
      SmallCapsFeatures      = {
        \g_lngx_text_sans_small_caps_features_tl
      }
    }
  },
  text~ mono~
  upright~ features      = {
    UprightFont            = {
      \g_lngx_text_mono_upright_tl
    },
    UprightFeatures        = {
      \g_lngx_text_mono_upright_features_tl
    },
    BoldFont               = {
      \g_lngx_text_mono_bold_upright_tl
    },
    BoldFeatures           = {
      \g_lngx_text_mono_bold_upright_features_tl
    },
    ItalicFont             = {
```

```
      \g_lngx_text_mono_italic_tl
    },
    ItalicFeatures         = {
      \g_lngx_text_mono_italic_features_tl
    },
    BoldItalicFont         = {
      \g_lngx_text_mono_bold_italic_tl
    },
    BoldItalicFeatures     = {
      \g_lngx_text_mono_bold_italic_features_tl
    },
    \tl_if_empty:cF {
      g _ lngx _ text _ mono _ slanted _ tl
    } {
      SlantedFont            = {
        \g_lngx_text_mono_slanted_tl
      },
      SlantedFeatures        = {
        \g_lngx_text_mono_slanted_features_tl
      },
    }
    \tl_if_empty:cF {
      g _ lngx _ text _ mono _ bold _ slanted _ tl
    } {
      BoldSlantedFont        = {
        \g_lngx_text_mono_bold_slanted_tl
      },
      BoldSlantedFeatures    = {
        \g_lngx_text_mono_bold_slanted_features_tl
      },
    }
    \tl_if_empty:cF {
      g _ lngx _ text _ mono _ swash _ tl
    } {
      SwashFont              = {
        \g_lngx_text_mono_swash_tl
      },
      SwashFeatures          = {
        \g_lngx_text_mono_swash_features_tl
      },
    }
    \tl_if_empty:cF {
      g _ lngx _ text _ mono _ bold _ swash _ tl
    } {
      BoldSwashFont          = {
        \g_lngx_text_mono_bold_swash_tl
      },
      BoldSwashFeatures      = {
        \g_lngx_text_mono_bold_swash_features_tl
      },
    }
    \tl_if_empty:cF {
      g _ lngx _ text _ mono _ small _ caps _ tl
    } {
        SmallCapsFont          = {
          \g_lngx_text_mono_small_caps_tl
        },
        SmallCapsFeatures      = {
          \g_lngx_text_mono_small_caps_features_tl
        }
      }
    }
  }
  \bool_if:NT \g_lngx_old_style_bool {
    \lngx_set_keys:n {
      text~
      upright~ features      = {
        Numbers                = { OldStyle }
      },
      text~ sans~
      upright~ features      = {
        Numbers                = { OldStyle }
      }
    }
    \tl_if_in:cnT {
      g_lngx_math_tl
    } { NewCM } {
      \bool_if:NT \g_lngx_old_style_one_bool {
        \lngx_set_keys:n {
          text~
          upright~ features      = {
            CharacterVariant       = { 6 }
          },
          text~ sans~
          upright~ features      = {
            CharacterVariant       = { 6 }
          }
        }
      }
    }
  }
  \tl_if_in:cnT {
    g _ lngx _ math _ tl
  } { NewCM } {
    \bool_if:NT \g_lngx_bourbaki_bool {
      \lngx_set_keys:n {
        math~ features         = {
          CharacterVariant       = { 1 }
        }
      }
    }
  }
```

If the New Computer Modern fonts are used, we don't need their `.fontspec` files as I already have incorporated all their settings in the package itself. So I have used the `IgnoreFontspecFile` option for fontspec.

```
  \tl_if_in:cnT {
    g _ lngx _ text _ upright _ tl
  } { NewCM } {
    \lngx_set_keys:n {
      text~
      upright~ features        = {
        IgnoreFontspecFile
      }
    }
  }
  \tl_if_in:cnT {
    g _ lngx _ text _ sans _ upright _ tl
  } { NewCM } {
    \lngx_set_keys:n {
      text~
      sans~ upright~ features  = {
        IgnoreFontspecFile
      }
    }
  }
  \tl_if_in:cnT {
    g _ lngx _ text _ mono _ upright _ tl
  } { NewCM } {
    \lngx_set_keys:n {
      text~
      mono~ upright~ features  = {
        IgnoreFontspecFile
      }
    }
  }
  \lngx_set_main_font:ee {
    \g_lngx_text_upright_features_tl
  } {
    \g_lngx_text_upright_tl
  }
  \lngx_set_sans_font:ee {
    \g_lngx_text_sans_upright_features_tl
  } {
    \g_lngx_text_sans_upright_tl
  }
  \lngx_set_mono_font:ee {
    \g_lngx_text_mono_upright_features_tl
  } {
    \g_lngx_text_mono_upright_tl
  }
  \lngx_set_math_font:ee {
    \g_lngx_math_features_tl
  } {
    \g_lngx_math_tl
  }
}
⟨/font⟩
```

```
⟨*ipa⟩
\ProvidesExplPackage{linguistix-ipa}
                    {2025-05-20}
                    {v0.1b}
                    {%
                      A package for typesetting the IPA
                      (International Phonetic Alphabet) from
                      the ‘LinguisTiX’ bundle.%
                    }
```

Then, I load unicode-math, LinguisTiX-nfss and LinguisTiX-base (if they are not already loaded).

```

\IfPackageLoadedF { unicode-math } {
  \RequirePackage { unicode-math }
}

\IfPackageLoadedF { linguistix-base } {
  \RequirePackage { linguistix-base }
}

\IfPackageLoadedF { linguistix-nfss } {
  \RequirePackage { linguistix-nfss }
}
```

\ipatext
\ipatext*

The \ipatext command along with its starred variant is developed here.

```

\NewDocumentCommand \ipatext { s m } {
  \IfBooleanTF { #1 } {
    {
      \lngxipa
      / #2 /
    }
  } {
    {
      \lngxipa
      [ #2 ]
    }
  }
}
```

(*End of definition for* `\ipatext` *and* `\ipatext*`. *These functions are documented on page 6.*)

ipa upright
ipa upright features
ipa bold upright
ipa bold upright features
ipa italic
ipa italic features
ipa bold italic
ipa bold italic features
ipa slanted
ipa slanted features
ipa bold slanted
ipa bold slanted features
ipa swash
ipa swash features
ipa bold swash
ipa bold swash features
ipa small caps
ipa small caps features
\g_lngx_ipa_upright_tl
\g_lngx_ipa_upright_features_tl
\g_lngx_ipa_bold_upright_tl
\g_lngx_ipa_bold_upright_features_tl
\g_lngx_ipa_italic_tl
\g_lngx_ipa_italic_features_tl
\g_lngx_ipa_bold_italic_tl
\g_lngx_ipa_bold_italic_features_tl
\g_lngx_ipa_slanted_tl
\g_lngx_ipa_slanted_features_tl
\g_lngx_ipa_bold_slanted_tl
\g_lngx_ipa_bold_slanted_features_tl
\g_lngx_ipa_swash_tl
\g_lngx_ipa_swash_features_tl
\g_lngx_ipa_bold_swash_tl
\g_lngx_ipa_bold_swash_features_tl
\g_lngx_ipa_small_caps_tl
\g_lngx_ipa_small_caps_features_tl

These variables store the values for fonts and features for the serif IPA.

```

\clist_map_inline:nn {
  upright,
  bold~ upright,
  italic,
  bold~ italic,
  slanted,
  bold~ slanted,
  swash,
  bold~ swash,
  small~ caps
} {
  \tl_set:Nn \l_tmpa_tl { #1 }
  \tl_replace_all:Nnn \l_tmpa_tl { ~ } { _ }
  \tl_gclear_new:c {
    g _ lngx _ ipa _ \l_tmpa_tl _ features _ tl
  }
  \keys_define:nn { lngx _ keys } {
    ipa~ #1
    .tl_gset_e:c           = {
      g _ lngx _ ipa _ \l_tmpa_tl _ tl
    },
    ipa~ #1~ features
    .code:n                = {
      \tl_set:Nn \l_tmpb_tl { #1 }
      \tl_replace_all:Nnn \l_tmpb_tl { ~ } { _ }
      \tl_put_right:ce {
        g _ lngx _ ipa _ \l_tmpb_tl _ features _ tl
      } { ##1 , }
      \tl_clear:N \l_tmpb_tl
    }
  }
  \tl_clear:N \l_tmpa_tl
}
```

(*End of definition for* `ipa upright` *and others. These functions are documented on page 7.*)

ipa sans upright
ipa sans upright features
ipa sans bold upright
ipa sans bold upright features
ipa sans italic
ipa sans italic features
ipa sans bold italic
ipa sans bold italic features
ipa sans slanted
ipa sans slanted features
ipa sans bold slanted
ipa sans bold slanted features
ipa sans swash
ipa sans swash features
ipa sans bold swash
ipa sans bold swash features
ipa sans small caps
ipa sans small caps features
\g_lngx_ipa_sans_upright_tl
\g_lngx_ipa_sans_upright_features_tl
\g_lngx_ipa_sans_bold_upright_tl
\g_lngx_ipa_sans_bold_upright_features_tl
\g_lngx_ipa_sans_italic_tl
\g_lngx_ipa_sans_italic_features_tl
\g_lngx_ipa_sans_bold_italic_tl
\g_lngx_ipa_sans_bold_italic_features_tl
\g_lngx_ipa_sans_slanted_tl
\g_lngx_ipa_sans_slanted_features_tl
\g_lngx_ipa_sans_bold_slanted_tl
\g_lngx_ipa_sans_bold_slanted_features_tl
\g_lngx_ipa_sans_swash_tl
\g_lngx_ipa_sans_swash_features_tl
\g_lngx_ipa_sans_bold_swash_tl
\g_lngx_ipa_sans_bold_swash_features_tl
\g_lngx_ipa_sans_small_caps_tl
\g_lngx_ipa_sans_small_caps_features_tl

These variables store the values for fonts and features for the sans IPA.

