The LinguisTiX bundle

निरंजन

19 January 2026 (v0.7)

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

1 **Introduction** 3
2 **Planned** 4
3 **Funding** 4
4 **Acknowledgements** 4
5 **LinguisTiX-base** 5
Interface... 17; Implementation... 23
6 **LinguisTiX-fixpex** 5
Interface... 18; Implementation... 24
7 **LinguisTiX-fonts** 5
Interface... 18; Implementation... 26
8 **LinguisTiX-eLossine** 7
Interface... 18; Implementation... 36
9 **LinguisTiX-ipa** 9
Interface... 19; Implementation... 51
10 **LinguisTiX-Languages** 12
Interface... 19; Implementation... 61
11 **LinguisTiX-Logos** 14
Interface... 20; Implementation... 68
12 **LinguisTiX-nfss** 14
Interface... 20; Implementation... 69

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. If you are an impatient reader 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 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*ɑ*]'. 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 9.

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 [g*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.

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.

As of 2025-05-29, I have recieved funding from the TeX users group's TeX development fund. They have decided to support the development of 'linguistix-glossing' (the logo will be available once the package is ready).

An experimental version of LinguisTiX-eLossinG is released on 2026-01-19. This version is for testing and getting feedback from the community. This marks the completion of the first grant provided by the TeX users group's. The project will still continue to develop further, so funding initiatives will be highly appreciated.

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!

Hardly in a week after the initial release, the TeX users group decided to financially support the development of a planned package in the bundle. I am grateful to them for their support.

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-fixpex

LaTeX3-interface | Implementation

This package offers a fix for the clash between expex and unicode-math. It provides a single command.

`\umgla` This is a replica of the unicode-math-`\gla`. Since the expex-`\gla` is more relevant in linguistics, I set it as the default. If one needs to use unicode-math-`\gla`, they can use this command.

7 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.

Antonis suggested a typographic enhancement for the logo of LaTeX. The default logo scales the 'A' and that affects the 'colour' of the font. This is why I renew the logo with the code given by Antonis. The original logo is also available with an alternative command.

`\LaTeX` LaTeX
`\ogLaTeX` LaTeX

The package provides only these commands. Let's now have a look at the keys provided for the text.

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`

`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 defaults.

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.

[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.

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

In (LA)TEX, the default shape of the 'empty set' symbol is: '∅', 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 by default and the slashed zero is provided as a character variant. Since the Unicode-correct `$\emptyset$` is activated by the package, it always renders: '∅' and not: '∅'. 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! ;-)*

8 LinguisTiX-Glossing

LaTeX3-interface | Implementation

This package provides a suit for creating interlinear glosses. It is supported by TEX users group's devfund. The package attempts to be an all-in-one solution for glossing. It doesn't provide any particular glosses. It only provides a method to create them. Using it, one may easily create packages like LinguisTiX-Leipzig to support a set of glosses. The glosses created by the package use the new code of the LATEX project as they are created in a tagging aware manner. Each gloss sets a hyperlink to its position in the list of glosses. Let's take a look at its commands and options.

\glx {⟨*comma separated list of glosses*⟩}
\glx* {⟨*comma separated list of glosses*⟩}

This simple commands take a comma separated as their argument. All the items from the list are glosses (either created by the user or provided by a package). Cases are ignored. Spaces around the glosses are ignored. The regular unstarred command prints the glosses of each of the item in the comma separated list, whereas the starred variant prints their expansions. Have a look at the following example.

```
\DocumentMetadata{tagging=on,lang={en-GB}}
\documentclass{article}
\usepackage{linguistix-glossing}
\newgloss{prs}{present tense}
\newgloss{pst}{past tense}

\begin{document}
\glx{prs,pst}\par
\glx{ prs, pst   }\par
\glx{ Prs,pSt}

\glx*{prs,pst}\par
\glx*{ prs, pst   }\par
\glx*{ Prs,pST}
\end{document}
```

This example produces identical output in three lines for glosses and the same for its expansions. Notice that there is no format to the cases of the glosses and similarly one level of spaces are trimmed.

`\newgloss` {⟨*gloss*⟩} {⟨*expansion*⟩}
`\renewgloss` {⟨*gloss*⟩} {⟨*expansion*⟩}

These commands create a new gloss or renew an existing gloss. They can be accessed with the **\glx** command as explained above. Using **\renewgloss** mid-document is not recommended. It will erase the data of page numbers for the previous version of it.

`\setupglossing` {⟨*keys for formatting glosses*⟩}

This command takes one argument, i.e., the keys that control everything regarding the use of glosses and their expansions. The keys it takes are described in the section that follows.

`\listofglosses` [⟨*setup keys*⟩]

This command prints the list of glosses using the default settings. If the optional argument is used, the adjustments are made locally only for a single run.

1 Setting up the glosses

The following keys can be passed to the command **\setupglossing**. They control the printing along with a lot of other things regarding glosses. All the customisation offered by the package can be accessed via this command.

`format` = {⟨*formatted element gloss/expansion*⟩}
`gloss` = {⟨*formatting commands for glosses*⟩} `\textsc{#1}`
`expansion` = {⟨*formatting commands for glosses*⟩}

The **format** key is used for setting the format of either gloss or expansion. It's a meta key that takes other key-val pairs in the argument. The nested keys control the formatting of the respective elements. No special formatting is applied to expansions, but glosses are by default printed in **\textsc**. These are the defaults of **gloss** and **expansion**.

`link color` = {⟨*link color*⟩} `black`

This option locally sets the colour for the hyperlinks. By default they are set to the black colour.

`sort` = {⟨*sorting style*⟩} `alphabetical` | `use`

This key controls how the keys printed in the list of glosses are ordered. They may be ordered alphabetically or following the sequence in which they were used, the former being the default.

`expansion case` = {⟨*case*⟩} `lowercase` | `title case all` | `title case first`

The expansion can be printed in one of these three cases. The default printing happens in lowercase.

`style` = {⟨*glossary style*⟩} `block` | `inline`

The package offers two styles. The **inline** style prints the glosses and their expansions without page numbers in the flowing text, whereas the **block** style, in default settings prints them in a multicolumn block with an unnumbered section with the glossary name.

`columns` = {⟨*number of columns*⟩} 2

The block style of glosses is printed in multicolumn layout by default. If the number of columns has to be adjusted, this key shall be used. The default value of it is 2. It works with only one column too.

`page numbers` = {⟨*truth value*⟩} `true` | `false`

By default, page numbers on which a particular gloss was used are printed in the `block` style. This can be turned off with this bool key.

`sectioning` = {⟨*section level*⟩} `section`

In `block` style, a section heading is printed. In order to choose the level of sectioning, this command can be used. The default is `section` which can be changed to any other desired level. In addition the key allows an option `null` which suppresses the use of any section heading.

`section number` = {⟨*truth value*⟩} `true` | `false`

By default, the section number for the glossary is turned off, but if one wants to print it, this bool key can be used with the `true` value.

`no bold` = {⟨*truth value*⟩} `true` | `false`

Generally, the glosses are printed in bold inside glossary. Some fonts don't have bold small caps (e.g., Latin Modern). If you need to stick to them, you can use this inverse bool key with true value in order to obtain non-bold glosses.

`separator` = {⟨*separator between glosses or expansions*⟩}

This is a meta key. If used with `\glx`, then it sets the separator between the glosses (`,␣` is the default). If used with `\glx*`, it sets the separator between the expansions (`,␣` is the default) and if used with the `\listofglosses`, it sets the separator between glosses and their expansions (`:␣` is the default).

`entry separator` = {⟨*separator between pairs of glosses and expansions*⟩}

Each pair of gloss and its expansion is separated using a token list controlled by this list. The default is `\par`.

9 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 can 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 [aɪ pʰiː eɪ]}
{\lngxipa /aɪ pʰiː eɪ/}
```

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

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 main font`	`ipa main font`
	`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`

Continued on the next page…

Family	LinguisTiX-fonts	LinguisTiX-ipa
	`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 font`	`ipa sans font`
	`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`
	`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 font`	`ipa mono font`
	`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

Apart from these, both the packages provide the following keys for appending to the

extra features for the respective fonts:

- `text extra features`
- `text sans extra features`
- `text mono extra features`
- `ipa extra features`
- `ipa sans extra features`
- `ipa mono extra features`

10 LinguisTiX-Languages

LaTeX3-interface | Implementation

This package is intended to provide support for loading Unicode fonts as well as other necessary settings for using languages. It is a wrapper around the babel package, but it provides some other useful settings which babel doesn't agree to add. This package is a little opinionated and pushes for 'modern' practices e.g., Unicode, LuaLaTeX, no-markup multilingual text etc. As of now, only a little support is available. If you want your language to be supported, you can ask for support at the bug tracker of the repository or you can send an email in the public mailing list for the project. You may subscribe to the mailing list at: mail.gnu.org.ua/mailman/listinfo/linguistix-languages. Here, I list down some LaTeX-aspects that may demand some modifications in the default settings.

Fonts: The package works with Unicode and does not worry about legacy methods. If you want support for your language, first and foremost, you should let me know standard OpenType fonts suitable for your language. Note that they should be freely licensed. I won't support proprietary software with LinguisTiX.

babel support: As mentioned before, the package adds on to the support provided by package babel. So check if the language files – specifically the modern `.ini` files – have the correct settings. Sometimes they may need to undergo native-speakers scrutiny. Whatever is wrong in babel, may not get corrected in LinguisTiX.

Numbers: LaTeX uses a lot of counters and all of them, by default, print Latin numerals/characters. E.g., `\arabic{page}` prints the page number in Latin, but `\roman{page}` prints the same in Roman convention, i.e., 'i, ii, …'. Does your language allow them? E.g., Greek doesn't like Latin alphabets, but doesn't mind Roman numerals. Instead of Latin alphabets, Greek prefers to use its own numeral system. Marathi doesn't like any of these, but it doesn't have alternative forms of numeration, so it changes certain cases drastically. E.g., in nested `enumerate` environment, Marathi renews the printing of nested `\item`s as ɪ, ɪ.ɪ, ɪ.ɪ.ɪ and ɪ.ɪ.ɪ.ɪ. This is reset to defaults when the language is changed. Keeping this in mind, I am listing down some places where I found non-native numbering (I might have missed something in which case it deserves to be reported as a bug, so feel free to do so!).

1. Page numbers (in front matter, main matter).
2. Part numbers.
3. Second, third and fourth levels of enumeration.

ExPex: Labels provided by ExPex package (see: tex.stackexchange.com/a/548668).

Typography: Language-specific conventions like using Italic for emphasis. It is a Latin-script specific convention (note that I don't mean slanted when I say Italic). Different languages have different conventions of emphasising (e.g., Marathi uses bold font for emphasis).

Miscellaneous: Anything other than these.

I am very much willing to support multilingual typesetting for multiple languages, but I need to know the things mentioned in this list in order to provide the best suited output. Please consider submitting a detailed feature request. The documentation of supported languages is in separate PDFs. This documentation only describes the user-side commands provided by the package.

`languages` `{⟨list of languages⟩}`
`\loadlanguages` `{⟨list of languages⟩}`

This key works with the central key-parser of LinguisTiX, i.e., `\linguistix`. It accepts one argument that is a list of languages user wants to load. Unlike babel, the first element of this list is set as the main language for the document. The command `\loadlanguages` has the identical behaviour. In fact, it is a wrapper around the key.

`\providelanguage` `{⟨language options⟩}` `{⟨language name⟩}`

This is a wrapper command over `\babelprovide`. The first argument is passed to the optional argument of `\babelprovide` and the second one to the mandatory argument of the same. For more information, please read babel's manual.

Languages supported by LinguisTiX-Languages are loaded with a package with that language's name. If it is absent, the package produces a warning.

`native numbering` `= {⟨strict/logical/off⟩}`

Many languages need native digits. Adding them in a multilingual document is quite complicated. This key sets the plugs provided for the socket of the same name. Language packages already take care of them, but if you want to change anything mid-document, you can use this key. It has three choices available as its value as seen below.

`strict` The 'strict' plug changes the `\lngx_counter:n` command to the counter of the main language of the document. That way all the counters are printed in the main language.

`logical` This plug changes the meaning of `\lngx_counter:n` to the `\localecounter` command provided by babel. It picks up the surrounding language and uses its native digits. E.g., when Marathi is being typeset, it will print counters in Marathi. When it is changed to English, it will start printing the same in English. Note that this will reflect in table of contents/tables/figures too. It is called logical numbering because it obeys TEX's logic more than what is generally considered the standard. E.g., imagine you have an English section followed by a Marathi section on the same page. Both of them will follow their own numerals for default TEX counters. Since both of them are on the same page, while shipping out, the last active language will be used for processing the page number (Marathi in this case). This creates a table of contents with Latin numeral as the section counter, but Marathi numeral as the page number. Only experiments can determine if an option like this can have valid use-cases, so it is provided. If you use it, be aware that the results might not be the most pleasant to your aesthetic values. They are so because of the logic of TEX.

`off` It is equivalent of the `noop` plug when the other two are not used at all. It is only required when you want to go back to LATEX defaults. E.g., if you have turned strict native numbering in some language and you want it to go back to LATEX defaults, you may use this.

11 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]` ⟶ LinguisTiX-fonts.

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.

12 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 2_ε's 'meta' font family. It refers to `rm`, `sf` or `tt` in the kernel. 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` ⟨*keyval list*⟩

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

LinguisTiX-fixpex

Documentation | Implementation

No LaTeX3 function provided by this package.

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., 'ı') and Bourbaki's emtpy set symbol (i.e., '∅') is asked by the user.

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

These commands take two arguments, retrieve the values of the data variables if `:VV` variants are 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.

`\lngx_other_main_font:nnn` {⟨*language*⟩} {⟨*features*⟩} {⟨*font*⟩}
`\lngx_other_main_font:nee` {⟨*language*⟩} {⟨*features*⟩} {⟨*font*⟩}
`\lngx_other_sans_font:nnn` {⟨*language*⟩} {⟨*features*⟩} {⟨*font*⟩}
`\lngx_other_sans_font:nee`
`\lngx_other_mono_font:nnn`
`\lngx_other_mono_font:nee`

These commands take three arguments. These are wrapper commands around the font-setting commands of babel. 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-glossing

Documentation | Implementation

`\lngx_gloss_format:n` {⟨*gloss*⟩}
`\lngx_expansion_format:n` {⟨*expansion*⟩}

This function is controlled by the key `format`. Its argument is the gloss or the expansion itself. According to the definition set in the key, the argument gets printed.

`\lngx_gloss_new:nn` {⟨*gloss*⟩} {⟨*expansion*⟩}

This function creates a new gloss. It is later equated with the `\newgloss` command.

`\lngx_gloss_list:`

This functions prints the list of glosses and is equated with `\listofglosses`.

`lngx_multicols` {⟨*section title*⟩}

This environment reads an integer variable, i.e., `\l__lngx_glossary_columns_int`. It is controlled by the `columns` key. If its number is more than one (which, by default *is* more than one), the `multicols` environment is used around the content that comes in between, or else no action is taken. It takes one compulsory argument, i.e., the content of the section title material. This environment should not be used outside this package.

LinguisTiX-ipa

Documentation | Implementation

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

`\lngx_set_main_ipa_font:nn` {⟨*features*⟩} {⟨*font*⟩}
`\lngx_set_main_ipa_font:VV`
`\lngx_main_ipa:`
`lngx_ipa_rm_nfss`

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` {⟨*features*⟩} {⟨*font*⟩}
`\lngx_set_sans_ipa_font:VV`
`\lngx_sans_ipa:`
`lngx_ipa_sf_nfss`

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` {⟨*features*⟩} {⟨*font*⟩}
`\lngx_set_mono_ipa_font:VV`
`\lngx_mono_ipa:`
`lngx_ipa_tt_nfss`

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.

LinguisTiX-Languages

Documentation | Implementation

Here are the L3 functions defined for LinguisTiX-Languages.

`\g_lngx_main_language_tl` A `tl` that globally stores the main language of the document.

`\g_lngx_languages_clist` A `clist` that globally stores the languages that are used.

`\lngx_languages:nn` {⟨*language options*⟩} {⟨*language name*⟩}
`\lngx_languages:VV` ⟨*language options tl*⟩ ⟨*language tl*⟩

These functions read the `V`-type argument provided to them and pass it to the `\babelprovide` command for loading languages.

`\lngx_load_languages:n` {⟨*list of languages*⟩}

This function loads the languages in LinguisTiX sense.

`\lngx_counter:n` This is a developers function provided for printing the counter based on the plug selected. It changes the meaning according to the active value of `native-numbering` socket.

`\lngx_misc_reset:` This function resets a lot of custom settings done by some languages. It has to be used inside `\addto` macro provided by the babel package.

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 tls 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 tls 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}
                    {2026-01-19}
                    {v0.7}
                    {%
                      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-glossing } {
  \RequirePackage { linguistix-glossing }
}
\IfPackageLoadedF { linguistix-ipa } {
  \RequirePackage { linguistix-ipa }
}
\IfPackageLoadedF { linguistix-languages } {
  \RequirePackage { linguistix-languages }
}
\IfPackageLoadedF { linguistix-leipzig } {
  \RequirePackage { linguistix-leipzig }
}
\IfPackageLoadedF { linguistix-logos } {
  \RequirePackage { linguistix-logos }
}
\IfPackageLoadedF { linguistix-nfss } {
  \RequirePackage { linguistix-nfss }
}
⟨/package⟩
```

Set the essentials of the package.

```
⟨*base⟩
\ProvidesExplPackage{linguistix-base}
                    {2026-01-19}
                    {v0.7}
                    {%
                      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* 17.)

\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.)

The unicode-math defines \gla which clashes with the same command defined by the expex package. Of course, the expex-\gla is more relevant in linguistics. Thus I will save that and provide a new command for the unicode-math-\gla. This is not relevant to people who are not using expex. Thus, the settings are loaded only conditionally.

```
⟨*fixpex⟩
\ProvidesExplPackage{linguistix-fixpex}
                    {2026-01-19}
                    {v0.7}
                    {%
                      The base package of the 'LinguisTiX'
                      bundle.%
                    }
```

This package is useful only if either expex or unicode-math is loaded. Otherwise, it is of no use. Thus, I create a message when either of them is not loaded.

```

\msg_new:nnn { fixpex } { pkg_not_loaded } {
  The~ 'LinguisTiX-fixpex'~ package~ is~ a~ first-aid~
  for~ resolving~ the~ conflict~ between~ 'unicode-math'~
  and\\ 'expex'.~ It~ should~ only~ be~ used~ if~ at~ least~
  one~ of~ the\\ two~ is~ loaded.~ Here~
  'LinguisTiX-fixpex'~ can~\\ be~ omitted~ since~ you~ are~
  not~ using~ '#1'.
}
```

I first start the hook begindocument/before.

```

\hook_gput_code:nnn { begindocument / before } { . } {
```

The unicode-math package defines \gla after \begin{document}, so the fix needs to be added after that is done. For that, I start the begindocument/end hook.

```
  \IfPackageLoadedTF { expex } {
    \IfPackageLoadedTF { unicode-math } {
      \hook_gput_code:nnn { begindocument / end } { . } {
```

\umgla This replicates the unicode-math-\gla for future use.

```
        \cs_gset_eq:NN \umgla \gla
```

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

The expex-\gla is then equated to the internal function of the package that does the actual function (Munn and Gregorio 2023).

```
        \cs_gset_eq:NN \gla   \glw@gla
      }
```

In the false branch of unicode-math, I issue an info message that is not visible on the terminal, but is printed in the log file.

```
    } {
      \msg_info:nnn { fixpex } { pkg_not_loaded } {
        unicode-math
      }
    }
```

Similarly, I do it for expex.

```
  } {
    \msg_info:nnn { fixpex } { pkg_not_loaded } {
      expex
    }
  }
}
```
⟨/fixpex⟩

Package essentials first.

```
⟨*font⟩
\ProvidesExplPackage{linguistix-fonts}
                    {2026-01-19}
                    {v0.7}
                    {%
                      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 }
}

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

\LaTeX
\ogLaTeX

We save the original code for the \LaTeX logo and then renew the command.

```

\NewCommandCopy \ogLaTeX \LaTeX

\RenewDocumentCommand \LaTeX {  } {%
  L\kern-.81ex\relax
  \raisebox{.6ex}{\textsc{a}}\kern-.23ex\relax
  \hbox{T}\kern-.4ex\relax
  \raisebox{-.5ex}{E}\kern-.3ex\relax
  X%
}
```

(*End of definition for \LaTeX and \ogLaTeX. These functions are documented on page 5.*)

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 6.*)

\g__lngx_text_main_fonts_prop
\g__lngx_text_main_font_features_tl
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

In the first few versions of the package, I used to save the font-names and their features in token lists, but I found a better way to deal with this later which was using `prop` lists. I had released the `tls` publicly (with a single `_` after the scope marker), which means ideally they should be available forever, but for performance and maintenance the newer approach is much preferred and hence I decided to shift to `prop` lists from v0.6. This time, I am correcting the mistake I made before. The `prop` lists that save the keys is not public. It need not be. Only the key-value pairs are public. They are unchanged anyway. This section describes the implementation of 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.)

```

\prop_gclear_new:N \g__lngx_text_main_fonts_prop
\tl_gclear_new:N \g__lngx_text_main_font_features_tl

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

All the keys here are prefixed with the word `text` in order to distinguish them from the keys provided by the LuaLaTeX-ipa package. The argument of these keys should be expanded for which I use `\prop_gput:Nne` function. Each `#1` is replaced by the items from clist and the loop is repeated, whereas `##1` is the argument passed to the key by user.

```
    \keys_define:nn { lngx_keys } {
      text~ #1
      .code:n                = {
```

I start a group first. Then clear and set a temporary string variable. I make the text of the key titlecased as required by fontspec and remove the spaces. Lastly, the word `Font` is appended. So, `bold italic` becomes `BoldItalicFont`.

```
        \group_begin:
        \str_clear:N \l_tmpa_str
        \str_set:Ne \l_tmpa_str {
          \text_titlecase_all:n { #1 }
          Font
        }
        \str_replace_all:Nnn \l_tmpa_str { ~ } { }
```

The string is used inside the relevant `prop`-key and group is ended.