```

\clist_map_inline:nn {
  upright,
  bold~ upright,
  italic,
  bold~ italic,
  slanted,
  bold~ slanted,
  swash,
  bold~ swash,
  small~ caps
} {
  \tl_set:Nn \l_tmpa_tl { #1 }
  \tl_replace_all:Nnn \l_tmpa_tl { ~ } { _ }
  \tl_gclear_new:c {
    g _ lngx _ ipa _ mono _ \l_tmpa_tl _ features _ tl
  }
  \keys_define:nn { lngx _ keys } {
    ipa~ mono~ #1
    .tl_gset_e:c          = {
      g _ lngx _ ipa _ mono _ \l_tmpa_tl _ tl
    },
    ipa~ mono~ #1~ features
    .code:n               = {
      \tl_set:Nn \l_tmpb_tl { #1 }
      \tl_replace_all:Nnn \l_tmpb_tl { ~ } { _ }
      \tl_put_right:ce {
        g _ lngx _ ipa _ mono _ \l_tmpb_tl _ features _ tl
      } { ##1 , }
      \tl_clear:N \l_tmpb_tl
    }
  }
  \tl_clear:N \l_tmpa_tl
}
```

(*End of definition for* `ipa sans upright` *and others. These functions are documented on page 7.*)

ipa mono upright
ipa mono upright features
ipa mono bold upright
ipa mono bold upright features
ipa mono italic
ipa mono italic features
ipa mono bold italic
ipa mono bold italic features
ipa mono slanted
ipa mono slanted features
ipa mono bold slanted
ipa mono bold slanted features
ipa mono swash
ipa mono swash features
ipa mono bold swash
ipa mono bold swash features
ipa mono small caps
ipa mono small caps features
\g_lngx_ipa_mono_upright_tl
\g_lngx_ipa_mono_upright_features_tl
\g_lngx_ipa_mono_bold_upright_tl
\g_lngx_ipa_mono_bold_upright_features_tl
\g_lngx_ipa_mono_italic_tl
\g_lngx_ipa_mono_italic_features_tl
\g_lngx_ipa_mono_bold_italic_tl
\g_lngx_ipa_mono_bold_italic_features_tl
\g_lngx_ipa_mono_slanted_tl
\g_lngx_ipa_mono_slanted_features_tl
\g_lngx_ipa_mono_bold_slanted_tl
\g_lngx_ipa_mono_bold_slanted_features_tl
\g_lngx_ipa_mono_swash_tl
\g_lngx_ipa_mono_swash_features_tl
\g_lngx_ipa_mono_bold_swash_tl
\g_lngx_ipa_mono_bold_swash_features_tl
\g_lngx_ipa_mono_small_caps_tl
\g_lngx_ipa_mono_small_caps_features_tl

These variables store the values for fonts and features for the monospaced IPA.

```

\clist_map_inline:nn {
  upright,
  bold~ upright,
  italic,
  bold~ italic,
  slanted,
  bold~ slanted,
  swash,
  bold~ swash,
  small~ caps
} {
  \tl_set:Nn \l_tmpa_tl { #1 }
  \tl_replace_all:Nnn \l_tmpa_tl { ~ } { _ }
  \tl_gclear_new:c {
    g _ lngx _ ipa _ sans _ \l_tmpa_tl _ features _ tl
  }
  \keys_define:nn { lngx _ keys } {
    ipa~ sans~ #1
    .tl_gset_e:c           = {
      g _ lngx _ ipa _ sans _ \l_tmpa_tl _ tl
    },
    ipa~ sans~ #1~ features
    .code:n                = {
      \tl_set:Nn \l_tmpb_tl { #1 }
      \tl_replace_all:Nnn \l_tmpb_tl { ~ } { _ }
      \tl_put_right:ce {
        g _ lngx _ ipa _ sans _ \l_tmpb_tl _ features _ tl
      } { ##1 , }
      \tl_clear:N \l_tmpb_tl
    }
  }
  \tl_clear:N \l_tmpa_tl
}
```

(*End of definition for* ***ipa mono upright*** *and others. These functions are documented on page 8.*)

ipa newcm

This key sets New Computer Modern fonts in all weights, all families in the context of IPA.

```

\keys_define:nn { lngx _ keys } {
  ipa~ newcm
  .meta:n                = {
    ipa~
    upright                = {
      NewCM10-Book.otf
    },
    ipa~
    bold~ upright          = {
      NewCM10-Bold.otf
    },
    ipa~
    italic                 = {
      NewCM10-BookItalic.otf
    },
    ipa~
    bold~ italic           = {
      NewCM10-BoldItalic.otf
    },
    ipa~
    slanted                = {
      NewCM10-Book.otf
    },
    ipa~
    bold~ slanted          = {
      NewCM10-Bold.otf
    },
    ipa~
    swash                  = {
      NewCM10-Book.otf
    },
    ipa~
    bold~ swash            = {
      NewCM10-Bold.otf
    },
    ipa~
    small~ caps            = {
      NewCM10-Book.otf
    },
    ipa~
    sans~ upright          = {
      NewCMSans10-Book.otf
    },
    ipa~
    sans~ bold~ upright    = {
      NewCMSans10-Bold.otf
    },
    ipa~
    sans~ italic           = {
      NewCMSans10-BookOblique.otf
    },
    ipa~
    sans~ bold~ italic     = {
      NewCMSans10-BoldOblique.otf
    },
    ipa~
    sans~ slanted          = {
      NewCMSans10-BookOblique.otf
    },
    ipa~
    sans~ bold~ slanted    = {
      NewCMSans10-BoldOblique.otf
    },
    ipa~
    sans~ swash            = {
      NewCMSans10-Book.otf
    },
    ipa~
    sans~ bold~ swash      = {
      NewCMSans10-Bold.otf
    },
    ipa~
    sans~ small~ caps      = {
      NewCMSans10-Book.otf
    },
    ipa~
    mono~ upright          = {
      NewCMMono10-Book.otf
    },
    ipa~
    mono~ bold~ upright    = {
      NewCMMono10-Bold.otf
    },
    ipa~
    mono~ italic           = {
      NewCMMono10-BookItalic.otf
    },
    ipa~
    mono~ bold~ italic     = {
      NewCMMono10-BoldOblique.otf
    },
    ipa~
    mono~ slanted          = {
      NewCMMono10-Book.otf
    },
    ipa~
    mono~ bold~ slanted    = {
      NewCMMono10-BoldOblique.otf
    },
    ipa~
    mono~ swash            = {
      NewCMMono10-Book.otf
    },
    ipa~
    mono~ bold~ swash      = {
      NewCMMono10-Bold.otf
    },
    ipa~
    mono~ small~ caps      = {
      NewCMMono10-Book.otf
    }
  }
}
```

(*End of definition for* `ipa newcm`*. This function is documented on page 7.*)

`ipa newcm sans` This key sets New Computer Modern sans fonts in all weights, all families in the context of IPA.

```

\keys_define:nn { lngx _ keys } {
  ipa~ newcm~ sans
  .meta:n                 = {
    ipa~
    upright                 = {
      NewCMSans10-Book.otf
    },
    ipa~
    bold~ upright           = {
      NewCMSans10-Bold.otf
    },
    ipa~
    italic                  = {
      NewCMSans10-BookOblique.otf
    },
    ipa~
    bold~ italic            = {
      NewCMSans10-BoldOblique.otf
    },
    ipa~
    slanted                 = {
      NewCMSans10-BookOblique.otf
    },
    ipa~
    bold~ slanted           = {
      NewCMSans10-BoldOblique.otf
    },
    ipa~
    swash                   = {
      NewCMSans10-Book.otf
    },
    ipa~
    bold~ swash             = {
      NewCMSans10-Bold.otf
    },
    ipa~
    small~ caps             = {
      NewCMSans10-Book.otf
    }
  }
}
```

(*End of definition for* `ipa newcm sans`. *This function is documented on page 7.*)

`ipa newcm mono` This key sets New Computer Modern monospaced fonts in all weights, all families in the context of IPA.