```
        \prop_gput:Nne \g__lngx_text_main_fonts_prop
                       { text~ #1 }
```

```
                    { \str_use:N \l_tmpa_str = { ##1 } }
      \group_end:
    },
```

Same is repeated for features.

```
    text~ #1~ features
    .code:n                = {
      \group_begin:
      \str_clear:N \l_tmpa_str
      \str_set:Ne \l_tmpa_str {
        \text_titlecase_all:n { #1 }
        Features
      }
      \str_replace_all:Nnn \l_tmpa_str { ~ } { }
      \prop_gput:Nne \g__lngx_text_main_fonts_prop
                    { text~ #1~ features }
                    {
                      \str_use:N \l_tmpa_str = { ##1 }
                    }
      \group_end:
    }
  }
}
```

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

`text extra features` This key adds to the property that stores the extra features for the serif fonts.

```

\keys_define:nn { lngx_keys } {
  text~ extra~ features
  .prop_gput:N           = \g__lngx_text_main_fonts_prop
}
```

(*End of definition for* `text extra features`. *This function is documented on page* 12.)

\g__lngx_text_sans_fonts_prop
\g__lngx_text_sans_font_features_tl
\g__lngx_text_mono_fonts_prop
\g__lngx_text_mono_font_features_tl
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
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

Since the only difference between the upcoming keys is that of the word **sans** and **mono**, we combine them together and use a nested **clist**. The rest of the mechanism is identical.

```
\prop_gclear_new:N \g__lngx_text_sans_fonts_prop
\tl_gclear_new:N \g__lngx_text_sans_font_features_tl

\prop_gclear_new:N \g__lngx_text_mono_fonts_prop
\tl_gclear_new:N \g__lngx_text_mono_font_features_tl

\clist_map_inline:nn {
  sans,
  mono
} {
  \clist_map_inline:nn {
    upright,
    bold~ upright,
    italic,
    bold~ italic,
    slanted,
    bold~ slanted,
    swash,
    bold~ swash,
    small~ caps
  } {
    \keys_define:nn { lngx_keys } {
      text~ #1~ ##1
      .code:n                 = {
        \group_begin:
        \str_clear:N \l_tmpa_str
        \str_set:Ne \l_tmpa_str {
          \text_titlecase_all:n { ##1 }
          Font
        }
        \str_replace_all:Nnn \l_tmpa_str { ~ } { }
        \prop_gput:cne { g__lngx_text_ #1 _fonts_prop }
                       { text~ #1~ ##1 }
                       { ####1 }
        \group_end:
      },
      text~ #1~ ##1~ features
      .code:n                 = {
        \group_begin:
        \str_clear:N \l_tmpa_str
        \str_set:Ne \l_tmpa_str {
          \text_titlecase_all:n { #1 }
          Features
        }
        \str_replace_all:Nnn \l_tmpa_str { ~ } { }
        \prop_gput:cne { g__lngx_text_ #1 _fonts_prop }
                       { text~ #1~ ##1~ features }
                       {
                         \str_use:N \l_tmpa_str = { ####1 }
                       }
        \group_end:
```

```
      }
    }
  }
  \keys_define:nn { lngx_keys } {
    text~ #1~ extra~ features
    .prop_gput:c           = {
                               g__lngx_text_ #1 _fonts_prop
                             }
  }
}
```

(*End of definition for* `\g__lngx_text_sans_fonts_prop` *and others. These functions are documented on page* 11.)

`\g__lngx_text_main_font_tl`
`\g__lngx_text_sans_font_tl`
`\g__lngx_text_mono_font_tl`
`text main font`
`text sans font`
`text mono font`

These keys add the parameter that sets the main font for text. They set an internal token list which is retrieved later by font setting command.

```

\clist_map_inline:nn {
  main,
  sans,
  mono
} {
  \keys_define:nn { lngx_keys } {
    text~ #1~ font
    .tl_gset:c             = { g__lngx_text_ #1 _font_tl }
  }
}
```

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

`\g__lngx_math_fonts_prop`
`\g__lngx_math_features_tl`
`\g__lngx_math_bold_fonts_prop`
`\g__lngx_math_bold_features_tl`
`math`
`math features`
`math bold`
`math bold features`

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

```

\prop_gclear_new:N \g__lngx_math_fonts_prop
\tl_gclear_new:N \g__lngx_math_features_tl

\prop_gclear_new:N \g__lngx_math_bold_fonts_prop
\tl_gclear_new:N \g__lngx_math_bold_features_tl

\keys_define:nn { lngx_keys } {
  math
  .tl_gset:N               = \g__lngx_math_font_tl,
  math~ bold
  .tl_gset:N               = \g__lngx_math_bold_font_tl,
  math~ features
  .prop_gput:N           = \g__lngx_math_fonts_prop,
  math~ bold~ features
  .prop_gput:N           = \g__lngx_math_bold_fonts_prop
}
```

(*End of definition for* `\g__lngx_math_fonts_prop` *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 book weight and in all of the serif, sans serif and monospaced families.

```

\keys_define:nn { lngx_keys } {
```

```
  newcm
  .meta:n                 = {
    text~ main~ font        = { NewCM10-Book.otf },
    text~ sans~ font        = { NewCMSans10-Book.otf },
    text~ mono~ font        = { NewCMMono10-Book.otf },
    math                    = { NewCMMath-Book.otf },
    math~ bold              = { NewCMMath-Bold.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                 = {
    main~ font              = { NewCMSans10-Book.otf },
    sans~ font              = { NewCMSans10-Book.otf },
    mono~ font              = { NewCMMono10-Book.otf },
    math                    = { NewCMSansMath-Regular.otf },
    math~ bold              = { NewCMSansMath-Regular.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                 = {
    main~ font              = { NewCMMono10-Book.otf },
    sans~ font              = { NewCMSans10-Book.otf },
    mono~ font              = { NewCMMono10-Book.otf },
    math                    = { NewCMSansMath-Regular.otf },
    math~ bold              = { NewCMSansMath-Regular.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                 = {
    main~ font              = { NewCM10-Regular.otf },
    sans~ font              = { NewCMSans10-Regular.otf },
    mono~ font              = { NewCMMono10-Regular.otf },
    math                    = { NewCMMath-Regular.otf },
    math~ bold              = { NewCMMath-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                 = {
    main~ font              = { NewCMSans10-Regular.otf },
    sans~ font              = { NewCMSans10-Regular.otf },
    mono~ font              = { NewCMMono10-Regular.otf },
    math                    = { NewCMMath-Regular.otf },
    math~ bold              = { NewCMMath-Bold.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                 = {
    main~ font              = { NewCMMono10-Regular.otf },
    sans~ font              = { NewCMSans10-Regular.otf },
    mono~ font              = { NewCMMono10-Regular.otf },
    math                    = { NewCMMath-Regular.otf },
    math~ bold              = { NewCMMath-Bold.otf },
  }
}
```

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

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

```
\lngx_set_keys:n {
  bourbaki's~ empty~ set,
```

Then we load the `old style numbers` boolean.

```
  old~ style~ numbers,
  newcm
}
```

`\lngx_set_main_font:nn` `\lngx_set_sans_font:nn` `\lngx_set_mono_font:nn` `\lngx_set_math_font:nn` If LinguisTiX-Languages package is loaded, I load the fonts with `\babelfont` command. In case it is not loaded, the fonts are set with `\setxxxxcommand`-type commands provided by fontspec.

```
\IfPackageLoadedF { linguistix-languages } {
  \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 }
  }
}
```

A wrapper command is provided for loading math fonts.

```

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

\cs_new_protected:Npn \lngx_set_math_bold_font:nn #1#2 {
  \setmathfont [
    #1,
    version              = { bold }
  ] { #2 }
}
```

All of these commands should expand their arguments, so I provide the appropriate variants.

```

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

(*End of definition for* `\lngx_set_main_font:nn` *and others. These functions are documented on page 18.*)

```
\__lngx_build_main_font_features:
\__lngx_build_sans_font_features:
\__lngx_build_mono_font_features:
\__lngx_build_math_font_features:
\__lngx_build_bold_math_font_features:
\g__lngx_text_main_font_features_tl
\g__lngx_text_sans_font_features_tl
\g__lngx_text_mono_font_features_tl
\g__lngx_math_font_features_tl
\g__lngx_bold_math_font_features_tl
```

These are some internal functions that basically iterate on the `prop` list items and each of them is put to the right of the respective token list. This way only the functions that are added by the user are exported to the font setting command.

```

\clist_map_inline:nn {
  main,
  sans,
  mono
} {
  \cs_new_protected:cpn {
    __lngx_build_ #1 _font_features:
  } {
    \prop_map_inline:cn { g__lngx_text_ #1 _fonts_prop } {
      \tl_gput_right:cn {
                            g__lngx_text_ #1 _font_features_tl
                          } { ####2 }
    }
  }
}

\cs_new_protected:Npn \__lngx_build_math_features: {
  \prop_map_inline:Nn \g__lngx_math_fonts_prop {
    \tl_gput_right:Nn \g__lngx_math_features_tl {
      { ##2 }
```

```
    }
  }
}

\cs_new_protected:Npn \__lngx_build_math_bold_features: {
  \prop_map_inline:Nn \g__lngx_math_bold_fonts_prop {
    \tl_gput_right:Nn \g__lngx_math_bold_features_tl {
      { ##2 }
    }
  }
}
```

(*End of definition for* `\__lngx_build_main_font_features:` *and others.*)

Now I start the pre-`begindocument` hook.

```
\hook_gput_code:nnn { begindocument / before } { . } {
```

If the boolean for old style numbers is true, I set the `Numbers` key to `OldStyle`. Similarly, if the NewCM-specific old one is requested, I turn the character-variant on.

```
  \lngx_set_keys:n {
    text~ extra~
    features               = {
      \bool_if:NT \g_lngx_old_style_bool {
        Numbers              = { OldStyle },
        \bool_if:NT \g_lngx_old_style_one_bool {
          CharacterVariant     = { 6 }
        }
      }
    },
    text~ sans~ extra~
    features               = {
      \bool_if:NT \g_lngx_old_style_bool {
        Numbers              = { OldStyle },
        \bool_if:NT \g_lngx_old_style_one_bool {
          CharacterVariant     = { 6 }
        }
      }
    }
  }
```

All the font features are built using the internal functions and then fonts are set.

```
  \__lngx_build_main_font_features:
  \lngx_set_main_font:VV
    \g__lngx_text_main_font_features_tl
    \g__lngx_text_main_font_tl
  \__lngx_build_sans_font_features:
  \lngx_set_sans_font:VV
    \g__lngx_text_sans_font_features_tl
    \g__lngx_text_sans_font_tl
  \__lngx_build_mono_font_features:
  \lngx_set_mono_font:VV
    \g__lngx_text_mono_font_features_tl
    \g__lngx_text_mono_font_tl
  \__lngx_build_math_features:
  \lngx_set_math_font:VV \g__lngx_math_features_tl
```

```
                          \g__lngx_math_font_tl
  \__lngx_build_math_bold_features:
  \lngx_set_math_bold_font:VV \g__lngx_math_bold_features_tl
                              \g__lngx_math_bold_font_tl
}
```
⟨/font⟩

```
⟨*glossing⟩
\ProvidesExplPackage{linguistix-glossing}
                     {2026-01-19}
                     {v0.7}
                     {%
                       Accessible glossing with LinguisTiX%
                     }
```