```

\keys_define:nn { lngx _ keys } {
  ipa~ newcm~ mono
  .meta:n                 = {
    ipa~
    upright                 = {
      NewCMMono10-Book.otf
    },
    ipa~
    bold~ upright           = {
      NewCMMono10-Bold.otf
    },
    ipa~
    italic                  = {
      NewCMMono10-BookItalic.otf
    },
    ipa~
    bold~ italic            = {
      NewCMMono10-BoldOblique.otf
    },
    ipa~
    slanted                 = {
      NewCMMono10-Book.otf
    },
    ipa~
    bold~ slanted           = {
      NewCMMono10-BoldOblique.otf
    },
    ipa~
    swash                   = {
      NewCMMono10-Book.otf
    },
    ipa~
    bold~ swash             = {
      NewCMMono10-Bold.otf
    },
    ipa~
    small~ caps             = {
      NewCMMono10-Book.otf
    }
  }
}
```

(*End of definition for* `ipa newcm mono`. *This function is documented on page 7.*)

`ipa newcm regular` This key sets New Computer Modern regular serif fonts in all weights, all families in the context of IPA.

```

\keys_define:nn { lngx _ keys } {
  ipa~ newcm~ regular
  .meta:n                 = {
    ipa~
    upright                 = {
      NewCM10-Regular.otf
    },
    ipa~
    bold~ upright           = {
      NewCM10-Bold.otf
    },
    ipa~
    italic                  = {
      NewCM10-Italic.otf
    },
    ipa~
    bold~ italic           = {
      NewCM10-BoldItalic.otf
    },
    ipa~
    slanted                = {
      NewCM10-Regular.otf
    },
    ipa~
    bold~ slanted          = {
      NewCM10-Bold.otf
    },
    ipa~
    swash                  = {
      NewCM10-Regular.otf
    },
    ipa~
    bold~ swash            = {
      NewCM10-Bold.otf
    },
    ipa~
    small~ caps            = {
      NewCM10-Regular.otf
    },
    ipa~
    sans~ upright          = {
      NewCMSans10-Regular.otf
    },
    ipa~
    sans~ bold             = {
      NewCMSans10-Bold.otf
    },
    ipa~
    sans~ italic           = {
      NewCMSans10-Oblique.otf
    },
    ipa~
    sans~ bold~ italic     = {
      NewCMSans10-BoldOblique.otf
    },
    ipa~
    sans~ slanted          = {
      NewCMSans10-Regular.otf
    },
    ipa~
    sans~ bold~ slanted    = {
      NewCMSans10-Bold.otf
    },
    ipa~
    sans~ swash            = {
      NewCMSans10-Regular.otf
    },
    ipa~
    sans~ bold~ swash      = {
      NewCMSans10-Bold.otf
    },
    ipa~
    sans~ small~ caps      = {
      NewCMSans10-Regular.otf
    },
    ipa~
    mono~ upright          = {
      NewCMMono10-Regular.otf
    },
    ipa~
    mono~ bold             = {
      NewCMMono10-Bold.otf
    },
    ipa~
    mono~ italic           = {
      NewCMMono10-Italic.otf
    },
    ipa~
    mono~ bold~ italic     = {
      NewCMMono10-Bold.otf
    },
    ipa~
    mono~ slanted          = {
      NewCMMono10-Regular.otf
    },
    ipa~
    mono~ bold~ slanted    = {
      NewCMMono10-Bold.otf
    },
    ipa~
    mono~ swash            = {
      NewCMMono10-Regular.otf
    },
    ipa~
    mono~ bold~ swash      = {
      NewCMMono10-Bold.otf
    },
    ipa~
    mono~ small~ caps      = {
      NewCMMono10-Regular.otf
    }
  }
}
```

(*End of definition for* `ipa newcm regular`*. This function is documented on page 7.*)

`ipa newcm regular sans` This key sets New Computer Modern regular sans fonts in all weights, all families in the context of IPA.

```

\keys_define:nn { lngx _ keys } {
  ipa~ newcm~ sans~ regular
  .meta:n                = {
    ipa~
    upright                = {
      NewCMSans10-Regular.otf
    },
    ipa~
    bold                  = {
      NewCMSans10-Bold.otf
    },
    ipa~
    italic                = {
      NewCMSans10-Oblique.otf
    },
    ipa~
    bold~ italic          = {
      NewCMSans10-BoldOblique.otf
    },
    ipa~
    slanted               = {
      NewCMSans10-Regular.otf
    },
    ipa~
    bold~ slanted         = {
      NewCMSans10-Bold.otf
    },
    ipa~
    swash                 = {
      NewCMSans10-Regular.otf
    },
    ipa~
    bold~ swash           = {
      NewCMSans10-Bold.otf
    },
    ipa~
    small~ caps           = {
      NewCMSans10-Regular.otf
    }
  }
}
```

(*End of definition for* ***ipa newcm regular sans****. This function is documented on page 7.*)

ipa newcm regular mono This key sets New Computer Modern regular monospaced fonts in all weights, all families in the context of IPA.

```

\keys_define:nn { lngx _ keys } {
  ipa~ newcm~ mono~ regular
  .meta:n               = {
    ipa~
    upright               = {
      NewCMMono10-Regular.otf
    },
    ipa~
    bold                  = {
      NewCMMono10-Bold.otf
    },
    ipa~
    italic                 = {
      NewCMMono10-Italic.otf
    },
    ipa~
    bold~ italic           = {
      NewCMMono10-Bold.otf
    },
    ipa~
    slanted                = {
      NewCMMono10-Regular.otf
    },
    ipa~
    bold~ slanted          = {
      NewCMMono10-Bold.otf
    },
    ipa~
    swash                  = {
      NewCMMono10-Regular.otf
    },
    ipa~
    bold~ swash            = {
      NewCMMono10-Bold.otf
    },
    ipa~
    small~ caps            = {
      NewCMMono10-Regular.otf
    }
  }
}
```

(*End of definition for* `ipa newcm regular mono`*. This function is documented on page 7.*)

We set the `ipa newcm` key by default.

```

\lngx_set_keys:n {ipa~ newcm}
```

If LuaLaTeX is loaded, the `HarfBuzz` renderer is selected by default.

```

\sys_if_engine_luatex:T {
  \lngx_set_keys:n {
    ipa~
    upright~ features      = {
      Renderer               = { HarfBuzz }
    },
    ipa~ sans~
    upright~ features      = {
      Renderer               = { HarfBuzz }
    },
    ipa~ mono~
    upright~ features      = {
      Renderer               = { HarfBuzz }
    }
  }
}
```

\lngx_set_main_ipa_font:nn
\lngx_main_ipa:
lngx_ipa_rm_nfss
\lngx_set_sans_ipa_font:nn
\lngx_sans_ipa:
lngx_ipa_sf_nfss
\lngx_set_mono_ipa_font:nn
\lngx_mono_ipa:
lngx_ipa_tt_nfss

Here, I develop font-setting commands for IPA. These commands are set with `\setfontfamily`, so they keep overriding the definitions of the same command names. These commands set NFSS families that we use later for setting the IPA fonts. These functions and NFSS families are public, but manipulating them has effects (mostly desired) at several other places, so use them with caution.

```

\cs_new_protected:Npn \lngx_set_main_ipa_font:nn #1#2 {
  \setfontfamily \lngx_main_ipa: [
    #1,
    NFSSFamily              = { lngx _ ipa _ rm _nfss }
  ] { #2 }
}

\cs_new_protected:Npn \lngx_set_sans_ipa_font:nn #1#2 {
  \setfontfamily \lngx_sans_ipa: [
    #1,
    NFSSFamily              = { lngx _ ipa _ sf _nfss }
  ] { #2 }
}

\cs_new_protected:Npn \lngx_set_mono_ipa_font:nn #1#2 {
  \setfontfamily \lngx_mono_ipa: [
    #1,
    NFSSFamily              = { lngx _ ipa _ tt _nfss }
  ] { #2 }
}

\cs_generate_variant:Nn \lngx_set_main_ipa_font:nn { ee }
\cs_generate_variant:Nn \lngx_set_sans_ipa_font:nn { ee }
\cs_generate_variant:Nn \lngx_set_mono_ipa_font:nn { ee }
```

(*End of definition for* `\lngx_set_main_ipa_font:nn` *and others. These functions are documented on page 12.*)

lngx_ipa

Here, I create a 'super font family' with `\lngx_super_font_family:nn`, a macro provided by LinguisTiX-NFSS. Please see the documentation of that package for more information. Note that `lngx_ipa` is a super family responsible for all the IPA-related functions of the package. It is associated with the NFSS families defined just now for the IPA.

```

\lngx_super_font_family:nn { lngx _ ipa } {
  rm                        = { lngx _ ipa _ rm _ nfss },
  sf                        = { lngx _ ipa _ sf _ nfss },
  tt                        = { lngx _ ipa _ tt _ nfss }
}
```

(*End of definition for* `lngx_ipa`*. This function is documented on page 13.*)

\lngxipa
\lngx_ipa:

I use `\lngx_softer_super_font_family:n` provided by LinguisTiX-NFSSfor defining this switch to the IPA.

```

\cs_new_protected:Npn \lngx_ipa: {
  \lngx_softer_super_font_family:n { lngx _ ipa }
}

\cs_set_eq:NN \lngxipa \lngx_ipa:
```

(*End of definition for* `\lngxipa` *and* `\lngx_ipa:`. *These functions are documented on page 7.*)

Now, I have used the exact same method that I described in the implementation of LinguisTiX-fonts for setting the size variants. This is done with lazy evaluation, just before `\begin{document}`.

```

\hook_gput_code:nnn { begindocument / before } { . } {
  \tl_if_in:cnT {
    g _ lngx _ ipa _ upright _ tl
  } { NewCM } {
    \tl_if_in:cnTF {
      g _ lngx _ ipa _ upright _ tl
    } { Book } {
      \lngx_set_keys:n {
        ipa~
        upright~ features      = {
          SizeFeatures           = {
            {
              Size                     = {-8},
              Font                     = {
                NewCM08-Book.otf
              }
            },
            {
              Size                     = {8-},
              Font                     = {
                NewCM10-Book.otf
              }
            }
          }
        }
      }
    } {
      \lngx_set_keys:n {
        ipa~
        upright~ features      = {
          SizeFeatures           = {
            {
              Size                     = {-8},
              Font                     = {
                NewCM08-Regular.otf
              }
            },
            {
              Size                     = {8-},
              Font                     = {
                NewCM10-Regular.otf
              }
            }
          }
        }
      }
    }
  }
  \tl_if_in:cnT {
    g _ lngx _ ipa _ upright _ tl
  } { NewCMSans } {
    \tl_if_in:cnTF {
      g _ lngx _ ipa _ upright _ tl
    } { Book } {
      \lngx_set_keys:n {
        ipa~
        upright~ features      = {
          SizeFeatures           = {
            {
              Size                   = {-8},
              Font                   = {
                NewCMSans08-Book.otf
              }
            },
            {
              Size                   = {8-},
              Font                   = {
                NewCMSans10-Book.otf
              }
            }
          }
        }
      }
    } {
      \lngx_set_keys:n {
        ipa~
        upright~ features      = {
          SizeFeatures           = {
            {
              Size                   = {-8},
              Font                   = {
                NewCMSans08-Regular.otf
              }
            },
            {
              Size                   = {8-},
              Font                   = {
                NewCMSans10-Regular.otf
              }
            }
          }
        }
      }
    }
  }
  \tl_if_in:cnT {
    g _ lngx _ ipa _ italic _ tl
  } { NewCM } {
    \tl_if_in:cnTF {
      g _ lngx _ ipa _ italic _ tl
    } { Book } {
      \lngx_set_keys:n {
        ipa~
      italic~ features       = {
        SizeFeatures           = {
          {
            Size                   = {-8},
            Font                   = {
              NewCM08-BookItalic.otf
            }
          },
          {
            Size                   = {8-},
            Font                   = {
              NewCM10-BookItalic.otf
            }
          }
        }
      }
    }
  } {
    \lngx_set_keys:n {
      ipa~
      italic~ features       = {
        SizeFeatures           = {
          {
            Size                   = {-8},
            Font                   = {
              NewCM08-Italic.otf
            }
          },
          {
            Size                   = {8-},
            Font                   = {
              NewCM08-Italic.otf
            }
          }
        }
      }
    }
  }
}
\tl_if_in:cnT {
  g _ lngx _ ipa _ italic _ tl
} { NewCMSans } {
  \tl_if_in:cnTF {
    g _ lngx _ ipa _ italic _ tl
  } { Book } {
    \lngx_set_keys:n {
      ipa~
      italic~ features       = {
        SizeFeatures           = {
          {
            Size                   = {-8},
            Font                   = {
              NewCMSans08-BookOblique.otf
            }
          },
          {
            Size                   = {8-},
            Font                   = {
              NewCMSans10-BookOblique.otf
            }
          }
        }
      }
    }
  } {
    \lngx_set_keys:n {
      ipa~
      italic~ features       = {
        SizeFeatures           = {
          {
            Size                   = {-8},
            Font                   = {
              NewCMSans08-Oblique.otf
            }
          },
          {
            Size                   = {8-},
            Font                   = {
              NewCMSans08-Oblique.otf
            }
          }
        }
      }
    }
  }
}
\tl_if_in:cnT {
  g _ lngx _ ipa _ sans _ upright _ tl
} { NewCMSans } {
  \tl_if_in:cnTF {
    g _ lngx _ ipa _ upright _ tl
  } { Book } {
    \lngx_set_keys:n {
      ipa~ sans~
      upright~ features      = {
        SizeFeatures           = {
          {
            Size                   = {-8},
            Font                   = {
              NewCMSans08-Book.otf
            }
          },
          {
            Size                   = {8-},
            Font                   = {
              NewCMSans10-Book.otf
            }
          }
          }
        }
      }
    } {
      \lngx_set_keys:n {
        ipa~ sans~
        upright~ features      = {
          SizeFeatures           = {
            {
              Size                   = {-8},
              Font                   = {
                NewCMSans08-Regular.otf
              }
            },
            {
              Size                   = {8-},
              Font                   = {
                NewCMSans10-Regular.otf
              }
            }
          }
        }
      }
    }
  }
  \tl_if_in:cnT {
    g _ lngx _ ipa _ sans _ italic _ tl
  } { NewCMSans } {
    \tl_if_in:cnTF {
      g _ lngx _ ipa _ italic _ tl
    } { Book } {
      \lngx_set_keys:n {
        ipa~ sans~
        italic~ features       = {
          SizeFeatures           = {
            {
              Size                   = {-8},
              Font                   = {
                NewCMSans08-BookOblique.otf
              }
            },
            {
              Size                   = {8-},
              Font                   = {
                NewCMSans10-BookOblique.otf
              }
            }
          }
        }
      }
    } {
      \lngx_set_keys:n {
        ipa~ sans~
        italic~ features       = {
```

```
        SizeFeatures           = {
          {
            Size                   = {-8},
            Font                   = {
              NewCMSans08-Oblique.otf
            }
          },
          {
            Size                   = {8-},
            Font                   = {
              NewCMSans10-Oblique.otf
            }
          }
        }
      }
    }
  }
}
```

Now, I set the keys with the appropriate values and end the package.