In order to print the multi-column glossary, I load the `\multicol` package.

```

\IfPackageLoadedF { multicol } {
  \RequirePackage { multicol }
}
```

I generate expansion-variants for kernel commands.

```

\cs_generate_variant:Nn \seq_if_in:NnF { Ne  }
```

Then I declare some variables that will be used for generating the glossing-auxiliary.

```

\bool_new:N       \l_lngx_expansion_bool
\tl_clear_new:N   \l_lngx_gloss_separator_tl
\tl_clear_new:N   \l_lngx_expansion_separator_tl
\tl_clear_new:N   \l_lngx_glossary_separator_tl
\dim_zero_new:N   \l_lngx_i_have_dim
\dim_zero_new:N   \l_lngx_i_need_dim
\dim_zero_new:N   \l_lngx_remain_dim
\dim_zero_new:N   \l_lngx_i_hack_dim
\int_gzero_new:N  \g__lngx_page_ref_int
\str_clear_new:N  \l_lngx_gls_language_str
\str_clear_new:N  \l__lngx_gls_sorting_order_str
\str_clear_new:N  \l__lngx_gls_expansion_case_str
\str_clear_new:N  \l__lngx_glossary_style_str
\str_clear_new:N  \l__lngx_separator_str
\seq_gclear_new:N \g__lngx_gls_use_order_seq

\str_set:Nn \l__lngx_separator_str { gloss }
```

Glossaries are hyperlinked with complex and cryptic labels. Some readers read the labels loudly when using assistive technology. In order to dodge that, I add the text to the Contents key. It uses Ulrike's ideas: tex.stackexchange.com/a/758083/174620.

```

\socket_if_exist:nT { hyp / link / GoTo / Contents } {
  \socket_new_plug:nnn { hyp / link / GoTo / Contents }
                       { text } {
    \pdfstringdef \__lngx_tmp_text: { #2 }
    \pdfannot_dict_put:nne { link / GoTo } { Contents } {
      ( \__lngx_tmp_text: )
    }
  }
}
```

After these initial declarations, I move to the socket that controls the description of the gloss. The socket itself has no arguments.

```

```

```
\socket_new:nn { lngx / description / gloss } { 0 }
```

__lngx_gloss_description:

When the socket is assigned the `on` plug, it defines the expandable internal command for glossing description. It is then used inside the tagging socket. The same command is made inactive when the socket is assigned the `off` plug. By default the `off` plug is assigned (this is experimental and may change after reviews from the blind people). The socket is activated by using it.

```

\socket_new_plug:nnn { lngx / description / gloss } { on } {
  \cs_set:Npn \__lngx_gloss_description: { Gloss~ }
}

\socket_new_plug:nnn { lngx / description / gloss }
                     { off } {
  \cs_set_eq:NN \__lngx_gloss_description: \prg_do_nothing:
}

\socket_assign_plug:nn { lngx / description / gloss }
                       { off }

\socket_use:n { lngx / description / gloss }
```

(*End of definition for* `\__lngx_gloss_description:`.)

Then I declare the tagging socket for glossing which takes two arguments. It should follow the default tagging which is why I use the `default` plug (which is the only plug the package does and will offer). The code is based on suggestions by Ulrike Fischer (github.com/latex3/tagging-project/discussions/975). The `E` tag is used for 'expansion' which more or less suits the nature of glosses. So it is used here. The command `\__lngx_gloss_description:` is controlled by the socket and is expandable.

```

\NewTaggingSocket { lngx / gloss } { 2 }

\NewTaggingSocketPlug { lngx / gloss } { default } {
  \mode_leave_vertical:
  \tag_mc_end:
  \exp_args:Ne
  \tag_struct_begin:n {
    tag                    = { Span },
    E                      = {
      \__lngx_gloss_description: #2
    }
  }
  \tag_mc_begin:n {
    tag                    = { Span }
  }
```

The argument is printed with the package-controlled formatting command. First I check if the hyperref package is loaded. If it is loaded, the link colour is changed to the one stored in the variable `\g_lngx_gloss_link_color_str` (black, by default).

```
  \IfPackageLoadedTF { hyperref } {
    \group_begin:
    \str_clear:N \l_tmpa_str
    \str_set:Nn \l_tmpa_str { #1 }
    \exp_args:Ne \hypersetup {
```

```
      linkcolor            = {
        \exp_not:V \g__lngx_gloss_link_color_str
      }
    }
```

The socket for adding text into the Contents directory is used here.

```
    \socket_if_exist:nT { hyp / link / GoTo / Contents } {
      \socket_assign_plug:nn {
                                hyp / link / GoTo / Contents
                              }
                              { text }
    }
    \lngx_gloss_format:n {
      \hyperlink { lngx_ #1 _glossary } { #1 }
    }
    \group_end:
  } {
```

If hyperref is not loaded, the text is simply printed with the formatting command.

```
    \lngx_gloss_format:n { #1 }
  }
  \tag_mc_end:
  \tag_struct_end:
  \tag_mc_begin:n { }
}
```

I assign the default tagging plug to the socket I just defined.

```

\AssignTaggingSocketPlug { lngx / gloss } { default }
```

format Now I define the key for adjusting the formatting of the glosses. It controls several keys contained in a separate set. In short, this key will take another keys as arguments.

```

\keys_define:nn { lngx_glossing } {
  format
  .meta:nn               = { lngx / gloss / format } { #1 },
```

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

link color This option sets the colour used for glossing links. It is set to `black` by default.

\g__lngx_gloss_link_color_str

```
  link~ color
  .str_gset:N            = \g__lngx_gloss_link_color_str,
  link~ color
  .initial:n             = { black },
```

(*End of definition for* `link color` *and* `\g__lngx_gloss_link_color_str`. *This function is documented on page 8.*)

sort Glosses can be sorted alphabetically or as they are used. The choice key for that is as follows. By default glosses are sorted alphabetically.

\l__lngx_gls_sorting_order_str

```
  sort
  .choices:nn            = { alphabetical, use } {
    \str_set_eq:NN \l__lngx_gls_sorting_order_str
                   \l_keys_choice_str
  },
  sort
  .initial:n             = { alphabetical },
```

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

expansion case
`\l__lngx_gls_expansion_case_str`

The expansion can be printed in lower case, title case (with the first letter capitalised for all the words) or title case (with the first letter capitalised only for the first word). The default is lower case.

```
expansion~ case
.choices:nn            = {
  lowercase, title~ case~ all, title~ case~ first
} {
  \str_set_eq:NN \l__lngx_gls_expansion_case_str
                 \l_keys_choice_str
},
expansion~ case
.initial:n             = { lowercase },
```

(*End of definition for* `expansion case` *and* `\l__lngx_gls_expansion_case_str`. *This function is documented on page 8.*)

style
`\l__lngx_glossary_style_str`

The glossary can be printed in two styles given below. The default is `block`.

```
style
.choices:nn            = { block, inline } {
  \str_set_eq:NN \l__lngx_glossary_style_str
                 \l_keys_choice_str
},
style
.initial:n             = { block },
```

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

columns
`\l__lngx_glossary_columns_int`

There is an option to change the number of columns used for printing the glossary. It is controlled here. Default is 2.

```
columns
.int_set:N             = \l__lngx_glossary_columns_int,
columns
.initial:n             = { 2 },
```

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

page numbers
`\l__lngx_glosses_page_number_bool`

Page numbers can be turned off with the following boolean. By default, they are active.

```
page~ numbers
.bool_set:N            = \l__lngx_glosses_page_number_bool,
page~ numbers
.initial:n             = { true },
```

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

sectioning
`\l__lngx_gls_sectioning_str`

The section used for printing the glossary title is controlled by the following command. By default, I use `\section` for printing the title.

```
sectioning
.str_set:N             = \l__lngx_gls_sectioning_str,
sectioning
.initial:n             = { section },
```

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

section number
\l__lngx_gls_section_number_bool

This controls if the sectioning level should be numbered or unnumbered. The default is false.

```
  section~ number
  .bool_set:N             = \l__lngx_gls_section_number_bool,
  section~ number
  .initial:n              = { false },
```

(*End of definition for* `section number` *and* `\l__lngx_gls_section_number_bool`. *This function is documented on page* 9.)

no bold
\l__lngx_gls_bold_bool

The `no bold` key is defined as an inverse boolean. By default the key is set to false (resulting in the controlled boolean being true).

```
  no~ bold
  .bool_set_inverse:N     = \l__lngx_gls_bold_bool,
  no~ bold
  .initial:n              = { false },
```

(*End of definition for* `no bold` *and* `\l__lngx_gls_bold_bool`. *This function is documented on page* 9.)

separator
\l__lngx_separator_tl

The separator between the glosses is controlled using this key. It controls the separator for inline glosses, expansion of glosses as well as glosses seen in the glossary. Each of these functions set a string variable which is expanded when this key is used. The default value of the string variable is gloss and the default value for this key is `,~`, which means by default the separator between glosses is a comma followed by a space.

```
  separator
  .code:n                 = {
    \tl_set:cn {
      l_lngx_
      \str_use:N \l__lngx_separator_str
      _separator_tl
    } { #1 }
  },
  separator
  .initial:n              = { ,~ },
```

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

entry separator
\l__lngx_entry_separator_tl

The separator between glossary entries is controlled using this key. The default is a `\par` token.

```
  entry~ separator
  .tl_set:N               = \l__lngx_entry_separator_tl,
  entry~ separator
  .initial:n              = { \par }
}
```

(*End of definition for* `entry separator` *and* `\l__lngx_entry_separator_tl`. *This function is documented on page* 9.)