```
\lngx_set_keys:n {
  ipa~ upright~ features   = {
    UprightFont            = {
      \g_lngx_ipa_upright_tl
    },
    UprightFeatures        = {
      \g_lngx_ipa_upright_features_tl
    },
    BoldFont                 = {
      \g_lngx_ipa_bold_upright_tl
    },
    BoldFeatures             = {
      \g_lngx_ipa_bold_upright_features_tl
    },
    ItalicFont               = {
      \g_lngx_ipa_italic_tl
    },
    ItalicFeatures           = {
      \g_lngx_ipa_italic_features_tl
    },
    BoldItalicFont           = {
      \g_lngx_ipa_bold_italic_tl
    },
    BoldItalicFeatures       = {
      \g_lngx_ipa_bold_italic_features_tl
    },
    \tl_if_empty:cF {
      g _ lngx _ ipa _ slanted _ tl
    } {
      SlantedFont              = {
        \g_lngx_ipa_slanted_tl
      },
      \tl_if_empty:cF {
        g _ lngx _ ipa _ slanted _ features _ tl
      } {
        SlantedFeatures          = {
          \g_lngx_ipa_slanted_features_tl
        },
      }
    }
    \tl_if_empty:cF {
      g _ lngx _ ipa _ bold _ slanted _ tl
    } {
      BoldSlantedFont          = {
        \g_lngx_ipa_bold_slanted_tl
      },
      \tl_if_empty:cF {
        g _ lngx _ ipa _ bold _ slanted _ features _ tl
      } {
        BoldSlantedFeatures      = {
          \g_lngx_ipa_bold_slanted_features_tl
        },
      }
    }
    \tl_if_empty:cF {
      g _ lngx _ ipa _ swash _ tl
    } {
      SwashFont                = {
        \g_lngx_ipa_swash_tl
      },
      \tl_if_empty:cF {
        g _ lngx _ ipa _ swash _ features _ tl
      } {
        SwashFeatures            = {
          \g_lngx_ipa_swash_features_tl
        },
      }
    }
    \tl_if_empty:cF {
      g _ lngx _ ipa _ bold _ swash _ tl
    } {
      BoldSwashFont            = {
        \g_lngx_ipa_bold_swash_tl
      },
      \tl_if_empty:cF {
        g _ lngx _ ipa _ bold _ swash _ features _ tl
      } {
        BoldSwashFeatures        = {
          \g_lngx_ipa_bold_swash_features_tl
        },
      }
    }
    \tl_if_empty:cF {
      g _ lngx _ ipa _ small _ caps _ tl
    } {
      SmallCapsFont            = {
        \g_lngx_ipa_small_caps_tl
      }
      \tl_if_empty:cF {
        g _ lngx _ ipa _ small _ caps _ features _ tl
      } {
        SmallCapsFeatures        = {
          \g_lngx_ipa_small_caps_features_tl
        }
      }
    }
  },
  ipa~
  sans~ upright~ features  = {
    UprightFont              = {
      \g_lngx_ipa_sans_upright_tl
    },
    UprightFeatures          = {
      \g_lngx_ipa_sans_upright_features_tl
    },
    BoldFont                 = {
      \g_lngx_ipa_sans_bold_upright_tl
    },
    BoldFeatures             = {
      \g_lngx_ipa_sans_bold_upright_features_tl
    },
    ItalicFont               = {
      \g_lngx_ipa_sans_italic_tl
    },
    ItalicFeatures           = {
      \g_lngx_ipa_sans_italic_features_tl
    },
    BoldItalicFont           = {
      \g_lngx_ipa_sans_bold_italic_tl
    },
    BoldItalicFeatures       = {
      \g_lngx_ipa_sans_bold_italic_features_tl
    },
    \tl_if_empty:cF {
      g _ lngx _ ipa _ slanted _ tl
    } {
      SlantedFont              = {
        \g_lngx_ipa_slanted_tl
      },
      \tl_if_empty:cF {
        g _ lngx _ ipa _ slanted _ features _ tl
      } {
        SlantedFeatures          = {
          \g_lngx_ipa_slanted_features_tl
        },
      }
    }
    \tl_if_empty:cF {
      g _ lngx _ ipa _ sans _ bold _ slanted _ tl
    } {
      BoldSlantedFont          = {
        \g_lngx_ipa_sans_bold_slanted_tl
      },
      \tl_if_empty:cF {
        g _ lngx _ ipa _ sans _ bold _ slanted _ features
        _ tl
      } {
        BoldSlantedFeatures      = {
          \g_lngx_ipa_sans_bold_slanted_features_tl
        },
      }
    }
    \tl_if_empty:cF {
      g _ lngx _ ipa _ sans_ swash _ tl
    } {
      SwashFont                = {
        \g_lngx_ipa_sans_swash_tl
      },
      \tl_if_empty:cF {
        g _ lngx _ ipa _ sans_ swash _ features _ tl
      } {
        SwashFeatures            = {
          \g_lngx_ipa_sans_swash_features_tl
        },
      }
    }
    \tl_if_empty:cF {
      g _ lngx _ ipa _ sans _ bold _ swash _ tl
    } {
      BoldSwashFont            = {
        \g_lngx_ipa_sans_bold_swash_tl
      },
      \tl_if_empty:cF {
        g _ lngx _ ipa _ sans _ bold _ swash _ features _
        tl
      } {
        BoldSwashFeatures        = {
          \g_lngx_ipa_sans_bold_swash_features_tl
        },
      }
    }
    \tl_if_empty:cF {
      g _ lngx _ ipa _ sans _ small _ caps _ tl
    } {
      SmallCapsFont            = {
        \g_lngx_ipa_sans_small_caps_tl
      }
      \tl_if_empty:cF {
        g _ lngx _ ipa _ sans _ small _ caps _ features _
        tl
      } {
        SmallCapsFeatures        = {
          \g_lngx_ipa_sans_small_caps_features_tl
        }
      }
    }
  },
  ipa~
  mono~ upright~ features  = {
    UprightFont              = {
      \g_lngx_ipa_mono_bold_upright_tl
    },
    UprightFeatures          = {
      \g_lngx_ipa_mono_bold_upright_features_tl
    },
    BoldFont                 = {
      \g_lngx_ipa_mono_bold_upright_tl
    },
    BoldFeatures             = {
      \g_lngx_ipa_mono_bold_upright_features_tl
    },
    ItalicFont               = {
      \g_lngx_ipa_mono_italic_tl
    },
    ItalicFeatures           = {
      \g_lngx_ipa_mono_italic_features_tl
    },
    BoldItalicFont           = {
      \g_lngx_ipa_mono_bold_italic_tl
    },
    BoldItalicFeatures       = {
      \g_lngx_ipa_mono_bold_italic_features_tl
    },
    \tl_if_empty:cF {
      g _ lngx _ ipa _ mono _ slanted _ tl
    } {
      SlantedFont              = {
        \g_lngx_ipa_mono_slanted_tl
      },
      \tl_if_empty:cF {
        g _ lngx _ ipa _ mono _ slanted _ features _ tl
      } {
        SlantedFeatures          = {
          \g_lngx_ipa_mono_slanted_features_tl
        },
      }
    }
    \tl_if_empty:cF {
      g _ lngx _ ipa _ mono _ bold _ slanted _ tl
    } {
      BoldSlantedFont          = {
        \g_lngx_ipa_mono_bold_slanted_tl
      },
      \tl_if_empty:cF {
        g _ lngx _ ipa _ mono _ bold _ slanted _ features
        _ tl
      } {
        BoldSlantedFeatures    = {
          \g_lngx_ipa_mono_bold_slanted_features_tl
        },
      }
    }
    \tl_if_empty:cF {
      g _ lngx _ ipa _ mono _ swash _ tl
    } {
      SwashFont                = {
        \g_lngx_ipa_mono_swash_tl
      },
      \tl_if_empty:cF {
        g _ lngx _ ipa _ mono _ swash _ features _ tl
      } {
        SwashFeatures            = {
          \g_lngx_ipa_mono_swash_features_tl
        },
      }
    }
    \tl_if_empty:cF {
      g _ lngx _ ipa _ mono _ bold _ swash _ tl
    } {
      BoldSwashFont            = {
        \g_lngx_ipa_mono_bold_swash_tl
      },
      \tl_if_empty:cF {
        g _ lngx _ ipa _ mono _ bold _ swash _ features _
        tl
      } {
        BoldSwashFeatures        = {
          \g_lngx_ipa_mono_bold_swash_features_tl
        },
      }
    }
    \tl_if_empty:cF {
      g _ lngx _ ipa _ mono _ small _ caps _ tl
    } {
      SmallCapsFont            = {
        \g_lngx_ipa_mono_small_caps_tl
      }
      \tl_if_empty:cF {
        g _ lngx _ ipa _ mono _ small _ caps _ features _
        tl
      } {
        SmallCapsFeatures        = {
          \g_lngx_ipa_mono_small_caps_features_tl
        }
      }
    }
  }
}
\tl_if_in:cnT {
  g _ lngx _ ipa _ upright _ tl
} { NewCM } {
  \lngx_set_keys:n {
    ipa~
    upright~ features        = {
      IgnoreFontspecFile,
      StylisticSet              = { 5 }
    }
  }
}
\tl_if_in:cnT {
  g _ lngx _ ipa _ sans _ upright _ tl
} { NewCM } {
  \lngx_set_keys:n {
    ipa~ sans~
    upright~ features        = {
      IgnoreFontspecFile,
      StylisticSet              = { 5 }
    }
  }
}
\tl_if_in:cnT {
  g _ lngx _ ipa _ mono _ upright _ tl
} { NewCM } {
  \lngx_set_keys:n {
    ipa~ mono~
    upright~ features        = {
      IgnoreFontspecFile,
      StylisticSet              = { 5 }
    }
  }
}
\lngx_set_main_ipa_font:ee {
  \g_lngx_ipa_upright_features_tl
} {
  \g_lngx_ipa_upright_tl
}
\lngx_set_sans_ipa_font:ee {
  \g_lngx_ipa_sans_upright_features_tl
} {
  \g_lngx_ipa_sans_upright_tl
}
\lngx_set_mono_ipa_font:ee {
  \g_lngx_ipa_mono_upright_features_tl
} {
  \g_lngx_ipa_mono_upright_tl
}
}
⟨/ipa⟩
```

```
⟨*logos⟩
\ProvidesExplPackage{linguistix-logos}
                    {2025-05-20}
                    {v0.1b}
                    {%
                      Logos of the ‘LinguisTiX’ bundle..%
                    }
```

The fontspec package (if not already loaded).

```

\IfPackageLoadedF { fontspec } {
  \RequirePackage { fontspec }
}
```

\lngx_logo_font: This is a command that switches to the New Computer Modern Uncial font family.