Sometimes language-specific settings are needed. I define the language string variable with the information retrieved from the lang key of the PDF.

```

\str_set:Ne \l_lngx_gls_language_str {
  \GetDocumentProperties { document / lang }
}
```

gloss
\lngx_gloss_format:n

The formatting of glosses is defined here. By default they are printed in small caps.

```
\keys_define:nn { lngx / gloss / format  } {
  gloss
  .cs_gset_protected:Np  = \lngx_gloss_format:n #1,
  gloss
  .initial:n             = { \textsc { #1 } },
```

(*End of definition for* `gloss` *and* `\lngx_gloss_format:n`*. These functions are documented on page 8.*)

expansion
\lngx_expansion_format:n

The formatting of expansions is defined here. There is no change in the printing in the defaults.

```
  expansion
  .cs_gset_protected:Np  = \lngx_expansion_format:n #1,
  expansion
  .initial:n             = { #1 }
}
```

(*End of definition for* `expansion` *and* `\lngx_expansion_format:n`*. These functions are documented on page 8.*)

\setupglossing

A wrapper around these options is provided.

```
\NewDocumentCommand \setupglossing { m } {
  \keys_set:nn { lngx_glossing } { #1 }
}
```

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

\newgloss
\lngx_gloss_new:nn

A function that creates new glosses starts here. It takes 2 arguments.

```
\cs_new_protected:Npn \lngx_gloss_new:nn #1#2 {
```

First and foremost, the string received as the first argument should change its case to lowercase. It is done by `\str_lowercase:n`. I will use a temporary string variable for storing the converted value. This needs to be done locally so I start a group and clear the local `str` variable.

```
  \group_begin:
  \str_clear:N \l_tmpa_str
  \str_set:Ne \l_tmpa_str { \str_lowercase:n { #1 } }
```

Every gloss has its expansion stored in a token list associated to it. The token list is declared here and it is set to the expansion (i.e., `#2`).

```
  \tl_gclear_new:c {
    g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
  }
  \seq_gclear_new:c {
    g_lngx_ \str_use:N \l_tmpa_str _pages_seq
  }
  \tl_gset:cn {
    g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
  } { #2 }
```

Whenever a gloss is defined, an internal protected command is defined. It doesn't take any argument.

```
  \cs_new_protected:cpn {
    __lngx_gloss_ \str_use:N \l_tmpa_str :
  } {
```

The arguments are passed to the tagging socket. Since the tagging socket doesn't expand everything, an exhaustive expansion is performed with the help of `\exp_args:Nee`.

```
    \exp_args:Nee \UseTaggingSocket
                  { lngx / gloss }
                  { \str_use:N \l_tmpa_str }
                  { #2 }
```

The kernel provides `\seq_remove_duplicates:N`, but as it iterates on each and every item, it is slow. The duplicates can be avoided if the items are added to the sequence conditionally and only when they don't exist already in the sequence. This way duplicates are not generated at all. This method is used for adding the page numbers to the sequence. Imagine if a gloss is used twice on a page, it doesn't make sense to add the same page number twice. I use `\label`-`\ref` mechanism for saving the page numbers of the glosses. An internal integer called `\g__lngx_page_ref_int` is used to generate unique numbers.

```
    \int_gincr:N \g__lngx_page_ref_int
    \exp_args:Ne
    \label { lngx_gloss_ \int_use:N \g__lngx_page_ref_int }
    \exp_args:Nee
    \seq_if_in:ceF {
      g_lngx_ \str_use:N \l_tmpa_str _pages_seq
    } {
      \pageref {
        lngx_gloss_ \int_use:N \g__lngx_page_ref_int
      }
    } {
      \seq_gput_right:ce {
        g_lngx_ \str_use:N \l_tmpa_str _pages_seq
      } {
        \pageref {
          lngx_gloss_ \int_use:N \g__lngx_page_ref_int
        }
      }
    }
```

The same logic is used for the sequence that stores the glosses in the order they are used.

```
    \seq_if_in:NeF \g__lngx_gls_use_order_seq {
      \str_use:N \l_tmpa_str
    } {
      \seq_gput_right:Ne \g__lngx_gls_use_order_seq
                         { \str_use:N \l_tmpa_str }
    }
  }
  \group_end:
}

\cs_gset_eq:NN \newgloss \lngx_gloss_new:nn
```

(*End of definition for* `\newgloss` *and* `\lngx_gloss_new:nn`. *These functions are documented on page* 8.)

\renewgloss Implementing the \renewgloss command is actually quite easy. The definition of \lngx_gloss_new:nn uses only a single command that errors if the control sequence is already defined, i.e., \cs_new_protected:cpn. In order to renew a gloss, simply undefining the existing command declared with \lngx_gloss_new:nn suffices. Later the arguments are passed to the same command again. No LaTeX3 equivalent for this is provided.

```
\NewDocumentCommand \renewgloss { m m } {
  \cs_undefine:c { __lngx_gloss_ #1 : }
  \lngx_gloss_new:nn { #1 } { #2 }
}
```

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

\glx The command to use a gloss takes three arguments where the first is an optional asterisk. If it is used, the expansion of the gloss is printed without any special tags, just as plain text. Otherwise the internal command for printing the gloss is used with the third argument. The third argument is a clist. Any number of glosses can be added to the list. The action is then repeated on each and every item of the list. The second argument is a list of options for customising the output. Everything is computed locally so that for the settings don't leak. I perform the action on the first item as desired, then the same is applied to the remaining items with a preceding separator. So that all the items are separated properly.

```
\NewDocumentCommand \glx { s O{ } m } {
  \group_begin:
  \IfBooleanT { #1 } {
    \bool_set_true:N \l_lngx_expansion_bool
    \str_set:Nn \l__lngx_separator_str { expansion }
    \keys_set:nn { lngx_glossing } {
      separator              = { , \c_space_tl }
    }
  }
  \keys_set:nn { lngx_glossing } { #2 }
  \tl_clear:N \l_tmpa_tl
  \seq_clear:N \l_tmpa_seq
  \seq_set_from_clist:Nn \l_tmpa_seq { #3 }
  \seq_pop_left:NN \l_tmpa_seq \l_tmpa_tl
  \str_set:Ne \l_tmpa_str {
    \exp_args:Ne \str_lowercase:n {
      \tl_use:N \l_tmpa_tl
    }
  }
  \bool_if:NTF \l_lngx_expansion_bool {
    \str_case:Vn \l__lngx_gls_expansion_case_str {
      { lowercase } {
        \text_lowercase:n {
          \tl_use:c {
            g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
          }
        }
      }
      { title~ case~ all } {
        \text_titlecase_all:n {
```

```
          \tl_use:c {
            g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
          }
        }
      }
      { title~ case~ first } {
        \text_titlecase_first:n {
          \tl_use:c {
            g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
          }
        }
      }
    }
  } {
    \use:c { __lngx_gloss_ \str_use:N \l_tmpa_str : }
  }
  \seq_if_empty:NF \l_tmpa_seq {
    \seq_map_inline:Nn \l_tmpa_seq {
      \group_begin:
      \str_clear:N \l_tmpa_str
      \str_set:Ne \l_tmpa_str {
        \exp_args:Ne \str_lowercase:n { ##1 }
      }
      \bool_if:NTF \l_lngx_expansion_bool {
        \str_case:Vn \l__lngx_gls_expansion_case_str {
          { lowercase } {
            \tl_use:N \l_lngx_expansion_separator_tl
            \text_lowercase:n {
              \tl_use:c {
                g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
              }
            }
          }
          { title~ case~ all } {
            \tl_use:N \l_lngx_expansion_separator_tl
            \text_titlecase_all:n {
              \tl_use:c {
                g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
              }
            }
          }
          { title~ case~ first } {
            \tl_use:N \l_lngx_expansion_separator_tl
            \text_titlecase_first:n {
              \tl_use:c {
                g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
              }
            }
          }
        }
      } {
        \tl_use:N \l_lngx_gloss_separator_tl
        \use:c { __lngx_gloss_ \str_use:N \l_tmpa_str : }
      }
```

```
      \group_end:
    }
  }
  \group_end:
}
```

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

`\__lngx_dotfill:nnn` For the dotfill between the gloss and the expansion, I create a custom internal command. The code is based on user Jonathan P. Spratte's answer seen here: topanswers.xyz/tex?q=8155#a7758. The dotfill should not be tagged at all and in fact it should be suppressed so that the readers don't go 'dot, dot, dot, dot …' (Frank has convinced us forever with his TUG 2025 talk).

```
\cs_new_protected:Npn \__lngx_dotfill:nnn #1#2#3 {
  %% Courtesy: Jonathan P. Spratte
  %% topanswers.xyz/tex?q=8155#a7758 (LPPL)
  \l__lngx_entry_separator_tl
  \smallskip
  \group_begin:
  \rightskip             = 0pt plus -1fil \prg_do_nothing:
  \parfillskip           = 0pt plus 1fil  \prg_do_nothing:
  \leftskip              = 1em plus 1fil  \prg_do_nothing:
  \finalhyphendemerits   = 0              \prg_do_nothing:
  \parindent             = -1em           \prg_do_nothing:
  \bool_if:NT \l__lngx_gls_bold_bool { \textbf } {
    \lngx_gloss_format:n {
      #1
    }
    \tl_use:N \l_lngx_glossary_separator_tl
  }
  #2
  \leavevmode
  \quad
  \cleaders
    \hbox to 0.44em { \hss . \hss }
    \hskip 0.5cm plus 1fill \prg_do_nothing:
  \quad
  \kern 0pt \prg_do_nothing:
  \em #3
  \l__lngx_entry_separator_tl
  \group_end:
}
```

(*End of definition for* `\__lngx_dotfill:nnn`.)

`lngx_multicols` Here I define the custom multicolumn environment which does nothing if the number of columns is 1.

```
\NewDocumentEnvironment { lngx_multicols } { m } {
  \int_compare:nNnTF { 1 } < {
    \int_use:N \l__lngx_glossary_columns_int
  } {
    \begin { multicols } {
```

```
      \int_use:N \l__lngx_glossary_columns_int
    } [ #1 ]
  } { #1 }
  \noindent
} {
  \int_compare:nNnT { 1 } < {
    \int_use:N \l__lngx_glossary_columns_int
  } {
    \end { multicols }
  }
}
```

(*End of definition for* `lngx_multicols`. *This function is documented on page 18.*)

`\lngx_gloss_list:` Finally we come to the command that prints the glosses. First it sets the boolean for creating the aux file to false.

```

\cs_new_protected:Npn \lngx_gloss_list: {
  \bool_gset_false:N \g_lngx_trigger_aux_file_bool
```

I start a group, clear a scratch sequence and set it equal to the sequence that stores the order of the glosses. If the aux file is read, the `aux` flag is added to the variable, or else it is read on the fly.

```
  \group_begin:
  \seq_clear:N \l_tmpa_seq
  \seq_set_eq:NN \l_tmpa_seq \g__lngx_gls_use_order_seq
```

If the sorting order is set to `alphabetical`, the sequence needs to get sorted. For that, I use LaTeX3's mechanism for sorting strings.

```
  \str_case:Vn \l__lngx_gls_sorting_order_str {
    { alphabetical } {
      \seq_sort:Nn \l_tmpa_seq {
        \str_compare:nNnTF { ##1 } > { ##2 } {
          \sort_return_swapped:
        } {
          \sort_return_same:
        }
      }
    }
  }
```

If the style used is `inline`, the glosses come after the each other. That means the default entry separator, i.e., `\par` must be changed. Here I set it to `,~` by default (locally). The separator between the gloss and the entry is defined as a colon followed by a space.

```
  \str_if_eq:VnTF \l__lngx_glossary_style_str { inline } {
    \group_begin:
    \keys_set:nn { lngx_glossing } {
      separator              = { \c_colon_str \c_space_tl },
      entry~ separator       = { ,~ }
    }
```

Then each item from the sequence is popped (from the left). It is then passed to a string variable to get rid of the catcodes. The string variable is then used in `\MakeLinkTarget*`. The gloss is then printed with its separator in bold shape.

```
    \tl_clear:N \l_tmpa_tl
```

```
    \str_clear:N \l_tmpa_str
    \seq_pop_left:NN \l_tmpa_seq \l_tmpa_tl
    \str_set:NV \l_tmpa_str \l_tmpa_tl
    \tag_mc_end:
    \tag_struct_begin:n {
      tag                     = { Span },
    }
    \tag_mc_begin:n {
      tag                     = { Span }
    }
    \MakeLinkTarget * {
      lngx_ \str_use:N \l_tmpa_str _glossary
    }
    \bool_if:NT \l__lngx_gls_bold_bool { \textbf } {
      \lngx_gloss_format:n {
        \tl_use:N \l_tmpa_tl
        \tl_use:N \l_lngx_glossary_separator_tl
      }
    }
    \tag_mc_end:
    \tag_struct_end:
```