```

\newfontfamily \lngx_logo_font: [
  IgnoreFontspecFile,
  UprightFont             = { NewCMUncial10-Book.otf },
  UprightFeatures         = {
    SizeFeatures            = {
      {
        Size                    = {-8},
        Font                    = {NewCMUncial08-Book.otf}
      },
      {
        Size                    = {8-},
        Font                    = {NewCMUncial10-Book.otf}
      },
    }
  },
  BoldFont                = { NewCMUncial10-Bold.otf },
  BoldFeatures            = {
    SizeFeatures            = {
      {
        Size                    = {-8},
        Font                    = {NewCMUncial08-Bold.otf}
      },
      {
        Size                    = {8-},
        Font                    = {NewCMUncial10-Bold.otf}
      },
    }
  },
  Renderer                = { HarfBuzz }
]{ NewCMUncial10-Book.otf }
```

(*End of definition for* `\lngx_logo_font:`. *This function is documented on page 14.*)

lngx_purple_color

```

\color_set:nn { lngx _ purple _ color } { blue ! 50 ! red }
```

(*End of definition for* `lngx_purple_color`. *This function is documented on page 14.*)

\lngxlogo

```

\NewDocumentCommand \lngxlogo { O{} } {%
  \group_begin:
  \lngx_logo_font:
  LinguisTi
  \color_group_begin:
  \color_select:n { lngx_purple_color }
  X
  \color_group_end:
  \IfBlankF { #1 } { - #1 }
  \group_end:
}
```

(*End of definition for* `\lngxlogo`. *This function is documented on page* 8.)

\lngxpkg

```

\cs_new:Npn \lngxpkg {
  \IfPackageLoadedTF { hyperref } {
    \texorpdfstring {
      \lngxlogo
    } {
      LinguisTiX
    }
  } {
    \lngxlogo
  }
}
```

(*End of definition for* `\lngxpkg`. *This function is documented on page* 9.)

\lngxbaselogo
\lngxfontslogo
\lngxipalogo
\lngxlogoslogo
\lngxnfsslogo

```

\clist_map_inline:nn {
  base,
  examples,
  fonts,
  ipa,
  logos,
  nfss
} {
  \cs_new:cpn { lngx #1 logo } {
    \texorpdfstring {
      \lngxlogo [ #1 ]
    } {
      LinguisTiX - #1
    }
  }
}
⟨/logos⟩
```

(*End of definition for* `\lngxbaselogo` *and others. These functions are documented on page* 9.)

```
⟨*nfss⟩
\ProvidesExplPackage{linguistix-nfss}
                    {2025-05-20}
                    {v0.1b}
                    {%
                      An extension to the core NFSS commands
                      from the 'LinguisTiX' bundle.%
                    }
```

I need a few temporary `tl`s. I declare them here. As noted by the use of `__`, these are package-internal `tl`s. Even though I don't have any intention to change them, these are better not touched by the users.

```

\tl_new:N \l__lngx_normalfont_tmp_tl
\tl_new:N \l__lngx_selectfont_tmp_tl
\tl_new:N \l__lngx_family_tmp_tl
\tl_new:N \l__lngx_nfss_tmp_tl
```

These `tl`s are required for saving some values that are accessed later by the package as well as by the users.

```

\tl_new:N \l_lngx_current_encoding_tl
\tl_new:N \l_lngx_current_meta_family_tl
\tl_new:N \l_lngx_current_super_family_tl
\tl_new:N \l_lngx_current_series_tl
\tl_new:N \l_lngx_current_shape_tl
```

`\c_lngx_default_rmdefault_tl`
`\c_lngx_default_sfdefault_tl`
`\c_lngx_default_ttdefault_tl`

Here, I start the `begindocument/end` hook. After the document has started, a lot of initialisation can be assumed to have happened. I set some publicly available `tl`s here.

```

\hook_gput_code:nnn { begindocument / end } { . } {
  \tl_const:Ne \c_lngx_default_rmdefault_tl { \rmdefault }
  \tl_const:Ne \c_lngx_default_sfdefault_tl { \sfdefault }
  \tl_const:Ne \c_lngx_default_ttdefault_tl { \ttdefault }
```

(*End of definition for* `\c_lngx_default_rmdefault_tl`*,* `\c_lngx_default_sfdefault_tl`*, and* `\c_lngx_default_ttdefault_tl`*. These functions are documented on page 14.*)

`\l_lngx_current_encoding_tl`
`\l_lngx_current_meta_family_tl`
`\l_lngx_current_super_family_tl`
`\l_lngx_current_series_tl`
`\l_lngx_current_shape_tl`

First, I set the value `default` for the initial super font family.

```
  \tl_set:Nn \l_lngx_current_super_family_tl { default }
```

The current encoding is saved in the relevant `tl`.

```
  \tl_set:Ne \l_lngx_current_encoding_tl {
    \encodingdefault
  }
```

If the class is beamer, the font-family is automatically set to sans. Otherwise, mostly it is serif. Sadly, there is no public facing interface for confidently saying this, but as of now, this seems to be the picture. I check the current class and set the family `tl` accordingly.

```
  \IfClassLoadedTF { beamer } {
    \tl_set:Ne \l_lngx_current_meta_family_tl { sf }
  } {
    \tl_set:Ne \l_lngx_current_meta_family_tl { rm }
  }
```

Here, the series and shape `tl`s are set to their defaults.

```
  \tl_set:Ne \l_lngx_current_series_tl { md }
  \tl_set:Ne \l_lngx_current_shape_tl { up }
}
```

(*End of definition for* `\l_lngx_current_encoding_tl` *and others. These functions are documented on page 14.*)

The `\normalfont` command overrides the encoding. I trick the command by saving the encoding that was active before `\normalfont` in a temporary `tl`.

```

\hook_gput_code:nnn { cmd / normalfont / before } { . } {
  \tl_set:Ne \l__lngx_normalfont_tmp_tl { \f@encoding }
}
```

After the processing of `\normalfont`, I equate the temporary `tl` with the one that the package is tracking. This way, the effect of `\normalfont` remains unchanged, but we still save the values that were there before using it. Only encoding needs this special setting. Other attributes aren't reset by `\normalfont`.

```

\hook_gput_code:nnn { cmd / normalfont / after } { . } {
  \tl_set_eq:NN \l_lngx_current_encoding_tl
                \l__lngx_normalfont_tmp_tl
  \tl_clear:N   \l__lngx_normalfont_tmp_tl
}
```

Similar thing is done by `\selectfont` too. I repeat the code for that.

```

\hook_gput_code:nnn { cmd / selectfont / before } { . } {
  \tl_set:Ne \l__lngx_selectfont_tmp_tl { \f@encoding }
}

\hook_gput_code:nnn { cmd / selectfont / after } { . } {
  \tl_set_eq:NN \l_lngx_current_encoding_tl
                \l__lngx_selectfont_tmp_tl
  \tl_clear:N   \l__lngx_selectfont_tmp_tl
}
```

Now, after each `\XXfamily` commands, I save the family name in the respective `tl` for accessing later. All of these commands too reset the encoding. I repeat my trick for them too.

```

\hook_gput_code:nnn { cmd / rmfamily / before } { . } {
  \tl_set:Nn \l_lngx_current_meta_family_tl { rm }
  \tl_set:Ne \l__lngx_family_tmp_tl { \f@encoding }
}

\hook_gput_code:nnn { cmd / rmfamily / after } { . } {
  \tl_set:Nn \l_lngx_current_meta_family_tl { rm }
  \tl_set_eq:NN \l_lngx_current_encoding_tl
                \l__lngx_family_tmp_tl
  \tl_clear:N   \l__lngx_family_tmp_tl
}

\hook_gput_code:nnn { cmd / sffamily / before } { . } {
  \tl_set:Nn \l_lngx_current_meta_family_tl { sf }
  \tl_set:Ne \l__lngx_family_tmp_tl { \f@encoding }
}

\hook_gput_code:nnn { cmd / sffamily / after } { . } {
  \tl_set:Nn \l_lngx_current_meta_family_tl { sf }
  \tl_set_eq:NN \l_lngx_current_encoding_tl
                \l__lngx_family_tmp_tl
  \tl_clear:N   \l__lngx_family_tmp_tl
}

\hook_gput_code:nnn { cmd / ttfamily / before } { . } {
  \tl_set:Nn \l_lngx_current_meta_family_tl { tt }
  \tl_set:Ne \l__lngx_family_tmp_tl { \f@encoding }
}

\hook_gput_code:nnn { cmd / ttfamily / after } { . } {
  \tl_set:Nn \l_lngx_current_meta_family_tl { tt }
  \tl_set_eq:NN \l_lngx_current_encoding_tl
                \l__lngx_family_tmp_tl
  \tl_clear:N   \l__lngx_family_tmp_tl
}
```

After the series commands, I save the series name in the `tl`. Note that, I don't use the traditional LaTeX labels `m`, `bx` etc. Using, `md` and `bx` is more intuitive, plus they also can be used in the argument of `\use:c` directly.

```

\hook_gput_code:nnn { cmd / mdseries / after } { . } {
  \tl_set:Nn \l_lngx_current_series_tl { md }
}

\hook_gput_code:nnn { cmd / bfseries / after } { . } {
  \tl_set:Nn \l_lngx_current_series_tl { bf }
}
```

For shape related commands too, I save the names that are more closer to their respective commands.