Then it is checked in which case the expansion is requested. According to that the `tl` is printed.

```
    \str_case:Vn \l__lngx_gls_expansion_case_str {
      { lowercase } {
        \lngx_expansion_format:n {
          \text_lowercase:n {
            \tl_use:c {
              g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
            }
          }
        }
      }
      { title~ case~ all } {
        \lngx_expansion_format:n {
          \text_titlecase_all:n {
            \tl_use:c {
              g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
            }
          }
        }
      }
      { title~ case~ first } {
        \lngx_expansion_format:n {
          \text_titlecase_first:n {
            \tl_use:v {
              g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
            }
          }
        }
      }
    }
```

After printing one entry successfully, if there are any more items left in the sequence, they

are printed with the same method, but with an entry separator at the beginning.

```
    \seq_if_empty:NF \l_tmpa_seq {
      \seq_map_inline:Nn \l_tmpa_seq {
        \group_begin:
        \tl_use:N \l__lngx_entry_separator_tl
        \MakeLinkTarget * { lngx_ ##1 _glossary }
        \textbf {
          \lngx_gloss_format:n {
            ##1
            \tl_use:N \l_lngx_glossary_separator_tl
          }
        }
        \str_case:Vn \l__lngx_gls_expansion_case_str {
          { lowercase } {
            \lngx_expansion_format:n {
              \text_lowercase:n {
                \exp_not:v { g_lngx_ ##1 _expansion_tl }
              }
            }
          }
          { title~ case~ all } {
            \lngx_expansion_format:n {
              \text_titlecase_all:n {
                \exp_not:v { g_lngx_ ##1 _expansion_tl }
              }
            }
          }
          { title~ case~ first } {
            \lngx_expansion_format:n {
              \text_titlecase_first:n {
                \exp_not:v { g_lngx_ ##1 _expansion_tl }
              }
            }
          }
        }
        \group_end:
      }
    }
    \group_end:
  } {
```

If the style is not `inline`, then the default `block` style is assumed and firstly the word 'glossary' is printed in a sectioning command controlled by the keys. The `\glossaryname` command is provided by babel. If it is undefined, that means the user hasn't loaded babel. In that case, I define the command with the string `Glossary`.

```
    \ProvideDocumentCommand \glossaryname { } { Glossary }
```

Then the `lngx_multicols` environment starts which doesn't do anything if the number of columns is 1.

```
    \begin { lngx_multicols } {
      \str_if_eq:VnF \l__lngx_gls_sectioning_str { null } {
        \use:e {
          \exp_not:N \use:c
          { \str_use:N \l__lngx_gls_sectioning_str }
```

```
          \bool_if:NF \l__lngx_gls_section_number_bool { * }
          { \exp_not:N \glossaryname }
        }
      }
    }
      \seq_map_inline:Nn \l_tmpa_seq {
```

In this style, even the page numbers are printed along with glosses. We save the page numbers in a temporary sequence which is locally cleared.

```
        \group_begin:
        \seq_clear:N \l_tmpb_seq
        \seq_map_inline:cn { g_lngx_ ##1 _pages_seq } {
```

The pages are hyperlinked with the internal PDF names.

```
          \seq_put_right:Ne \l_tmpb_seq { ####1 }
        }
```

The page numbers are separated using dotfill. Before the glosses, `\MakeLinkTarget*` is used.

```
        \__lngx_dotfill:nnn {
          \MakeLinkTarget * { lngx_ ##1 _glossary }
          ##1
        } {
```

The case of expansion is checked and then the appropriate casing commands are used for expansions.

```
          \str_case:Vn \l__lngx_gls_expansion_case_str {
            { lowercase } {
              \lngx_expansion_format:n {
                \text_lowercase:n {
                  \exp_not:v { g_lngx_ ##1 _expansion_tl }
                }
              }
            }
            { title~ case~ all } {
              \lngx_expansion_format:n {
                \text_titlecase_all:n {
                  \exp_not:v { g_lngx_ ##1 _expansion_tl }
                }
              }
            }
            { title~ case~ first } {
              \lngx_expansion_format:n {
                \text_titlecase_first:n {
                  \exp_not:v { g_lngx_ ##1 _expansion_tl }
                }
              }
            }
          }
        } {
```

The list of page numbers is printed.

```
          \seq_use:Nn \l_tmpb_seq { ,~ }
        }
        \group_end:
      }
```

```
    \end { lngx_multicols }
  }
  \group_end:
}
```

(*End of definition for* `\lngx_gloss_list:`*. This function is documented on page 18.*)

`\listofglosses` Here is the command that defines the user-side command for printing the glosses. It defines the separator by default if not provided. All settings are local in order to avoid leaking. `\l_lngx_separator_tl` is the generic string that is used inside the `separator` key that sets the separator contextually. This command uses the LaTeX3 function for printing the glosses.

```

\NewDocumentCommand \listofglosses { O { } } {
  \group_begin:
  \str_set:Nn \l__lngx_separator_str { glossary }
  \keys_set:nn { lngx_glossing } {
    separator            = { \c_colon_str \c_space_tl }
  }
  \keys_set:nn { lngx_glossing } { #1 }
  \lngx_gloss_list:
  \group_end:
}
⟨/glossing⟩
```

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

```
⟨*ipa⟩
\ProvidesExplPackage{linguistix-ipa}
                    {2026-01-19}
                    {v0.7}
                    {%
                      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 }
}

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

\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* 10.)

\g__lngx_ipa_main_fonts_prop
\g__lngx_ipa_main_font_features_tl
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

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

```
\prop_gclear_new:N \g__lngx_ipa_main_fonts_prop
\tl_gclear_new:N \g__lngx_ipa_main_font_features_tl

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

All the keys here are prefixed with the word `ipa` in order to distinguish them from the keys provided by the LinguisTiX-fonts package. These keys have identical method as their text counterparts, though.

```
    \keys_define:nn { lngx_keys } {
      ipa~ #1
      .code:n                 = {
        \group_begin:
        \str_clear:N \l_tmpa_str
        \str_set:Ne \l_tmpa_str {
          \text_titlecase_all:n { #1 }
          Font
        }
        \str_replace_all:Nnn \l_tmpa_str { ~ } { }
        \prop_gput:Nne \g__lngx_ipa_main_fonts_prop
                       { ipa~ #1 }
                       { \str_use:N \l_tmpa_str = { ##1 } }
        \group_end:
      },
      ipa~ #1~ features
      .code:n                 = {
        \group_begin:
        \str_clear:N \l_tmpa_str
        \str_set:Ne \l_tmpa_str {
          \text_titlecase_all:n { #1 }
          Features
        }
        \str_replace_all:Nnn \l_tmpa_str { ~ } { }
        \prop_gput:Nne \g__lngx_ipa_main_fonts_prop
                       { ipa~ #1~ features }
                       {
                         \str_use:N \l_tmpa_str = { ##1 }
                       }
        \group_end:
      }
    }
}
```

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

`ipa extra features` This key adds to the property that stores the extra features for the serif fonts.

```
\keys_define:nn { lngx_keys } {
  ipa~ extra~ features
  .prop_gput:N           = \g__lngx_ipa_main_fonts_prop
}
```

(*End of definition for* `ipa extra features`*. This function is documented on page* 12.)

\g__lngx_ipa_sans_fonts_prop
\g__lngx_ipa_sans_font_features_tl
\g__lngx_ipa_mono_fonts_prop
\g__lngx_ipa_mono_font_features_tl
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
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

Since the only difference between the upcoming keys is that of the word **sans** and **mono**, we combine them together and use a nested **clist**. The rest of the mechanism is identical.

```
\prop_gclear_new:N \g__lngx_ipa_sans_fonts_prop
\tl_gclear_new:N \g__lngx_ipa_sans_font_features_tl
\prop_gclear_new:N \g__lngx_ipa_mono_fonts_prop
\tl_gclear_new:N \g__lngx_ipa_mono_font_features_tl

\clist_map_inline:nn {
  sans,
  mono
} {
  \clist_map_inline:nn {
    upright,
    bold~ upright,
    italic,
    bold~ italic,
    slanted,
    bold~ slanted,
    swash,
    bold~ swash,
    small~ caps
  } {
    \keys_define:nn { lngx_keys } {
      ipa~ #1~ ##1
      .code:n                = {
        \group_begin:
        \str_clear:N \l_tmpa_str
        \str_set:Ne \l_tmpa_str {
          \text_titlecase_all:n { ##1 }
          Font
        }
        \str_replace_all:Nnn \l_tmpa_str { ~ } { }
        \prop_gput:cne { g__lngx_ipa_ #1 _fonts_prop }
                       { ipa~ #1~ ##1 }
                       { ####1 }
        \group_end:
      },
      ipa~ #1~ ##1~ features
      .code:n                = {
        \group_begin:
        \str_clear:N \l_tmpa_str
        \str_set:Ne \l_tmpa_str {
          \text_titlecase_all:n { #1 }
          Features
        }
        \str_replace_all:Nnn \l_tmpa_str { ~ } { }
        \prop_gput:cne { g__lngx_ipa_ #1 _fonts_prop }
                       { ipa~ #1~ ##1~ features }
                       {
                         \str_use:N \l_tmpa_str = { ####1 }
                       }
        \group_end:
      }
```

```
    }
  }
  \keys_define:nn { lngx_keys } {
    ipa~ #1~ extra~ features
    .prop_gput:c           = {
                               g__lngx_ipa_ #1 _fonts_prop
                             }
  }
}
```

(*End of definition for* `\g__lngx_ipa_sans_fonts_prop` *and others. These functions are documented on page* 11.)

`\g__lngx_ipa_main_font_tl`
`\g__lngx_ipa_sans_font_tl`
`\g__lngx_ipa_mono_font_tl`

These keys provide keys to set fonts for IPA.

`ipa main font`
`ipa sans font`
`ipa mono font`

```
\clist_map_inline:nn {
  main,
  sans,
  mono
} {
  \keys_define:nn { lngx_keys } {
    ipa~ #1~ font
    .tl_gset:c             = { g__lngx_ipa_ #1 _font_tl }
  }
}
```

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

`ipa newcm`

This key is of type `.meta:n`. It sets certain other keys that enable the New Computer Modern fonts in book weight and in all of the serif, sans serif and monospaced families for IPA. Stylistic set 5 of NewCM is dedicated to linguistics. So we use it here. For correct diacritic placement, we need HarfBuzz renderer. That also is loaded here.

```
\keys_define:nn { lngx_keys } {
  ipa~ newcm
  .meta:n                = {
    ipa~ extra~
    features               = {
      Renderer               = {HarfBuzz},
      StylisticSet           = {05}
    },
    ipa~ sans~ extra~
    features               = {
      Renderer               = {HarfBuzz},
      StylisticSet           = {05}
    },
    ipa~ mono~ extra~
    features               = {
      Renderer               = {HarfBuzz},
      StylisticSet           = {05}
    },
    ipa~ main~ font        = { NewCM10-Book.otf },
    ipa~ sans~ font        = { NewCMSans10-Book.otf },
    ipa~ mono~ font        = { NewCMMono10-Book.otf }
```

```
  }
}
```

(*End of definition for* `ipa newcm`. *This function is documented on page* 10.)