```

\hook_gput_code:nnn { cmd / upshape / after } { . } {
  \tl_set:Nn \l_lngx_current_shape_tl { up }
}

\hook_gput_code:nnn { cmd / itshape / after } { . } {
  \tl_set:Nn \l_lngx_current_shape_tl { it }
}

\hook_gput_code:nnn { cmd / scshape / after } { . } {
  \tl_set:Nn \l_lngx_current_shape_tl { sc }
}

\hook_gput_code:nnn { cmd / sscshape / after } { . } {
  \tl_set:Nn \l_lngx_current_shape_tl { ssc }
}

\hook_gput_code:nnn { cmd / slshape / after } { . } {
  \tl_set:Nn \l_lngx_current_shape_tl { sl }
}

\hook_gput_code:nnn { cmd / swshape / after } { . } {
  \tl_set:Nn \l_lngx_current_shape_tl { sw }
}

\hook_gput_code:nnn { cmd / ulcshape / after } { . } {
  \tl_set:Nn \l_lngx_current_shape_tl { ulc }
}

\hook_gput_code:nnn { cmd / ulcshape / after } { . } {
  \tl_set:Nn \l_lngx_current_shape_tl { #1 }
}
```

\lngx_if_encoding_p:n
\lngx_if_encoding:n*TF*

I provide a conditional for checking the current encoding with the given argument.

```
\prg_new_conditional:Nnn \lngx_if_encoding:n {
  p,
  T,
  F,
  TF
} {
  \tl_if_eq:NnTF \l_lngx_current_encoding_tl { #1 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

(*End of definition for* \lngx_if_encoding:nTF*. This function is documented on page 15.*)

\IfEncodingTF
\IfEncodingT
\IfEncodingF

For non-LaTeX3 contexts, these simpler alternatives are provided.

```
\cs_new_eq:NN \IfEncodingTF \lngx_if_encoding:nTF
\cs_new_eq:NN \IfEncodingT  \lngx_if_encoding:nT
\cs_new_eq:NN \IfEncodingF  \lngx_if_encoding:nF
```

(*End of definition for* \IfEncodingTF*,* \IfEncodingT*, and* \IfEncodingF*. These functions are documented on page 10.*)

\lngx_if_meta_family_p:n
\lngx_if_meta_family:n*TF*

A conditional for checking the meta family with the given argument.

```
\prg_new_conditional:Nnn \lngx_if_meta_family:n {
  p,
  T,
  F,
  TF
} {
  \tl_if_eq:NnTF \l_lngx_current_meta_family_tl { #1 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

(*End of definition for* `\lngx_if_meta_family:nTF`. *This function is documented on page 15.*)

`\IfMetaFamilyTF`
`\IfMetaFamilyT`
`\IfMetaFamilyF`

User-facing conditionals for meta family.

```

\cs_new_eq:NN \IfMetaFamilyTF \lngx_if_meta_family:nTF
\cs_new_eq:NN \IfMetaFamilyT  \lngx_if_meta_family:nT
\cs_new_eq:NN \IfMetaFamilyF  \lngx_if_meta_family:nF
```

(*End of definition for* `\IfMetaFamilyTF`, `\IfMetaFamilyT`, *and* `\IfMetaFamilyF`. *These functions are documented on page 10.*)

`\lngx_if_super_family_p:n`
`\lngx_if_super_family:nTF`

A conditional for checking the super family with the given argument.

```

\prg_new_conditional:Nnn \lngx_if_super_family:n {
  p,
  T,
  F,
  TF
} {
  \tl_if_eq:NnTF \l_lngx_current_super_family_tl { #1 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

(*End of definition for* `\lngx_if_super_family:nTF`. *This function is documented on page 15.*)

`\IfSuperFamilyTF`
`\IfSuperFamilyT`
`\IfSuperFamilyF`

User-facing conditionals for super family.

```

\cs_new_eq:NN \IfSuperFamilyTF \lngx_if_super_family:nTF
\cs_new_eq:NN \IfSuperFamilyT  \lngx_if_super_family:nT
\cs_new_eq:NN \IfSuperFamilyF  \lngx_if_super_family:nF
```

(*End of definition for* `\IfSuperFamilyTF`, `\IfSuperFamilyT`, *and* `\IfSuperFamilyF`. *These functions are documented on page 10.*)

`\lngx_if_series_p:n`
`\lngx_if_series:nTF`

A conditional for checking the current series with the given argument.

```

\prg_new_conditional:Nnn \lngx_if_series:n {
  p,
  T,
  F,
  TF
} {
  \tl_if_eq:NnTF \l_lngx_current_series_tl { #1 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

(*End of definition for* `\lngx_if_series:nTF`. *This function is documented on page 15.*)

\IfSeriesTF
\IfSeriesT
\IfSeriesF

Its user-side macros.

```

\cs_new_eq:NN \IfSeriesTF \lngx_if_series:nTF
\cs_new_eq:NN \IfSeriesT  \lngx_if_series:nT
\cs_new_eq:NN \IfSeriesF  \lngx_if_series:nF
```

(*End of definition for* `\IfSeriesTF`*,* `\IfSeriesT`*, and* `\IfSeriesF`*. These functions are documented on page 11.*)

\lngx_if_shape_p:n
\lngx_if_shape:n*TF*

A conditional for checking the current shape with the current argument.

```

\prg_new_conditional:Nnn \lngx_if_shape:n {
  p,
  T,
  F,
  TF
} {
  \tl_if_eq:NnTF \l_lngx_current_shape_tl { #1 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

(*End of definition for* `\lngx_if_shape:nTF`*. This function is documented on page 15.*)

\IfShapeTF
\IfShapeT
\IfShapeF

User-side macros for the same.

```

\cs_new_eq:NN \IfShapeTF \lngx_if_shape:nTF
\cs_new_eq:NN \IfShapeT  \lngx_if_shape:nT
\cs_new_eq:NN \IfShapeF  \lngx_if_shape:nF
```

(*End of definition for* `\IfShapeTF`*,* `\IfShapeT`*, and* `\IfShapeF`*. These functions are documented on page 11.*)

Now I will use the `\clist_map_inline:nn` technique for generating multiple conditionals of the same pattern. For that, I need a `cnn` variant of `\prg_new_conditional:Nnn` that I create with the following.

```

\cs_generate_variant:Nn \prg_new_conditional:Nnn { cnn }
```

\lngx_if_meta_family_rm_p:
\lngx_if_meta_family_rm:*TF*
\lngx_if_meta_family_sf_p:
\lngx_if_meta_family_sf:*TF*
\lngx_if_meta_family_tt_p:
\lngx_if_meta_family_tt:*TF*

These are separate conditionals for `rm`, `sf` and `tt` families. They don't require arguments. No user side commands are provided for these.

```

\clist_map_inline:nn {
  rm,
  sf,
  tt
} {
  \prg_new_conditional:cnn {
    lngx _ if _ meta _ family _ #1 :
  } {
    p, T, F, TF
  } {
    \tl_if_eq:NnTF \l_lngx_current_meta_family_tl { #1 } {
      \prg_return_true:
    } {
      \prg_return_false:
    }
  }
}
```

(*End of definition for* `\lngx_if_meta_family_rm:TF`*,* `\lngx_if_meta_family_sf:TF`*, and* `\lngx_if_meta_family_tt:TF`*. These functions are documented on page 15.*)

`\lngx_if_series_md_p:`
`\lngx_if_series_md:TF`
`\lngx_if_series_bf_p:`
`\lngx_if_series_bf:TF`

Separate conditionals for both the series.

```

\clist_map_inline:nn {
  md,
  bf
} {
  \prg_new_conditional:cnn { lngx _ if _ series _ #1 : } {
    p, T, F, TF
  } {
    \tl_if_eq:NnTF \l_lngx_current_series_tl { #1 } {
      \prg_return_true:
    } {
      \prg_return_false:
    }
  }
}
```

(*End of definition for* `\lngx_if_series_md:TF` *and* `\lngx_if_series_bf:TF`*. These functions are documented on page 15.*)

`\lngx_if_shape_up_p:`
`\lngx_if_shape_up:TF`
`\lngx_if_shape_it_p:`
`\lngx_if_shape_it:TF`
`\lngx_if_shape_sc_p:`
`\lngx_if_shape_sc:TF`
`\lngx_if_shape_ssc_p:`
`\lngx_if_shape_ssc:TF`
`\lngx_if_shape_sl_p:`
`\lngx_if_shape_sl:TF`
`\lngx_if_shape_sw_p:`
`\lngx_if_shape_sw:TF`
`\lngx_if_shape_ulc_p:`
`\lngx_if_shape_ulc:TF`

Separate conditionals for all the shapes.

```

\clist_map_inline:nn {
  up,
  it,
  sc,
  ssc,
  sl,
  sw,
  ulc
} {
  \prg_new_conditional:cnn { lngx _ if _ shape _ #1 : } {
    p, T, F, TF
  } {
    \tl_if_eq:NnTF \l_lngx_current_shape_tl { #1 } {
      \prg_return_true:
    } {
      \prg_return_false:
    }
  }
}
```

(*End of definition for* `\lngx_if_shape_up:TF` *and others. These functions are documented on page 15.*)