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

```
\keys_define:nn { lngx_keys } {
  ipa~ newcm~ sans
  .meta:n                = {
    ipa~ extra~
    features               = {
      Renderer               = {HarfBuzz},
      StylisticSet           = {05}
    },
    ipa~ sans~ extra~
    features               = {
      Renderer               = {HarfBuzz},
      StylisticSet           = {05}
    },
    ipa~ mono~ extra~
    features               = {
      Renderer               = {HarfBuzz},
      StylisticSet           = {05}
    },
    ipa~ main~ font        = { NewCMSans10-Book.otf },
    ipa~ sans~ font        = { NewCMSans10-Book.otf },
    ipa~ mono~ font        = { NewCMMono10-Book.otf }
  }
}
```

(*End of definition for* `ipa newcm sans`. *This function is documented on page* 10.)

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

```
\keys_define:nn { lngx_keys } {
  ipa~ newcm~ mono
  .meta:n                = {
    ipa~ extra~
    features               = {
      Renderer               = {HarfBuzz},
      StylisticSet           = {05}
    },
    ipa~ sans~ extra~
    features               = {
      Renderer               = {HarfBuzz},
      StylisticSet           = {05}
    },
    ipa~ mono~ extra~
    features               = {
      Renderer               = {HarfBuzz},
      StylisticSet           = {05}
    },
    ipa~ main~ font        = { NewCMMono10-Book.otf },
    ipa~ sans~ font        = { NewCMSans10-Book.otf },
```

```
    ipa~ mono~ font        = { NewCMMono10-Book.otf }
  }
}
```

(*End of definition for* ***ipa newcm mono****. This function is documented on page* 10.)

`ipa 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 } {
  ipa~ newcm~ regular
  .meta:n               = {
    ipa~ extra~
    features              = {
      Renderer              = {HarfBuzz},
      StylisticSet          = {05}
    },
    ipa~ sans~ extra~
    features              = {
      Renderer              = {HarfBuzz},
      StylisticSet          = {05}
    },
    ipa~ mono~ extra~
    features              = {
      Renderer              = {HarfBuzz},
      StylisticSet          = {05}
    },
    ipa~ main~ font       = { NewCM10-Regular.otf },
    ipa~ sans~ font       = { NewCMSans10-Regular.otf },
    ipa~ mono~ font       = { NewCMMono10-Regular.otf }
  }
}
```

(*End of definition for* ***ipa newcm regular****. This function is documented on page* 10.)

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

```
\keys_define:nn { lngx_keys } {
  ipa~ newcm~ regular~ sans
  .meta:n               = {
    ipa~ extra~
    features              = {
      Renderer              = {HarfBuzz},
      StylisticSet          = {05}
    },
    ipa~ sans~ extra~
    features              = {
      Renderer              = {HarfBuzz},
      StylisticSet          = {05}
    },
    ipa~ mono~ extra~
    features              = {
      Renderer              = {HarfBuzz},
```

```
      StylisticSet           = {05}
    },
    ipa~ main~ font        = { NewCMSans10-Regular.otf },
    ipa~ sans~ font        = { NewCMSans10-Regular.otf },
    ipa~ mono~ font        = { NewCMMono10-Regular.otf }
  }
}
```

(*End of definition for* `ipa newcm regular sans`. *This function is documented on page* 10.)

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

```
\keys_define:nn { lngx_keys } {
  ipa~ newcm~ regular~ mono
  .meta:n                = {
    ipa~ extra~
    features               = {
      Renderer               = {HarfBuzz},
      StylisticSet           = {05}
    },
    ipa~ sans~ extra~
    features               = {
      Renderer               = {HarfBuzz},
      StylisticSet           = {05}
    },
    ipa~ mono~ extra~
    features               = {
      Renderer               = {HarfBuzz},
      StylisticSet           = {05}
    },
    ipa~ main~ font        = { NewCMMono10-Regular.otf },
    ipa~ sans~ font        = { NewCMSans10-Regular.otf },
    ipa~ mono~ font        = { NewCMMono10-Regular.otf }
  }
}
```

(*End of definition for* `ipa newcm regular mono`. *This function is documented on page* 10.)

We set the `ipa newcm` key by default.

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

`\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 { VV }
\cs_generate_variant:Nn \lngx_set_sans_ipa_font:nn { VV }
\cs_generate_variant:Nn \lngx_set_mono_ipa_font:nn { VV }
```

(*End of definition for* `\lngx_set_main_ipa_font:nn` *and others. These functions are documented on page* 19.)

`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* 19.)

`\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_gset_eq:NN \lngxipa \lngx_ipa:
```

(*End of definition for* `\lngxipa` *and* `\lngx_ipa:`. *These functions are documented on page* 10.)

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}`.

```
\clist_map_inline:nn {
  main,
  sans,
  mono
} {
  \cs_new_protected:cpn {
```

```
    lngx_build_ #1 _ipa_font_features:
  } {
    \prop_map_inline:cn { g__lngx_ipa_ #1 _fonts_prop } {
      \tl_gput_right:cn {
                          g__lngx_ipa_ #1 _font_features_tl
                        } { ####2 }
    }
  }
}

\hook_gput_code:nnn { begindocument / before } { . } {
  \lngx_build_main_ipa_font_features:
  \lngx_set_main_ipa_font:VV
    \g__lngx_ipa_main_font_features_tl
    \g__lngx_ipa_main_font_tl
  \lngx_build_sans_ipa_font_features:
  \lngx_set_sans_ipa_font:VV
    \g__lngx_ipa_sans_font_features_tl
    \g__lngx_ipa_sans_font_tl
  \lngx_build_mono_ipa_font_features:
  \lngx_set_mono_ipa_font:VV
    \g__lngx_ipa_mono_font_features_tl
    \g__lngx_ipa_mono_font_tl
}
⟨/ipa⟩
```

```
⟨*lang⟩
\ProvidesExplPackage{linguistix-languages}
                     {2026-01-19}
                     {v0.7}
                     {%
                       An assistant package for automatically
                       loading fonts and more settings for
                       languages.%
                     }
```

LinguisTiX-base is loaded (if not already done) for the key-value parser.

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

The babel package is loaded with `provide*=*` option as it mandates the use of modern mechanism.

```
\IfPackageLoadedF { babel } {
  \RequirePackage [ provide * = * ] { babel }
}
```

`\g_lngx_main_language_tl` I declare a `tl` that I will use for storing the main language. It is publicly available.

```
\tl_new:N \g_lngx_main_language_tl
```

(*End of definition for* `\g_lngx_main_language_tl`. *This function is documented on page* 19.)

`\g_lngx_languages_clist` I declare a `clist` that I will use for storing languages. It is publicly available.

```
\clist_new:N \g_lngx_languages_clist
```

(*End of definition for* `\g_lngx_languages_clist`. *This function is documented on page* 19.)

`\lngx_languages:nn` `\providelanguage` I develop a wrapper macro with a `:VV` variant.

```
\cs_new_protected:Npn \lngx_languages:nn #1#2 {
  \babelprovide [ #1 ] { #2 }
}

\cs_generate_variant:Nn \lngx_languages:nn { VV }
\cs_gset_eq:NN \providelanguage \lngx_languages:nn
```

(*End of definition for* `\lngx_languages:nn` *and* `\providelanguage`. *These functions are documented on page* 19.)

The babel package produces an 'info' message if the fonts are not set with `\babelfont`. Mostly they aren't set with this mechanism, so this warning is inevitable in default situations. Imagine loading LinguisTiX-fonts first and then loading this package. The fonts are already set with `\setmainfont` and friends. Thus we will be prompted with this warning always. In order to avoid that, I renew the wrapper functions around `\setmainfont` to `\babelfont`. Note that this only affects the usage when LinguisTiX-fonts is loaded. If you use LinguisTiX-Languages and then use `\setmainfont`-like commands, you will get `babel`'s warning and I have no intention to suppress that behaviour.

```
\IfPackageLoadedTF { linguistix-fonts } {
  \cs_gset_protected:Npn \lngx_set_main_font:nn #1#2 {
    \babelfont { rm } [ #1 ] { #2 }
  }
  \cs_gset_protected:Npn \lngx_set_sans_font:nn #1#2 {
    \babelfont { sf } [ #1 ] { #2 }
  }
  \cs_gset_protected:Npn \lngx_set_mono_font:nn #1#2 {
    \babelfont { tt } [ #1 ] { #2 }
  }
} {
  \cs_new_protected:Npn \lngx_set_main_font:nn #1#2 {
    \babelfont { rm } [ #1 ] { #2 }
  }
  \cs_new_protected:Npn \lngx_set_sans_font:nn #1#2 {
    \babelfont { sf } [ #1 ] { #2 }
  }
  \cs_new_protected:Npn \lngx_set_mono_font:nn #1#2 {
    \babelfont { tt } [ #1 ] { #2 }
  }
}
```

\lngx_other_main_font:nnn
\lngx_other_sans_font:nnn
\lngx_other_mono_font:nnn

The following macros set fonts for other languages using the \babelfont command.

```
\cs_gset_protected:Npn \lngx_other_main_font:nnn #1#2#3 {
  \babelfont [ #1 ] { rm } [ #2 ] { #3 }
}

\cs_gset_protected:Npn \lngx_other_sans_font:nnn #1#2#3 {
  \babelfont [ #1 ] { sf } [ #2 ] { #3 }
}

\cs_gset_protected:Npn \lngx_other_mono_font:nnn #1#2#3 {
  \babelfont [ #1 ] { tt } [ #2 ] { #3 }
}

\cs_generate_variant:Nn \lngx_other_main_font:nnn { nee }
\cs_generate_variant:Nn \lngx_other_sans_font:nnn { nee }
\cs_generate_variant:Nn \lngx_other_mono_font:nnn { nee }
```

(*End of definition for* \lngx_other_main_font:nnn, \lngx_other_sans_font:nnn, *and* \lngx_other_mono_font:nnn. *These functions are documented on page* 18.)

\lngx_load_languages:n
\loadlanguages

I provide a simple macro that only does the job of loading languages, both in LaTeX3 style, as well as the in the plain style.

```
\cs_new_protected:Npn \lngx_load_languages:n #1 {
  \lngx_set_keys:n { languages = { #1 } }
}

\cs_gset_eq:NN \loadlanguages \lngx_load_languages:n
```

(*End of definition for* \lngx_load_languages:n *and* \loadlanguages. *These functions are documented on page* 19.)

I equate the `\arabic` command to a new command I want to provide. This is done in order to get control over the default LaTeX counters. The command is manipulated when plugs are activated.

`\lngx_counter:n`

```
\cs_gset_eq:NN \lngx_counter:n \arabic
```

(*End of definition for* `\lngx_counter:n`. *This function is documented on page* 19.)

Now all the default counters are changed from `\arabic` to `\lngx_counter:n`.