These keys are used in the argument of `\lngx_super_font_family:nn`. This is why they are separated from the set `lngx_keys`. We create new `tl`s using these keys that save the `rm`, `sf` and `tt` defaults of the new super font family. `\l__lngx_nfss_tmp_tl` is defined by the command that creates the super font family.

```

\clist_map_inline:nn {
  rm,
  sf,
  tt
} {
  \keys_define:nn { lngx _ nfss } {
    #1
    .code:n                = {
      \tl_gclear_new:c {
        g _ lngx _ \l__lngx_nfss_tmp_tl _ #1 default _ tl
      }
      \tl_gset:cn {
        g _ lngx _ \l__lngx_nfss_tmp_tl _ #1 default _ tl
      } { ##1 }
    }
  }
}
```

`\lngx_super_font_family:nn`
`\superfontfamily`

I first set the temporary `tl` with the name of the super font family retrieved from the first argument.

```

\cs_new_protected:Npn \lngx_super_font_family:nn #1#2 {
  \tl_set:Ne \l__lngx_nfss_tmp_tl { #1 }
```

Now, I pass the second argument to the key-set I just defined. The temporary `tl` is cleared. This function comes with a user-side macro.

```
  \keys_set:nn { lngx _ nfss } { #2 }
  \tl_clear:N \l__lngx_nfss_tmp_tl
}

\cs_set_eq:NN \superfontfamily
              \lngx_super_font_family:nn
```

(*End of definition for* `\lngx_super_font_family:nn` *and* `\superfontfamily`*. These functions are documented on page 15.*)

`\lngx_soft_super_font_family:nn`
`\softsuperfontfamily`

I set the `tl` that saves the current font family to the first argument.

```

\cs_new_protected:Npn \lngx_soft_super_font_family:nn #1#2 {
  \tl_set:Ne \l_lngx_current_super_family_tl { #1 }
```

I first check if the `tl`s for rm, sf and tt are empty or not. Only if they are not, I use their content in the respective `\XXdefault`. This makes the use of all the keys optional. Only the keys that the user has used are processed here.

```
  \clist_map_inline:nn {
    rm,
    sf,
    tt
  } {
    \tl_if_empty:cF { g _ lngx _ #1 _ ##1 default _ tl } {
      \cs_set:cpe { ##1 default } {
        \tl_use:c { g _ lngx _ #1 _ ##1 default _ tl }
      }
    }
  }
```

After setting the `\XXdefault`, I use the `\normalfont` to initialise the super font family.

```
  \normalfont
```

Now all the aspects are reset. But, we have them saved in our `tl`s. So now depending on the attributes that the user wants to retrieve, I call those attributes again. The second argument is (expected to be) a comma-separated list of all such attributes. Thus, we change the super font family, but retain the already active attributes. This command has a user-facing macro.

```
  \clist_map_inline:nn { #2 } {
    \str_case:nn { ##1 } {
      { encoding } {
        \exp_args:NV \fontencoding
                     \l_lngx_current_encoding_tl
      }
      { family } {
        \use:c {
          \l_lngx_current_meta_family_tl family
        }
        \exp_args:NV \fontencoding
                     \l_lngx_current_encoding_tl
        \selectfont
      }
      { series } {
        \use:c {
          \l_lngx_current_series_tl series
        }
      }
      { shape }  {
        \use:c {
          \l_lngx_current_shape_tl shape
        }
      }
    }
  }
}

\cs_set_eq:NN \softsuperfontfamily
              \lngx_soft_super_font_family:nn
```

(*End of definition for* `\lngx_soft_super_font_family:nn` *and* `\softsuperfontfamily`*. These functions are documented on page 16.*)

`\lngx_softer_super_font_family:n`
`\softersuperfontfamily`

This function excludes the encoding and resets all the other attributes. It comes with a user-side macro.

```

\cs_new_protected:Npn \lngx_softer_super_font_family:n #1 {
  \lngx_soft_super_font_family:nn { #1 } {
    family,
    series,
    shape
  }
}

\cs_set_eq:NN \softersuperfontfamily
              \lngx_softer_super_font_family:n
```

(*End of definition for* `\lngx_softer_super_font_family:n` *and* `\softersuperfontfamily`*. These functions are documented on page 16.*)

`\lngx_softest_super_font_family:n`
`\softestsuperfontfamily`

This function resets all the attributes. It is available as a user-side macro.

```

\cs_new_protected:Npn \lngx_softest_super_font_family:n #1 {
  \lngx_soft_super_font_family:nn { #1 } {
    encoding,
    family,
    series,
    shape
  }
}

\cs_set_eq:NN \softestsuperfontfamily
              \lngx_softest_super_font_family:n
```

(*End of definition for* `\lngx_softest_super_font_family:n` *and* `\softestsuperfontfamily`*. These functions are documented on page 16.*)

`\lngx_soft_normal_font:n`
`\softnormalfont`

Following the same logic, I now provide the command for resetting to the default super family, but retaining the active attributes. I provide a user-side macro for this.

```

\cs_new_protected:Npn \lngx_soft_normal_font:n #1 {
  \tl_set:Ne \l_lngx_current_super_family_tl { default }
  \clist_map_inline:nn {
    rm,
    sf,
    tt
  } {
    \cs_set:cpe { ##1 default } {
      \tl_use:c { c _ lngx _ default _ ##1 default _ tl }
    }
  }
  \normalfont
  \clist_map_inline:nn { #1 } {
    \str_case:nn { ##1 } {
      { encoding } {
        \exp_args:NV \fontencoding
                     \l_lngx_current_encoding_tl
      }
      { family } {
        \use:c {
          \l_lngx_current_meta_family_tl family
        }
        \exp_args:NV \fontencoding
                     \l_lngx_current_encoding_tl
        \selectfont
      }
      { series } {
        \use:c {
          \l_lngx_current_series_tl series
        }
      }
      { shape }  {
        \use:c {
          \l_lngx_current_shape_tl shape
        }
      }
    }
  }
}

\cs_set_eq:NN \softnormalfont \lngx_soft_normal_font:n
```

(*End of definition for* `\lngx_soft_normal_font:n` *and* `\softnormalfont`. *These functions are documented on page 16.*)

`\lngx_softer_normal_font:`
`\softernormalfont`

This is a parallel to the 'softer' super family command for the default super family.

```

\cs_new_protected:Npn \lngx_softer_normal_font: {
  \lngx_soft_normal_font:n {
    family,
    series,
    shape
  }
}

\cs_set_eq:NN \softernormalfont \lngx_softer_normal_font:
```

(*End of definition for* `\lngx_softer_normal_font:` *and* `\softernormalfont`. *These functions are documented on page 16.*)

`\lngx_softest_normal_font:`
`\softestnormalfont`

This is a parallel to the 'softest' super family command for the default super family.

```

\cs_new_protected:Npn \lngx_softest_normal_font: {
  \lngx_soft_normal_font:n {
    encoding,
    family,
    series,
    shape
  }
}

\cs_set_eq:NN \softestnormalfont \lngx_softest_normal_font:
```

(*End of definition for* `\lngx_softest_normal_font:` *and* `\softestnormalfont`. *These functions are documented on page 16.*)

`\CurrentEncoding`
`\CurrentMetaFamily`
`\CurrentSeries`
`\CurrentShape`

Lastly, we create the commands that print the current values of the font attributes and end the package.

```
\cs_new:Npn \CurrentEncoding {
  \tl_use:N \l_lngx_current_encoding_tl
}
```

```
\cs_new:Npn \CurrentMetaFamily {
  \tl_use:N \l_lngx_current_meta_family_tl
}
\cs_new:Npn \CurrentSuperFamily {
  \tl_use:N \l_lngx_current_super_family_tl
}
\cs_new:Npn \CurrentSeries {
  \tl_use:N \l_lngx_current_series_tl
}
\cs_new:Npn \CurrentShape {
  \tl_use:N \l_lngx_current_shape_tl
}
⟨/nfss⟩
```

(*End of definition for* `\CurrentEncoding` *and others. These functions are documented on page 10.*)

References

Bringhurst, R. (2004). *The elements of typographic style* (4th ed.). Point Roberts, WA: Hartley & Marks, Publishers.

Gnu Free Documentation License

Version 1.3, 3 November 2008
Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

https://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document 'free' in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of 'copyleft', which means that derivative works of the document must themselves be free in the same sense. It complements the Gnu General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. Applicability and Definitions

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The '**Document**', below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as '**you**'. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A '**Modified Version**' of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A '**Secondary Section**' is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The '**Invariant Sections**' are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The '**Cover Texts**' are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A '**Transparent**' copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not 'Transparent' is called '**Opaque**'.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The '**Title Page**' means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, 'Title Page' means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The '**publisher**' means any person or entity that distributes copies of the Document to the public.

A section '**Entitled XYZ** ' means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as '**Acknowledgements**', '**Dedications**', '**Endorsements**', or '**History**'.) To '**Preserve the Title**' of such a section when you modify the Document means that it remains a section '**Entitled XYZ** ' according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. Copying in Quantity

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled 'History' in the various original documents, forming one section Entitled 'History'; likewise combine any sections Entitled 'Acknowledgements', and any sections Entitled 'Dedications'. You must delete all sections Entitled 'Endorsements'.

6. Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. Aggregation with Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an 'aggregate' if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the

8. Translation

9. Termination

10. Future Revisions of this License