```
\cs_set:Npn \thechapter {
  \lngx_counter:n { chapter }
}
\cs_set:Npn \thesection {
  \lngx_counter:n { section }
}
\cs_set:Npn \thesubsection {
  \lngx_counter:n { subsection }
}
\cs_set:Npn \thesubsubsection {
  \lngx_counter:n { subsubsection }
}
\cs_set:Npn \theparagraph {
  \lngx_counter:n { section }
}
\cs_set:Npn \thesubparagraph {
  \lngx_counter:n { section }
}
\cs_set:Npn \thepage {
  \lngx_counter:n { page }
}
\cs_set:Npn \thefigure {
  \lngx_counter:n { figure }
}
\cs_set:Npn \thetable {
  \lngx_counter:n { table }
}
\cs_set:Npn \thefootnote {
  \lngx_counter:n { footnote }
}
\cs_set:Npn \thempfootnote {
  \lngx_counter:n { mpfootnote }
}
\cs_set:Npn \theequation {
  \lngx_counter:n { equation }
}
```

Here, I define the socket `lngx/native-numbering`.

```
\socket_new:nn { lngx / native-numbering } { 0 }
```

`strict` This plug sets the numbering strictly to the main language. If used, the function `\lngx_counter:n` is changed to the respective `\xxxxcounter` command (where `xxxx` stands for the main language of the document).

```
\socket_new_plug:nnn { lngx / native-numbering }
                     { strict } {
  \cs_gset_eq:Nc \lngx_counter:n {
    \tl_use:N \g_lngx_main_language_tl counter
  }
}
```

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

`logical` Here, I define the `logical` plug for `lngx/native-numbering`. The mechanism is pretty similar as the one used for `strict`, but here I don't renew it to the main language counter, but instead I use the `\localecounter` command provided by the babel package. The counters are then printed contextually (and TEX-logically).

```
\socket_new_plug:nnn { lngx / native-numbering }
                     { logical } {
  \cs_gset_protected:Npn \lngx_counter:n ##1 {
    \localecounter { digits } { ##1 }
  }
}
```

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

`off` If the `off` plug is selected, then native digits are not needed. Thus the `\lngx_counter:n` is set to the unmodified `\arabic` again.

```
\socket_new_plug:nnn { lngx / native-numbering} { off } {
  \cs_gset_eq:NN \lngx_counter:n \arabic
}
```

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

`native numbering` The three choices for the `native numbering` key, i.e., `strict`, `logical` and `off` are defined here. All of them activate the plugs of their name with the `lngx/native-numbering` socket.

```
\cs_generate_variant:Nn \socket_assign_plug:nn { ne }

\keys_define:nn { lngx_keys } {
  native~ numbering
  .choices:nn            = { strict,logical,off } {
    \socket_assign_plug:ne { lngx / native-numbering } {
      \str_use:N \l_keys_choice_str
    }
    \socket_use:n { lngx / native-numbering }
  },
```

Similarly, we set the default value to on.

```
  native~ numbering
  .default:n             = { strict }
}
```

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

\lngx_misc_reset: Despite having sufficient control with the two plugs, there are some additional settings required by some languages that are often not needed by most others. E.g., Marathi renews the way enumerated lists are printed and that is supposed to be renewed when the language is changed. I provide a shorthand to be used for resetting such settings. It can be used in the packages of languages that don't need special settings.

```
\cs_new_protected:Npn \lngx_misc_reset: {
  \cs_set:Npn \theenumii { \alph { enumii } }
  \cs_set:Npn \labelenumii { ( \theenumii ) }
  \cs_set:Npn \theenumiii { \roman { enumiii } }
  \cs_set:Npn \labelenumiii { \theenumiii . }
  \cs_set:Npn \theenumiv { \Alph { enumiv } }
  \cs_set:Npn \labelenumiv { \theenumiv . }
  \IfPackageLoadedT { expex } {
    \lingset { alpha }
  }
  \cs_gset_eq:NN \emph \textit
}
```

(*End of definition for* `\lngx_misc_reset:`*. This function is documented on page* 19.)

Here, I write a message to be issued when user loads an unsupported language.

```
\msg_new:nnn { linguistix-languages } { no_support } {
  '#1'~ is~ not~ supported.\\
  If~ you~ want~ it~ to~ be~ supported,~ please~ report~
  to~ the~ maintainers.
}
```

languages I use the `.code:n` type for developing the `languages` key.

```
\keys_define:nn { lngx_keys } {
  languages
  .code:n                = {
```

I pass the argument of this key to a global `clist`. It is stored for public use.

```
    \clist_gset:Nn \g_lngx_languages_clist { #1 }
```

Since this is a public `clist` for accessing the names of the languages, I copy it to a temporary one so that the items of public interest are not lost during the operations.

```
    \clist_set_eq:NN \l_tmpa_clist \g_lngx_languages_clist
```

I check if the `clist` is empty or not. If it is empty, that means the user used the key without a value. In that case, babel already loads an 'info'-message saying that no language is loaded. So we ignore the branch and silently move to the false branch.

```
    \clist_if_empty:NF \l_tmpa_clist {
```

In the false branch, I pop out the first element from the clist to `\l_tmpa_tl`. This is the first language passed by the user. In LinguisTiX-Languages, I assume that it is intended to be the first language. It is important to pop the element out because the settings used for the main language are different than the ones used for other languages.

```
      \clist_pop:NN \l_tmpa_clist \l_tmpa_tl
```

Since this `tl` stores the language that is going to be the main one, I equate it to another public `tl` that I will be using later in language files.

```
      \tl_set_eq:NN \g_lngx_main_language_tl \l_tmpa_tl
```

In `\l_tmpb_tl`, I save the options that need to go with the language stored in `\l_tmpa_tl`. The package used to have `onchar` option loaded conditionally with LuaLaTeX, but to avoid potential clashes, now it has moved to the individual package files of languages. Now I directly load the `main` option which makes the concerned language the 'main' language of the document.

```
      \tl_set:Ne \l_tmpb_tl {
        main,
```

To load the data from `ini` files, I use the `import` parameter.

```
        import
      }
```

I use the `\babelprovide` wrapper we saw earlier with the values of the first language.

```
      \lngx_languages:VV \l_tmpb_tl \l_tmpa_tl
```

I scan if the package for this language is available. If it is, it is loaded.

```
      \file_if_exist:nTF { linguistix - \l_tmpa_tl . sty } {
        \exp_args:Ne \RequirePackage
                     { linguistix - \l_tmpa_tl }
      } {
```

If it is not, I issue the `no_ldf` warning message. It takes one argument that is the name of the language. It is extracted using the `V` argument type.

```
        \msg_warning:nnV { linguistix-languages }
                         { no_support }
                         \l_tmpa_tl
      }
```

The temporary `tl`s are cleared.

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

I again check if the `clist` is empty. If it is, it means the user is typesetting a monolingual document as they don't need any other language than the 'main' one.

```
      \clist_if_empty:NF \l_tmpa_clist {
```

Now I have to repeat the same actions for all the pending languages. I do it with `\clist_map_inline:Nn`.

```
        \clist_map_inline:Nn \l_tmpa_clist {
          \clist_pop:NN \l_tmpa_clist \l_tmpa_tl
          \tl_set:Ne \l_tmpb_tl { import }
          \lngx_languages:VV \l_tmpb_tl \l_tmpa_tl
          \file_if_exist:nTF {
            linguistix - \l_tmpa_tl . sty
          } {
            \exp_args:Ne \RequirePackage
                         { linguistix - \l_tmpa_tl }
          } {
            \msg_warning:nnV { linguistix-languages }
                             { no_ldf }
                             \l_tmpa_tl
          }
          \tl_clear:N \l_tmpa_tl
          \tl_clear:N \l_tmpb_tl
        }
      }
```

```
      }
    }
}
⟨/lang⟩
```

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

```
⟨*logos⟩
\ProvidesExplPackage{linguistix-logos}
                     {2026-01-19}
                     {v0.7}
                     {%
                       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: [
  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}
      },
    }
  }
]{ NewCMUncial10-Book.otf }
```

(*End of definition for* `\lngx_logo_font:`. *This function is documented on page* 20.)

lngx_purple_color The following defines the `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* 20.)

\lngxlogo Here, I define the commands for printing various logos.

```
\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* 14.)

Since we need expandable commands, I use the non-protected function, `\cs_new:Npn` for defining them.

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

Here, I define all the logos with a `clist`. The package names are stored in the `clist` and then used at appropriate positions.

```
\clist_map_inline:nn {
  base,examples,fixpex,fonts,ipa,languages,logos,nfss,
  marathi,british,american,english,greek,malayalam,glossing,
  leipzig
} {
```

`#1` is substituted with the package name. First, for the command-name itself, then as the optional argument of `\lngxlogo` and then in the PDF-string.

```
  \cs_new:cpn { lngx #1 logo } {
    \texorpdfstring {
      \lngxlogo [ #1 ]
    } {
      LinguisTiX - #1
    }
  }
}
```

⟨/logos⟩

⟨*nfss⟩

```
\ProvidesExplPackage{linguistix-nfss}
                     {2026-01-19}
                     {v0.7}
                     {%
                       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* 20.)

\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
  }
```

When the package was first released, there was no public interface for guessing the current meta family, but from `ltnews42`, `\@currentmetafamily` became available. Thanks Frank for pointing this out.

```
  \tl_set:Ne \l_lngx_current_meta_family_tl {
    \@currentmetafamily % new from ltnews42, thanks Frank!
  }
```

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

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

(*End of definition for* `\l_lngx_current_encoding_tl` *and others. These functions are documented on page 20.*)

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

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

After the processing of `\selectfont`, I equate the temporary `tl` with the one that the package is tracking. This way, the effect of `\selectfont` 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 `\selectfont`.

```
\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 }
}
```

`\lngx_if_encoding_p:n`
`\lngx_if_encoding:nTF`

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* 20.)

`\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* 16.)

`\lngx_if_meta_family_p:n` `\lngx_if_meta_family:nTF`

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* 20.)

`\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* 16.)

`\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* 20.)

`\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* 16.)

`\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* 20.)

`\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* 16.)

`\lngx_if_shape_p:n` `\lngx_if_shape:nTF` 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* 20.)

`\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* 16.)

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* 20.)

`\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* 21.)

`\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* 21.)

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_gset_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* 21.)

\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_gset_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 21.*)

`\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_gset_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 21.*)

`\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_gset_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* 21.)

`\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_gset_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* 21.)

`\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_gset_eq:NN \softernormalfont \lngx_softer_normal_font:
```

(*End of definition for* `\lngx_softer_normal_font:` *and* `\softernormalfont`*. These functions are documented on page 21.*)

`\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_gset_eq:NN \softestnormalfont \lngx_softest_normal_font:
```

(*End of definition for* `\lngx_softest_normal_font:` *and* `\softestnormalfont`*. These functions are documented on page 21.*)

`\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 16.*)

References

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

Munn, Alan and Enrico Gregorio (5th Dec. 2023). *ExPex fails with unicode-math. How to avoid the clash?* URL: https://tex.stackexchange.com/q/703094 (visited on 21/12/2025).