

The LINGUIS \mathcal{T} \mathbf{I} \mathbf{X} bundle

निरंजन

27 April 2026 (v0.9b)

🏠 <https://ctan.org/pkg/linguistix>
💎 <https://puszcza.gnu.org.ua/projects/linguistix>
🔗 <https://matrix.to/#/#linguistix:matrix.org>

Abstract

There are quite a few L^AT_EX packages that support typesetting in linguistics, but most of them lack a modern L^AT_EX-like users syntax as well as a programming interface. The LINGUIS \mathcal{T} \mathbf{I} \mathbf{X} bundle fills this gap. It contains several packages enhancing the general support for linguistics in L^AT_EX. 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	8	LINGUIS \mathcal{T} \mathbf{I} \mathbf{X} -eLOSSING	8
				Interface... 20; Implementation... 39	
2	Planned	4	9	LINGUIS \mathcal{T} \mathbf{I} \mathbf{X} -ipa	11
				Interface... 21; Implementation... 55	
3	Funding	4	10	LINGUIS \mathcal{T} \mathbf{I} \mathbf{X} -LANGUAGES	14
				Interface... 21; Implementation... 66	
4	Acknowledgements	4	11	LINGUIS \mathcal{T} \mathbf{I} \mathbf{X} -LOGOS	16
				Interface... 22; Implementation... 73	
5	LINGUIS \mathcal{T} \mathbf{I} \mathbf{X} -BASE	5	12	LINGUIS \mathcal{T} \mathbf{I} \mathbf{X} -NFSS	17
	Interface... 19; Implementation... 25			Interface... 22; Implementation... 74	
6	LINGUIS \mathcal{T} \mathbf{I} \mathbf{X} -fixpex	5			
	Interface... 20; Implementation... 26				
7	LINGUIS \mathcal{T} \mathbf{I} \mathbf{X} -FONTS	5			
	Interface... 20; Implementation... 28				
				Index	87

The LINGUIS \mathcal{T} \mathbf{I} \mathbf{X} bundle

Copyright © 2025, 2026 निरंजन

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but *without any warranty*; without even the implied warranty of *merchantability* or *fitness for a particular purpose*. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

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

I 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.

I 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 \LaTeX . Visually, it matches the default Computer Modern design of \LaTeX , 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 '[aa]'. 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 Tzolomitis, 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 `LINGUIS\X-ipa`) renders '[aa]'. The package enables New Computer Modern family with stylistic set `o5` 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 '[gg]'. 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 [gg] and not [g̱g̱].

In order to avail all of these features, I have set New Computer Modern as the default font-family of `LINGUIS\X`. 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 L^AT_EX-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. LINGUISCIX 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 T_EX users group's T_EX 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 LINGUISCIX-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 T_EX 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 L^AT_EX₃'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 L^AT_EX. 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 T_EX users group decided to financially support the development of a planned package in the bundle. I am grateful to them for their support.

Throughout the development of LINGUISCIX-eLOSSING, Shireen Irani helped me with her valuable comments regarding the accessibility needs in the field of linguistics. I thank her for her constant 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 `LINGUIS \mathcal{T} IX` 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 LINGUIS \mathcal{T} IX-BASE

[L^AT_EX 3-interface](#) | [Implementation](#)

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

`\linguistix` $\{ \langle key-value-list \rangle \}$

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 $\langle key-value-list \rangle$. So you can load any package of `LINGUIS \mathcal{T} IX` and use the `\linguistix` command. The only exception to this is `LINGUIS \mathcal{T} IX-NFSS`. We will see how it is different in its section.

6 LINGUIS \mathcal{T} IX-FIXPEX

[L^AT_EX 3-interface](#) | [Implementation](#)

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

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

7 LINGUIS \mathcal{T} IX-FONTS

[L^AT_EX 3-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 `LINGUIS \mathcal{T} IX-IPA` separately.

Antonis suggested a typographic enhancement for the logo of L^AT_EX. 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` L^AT_EX
`\ogLaTeX` L^AT_EX

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

I 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` package 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 `LINGUISCIX-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.

<code>old style numbers</code>	<code>= {\(truth value)}</code>	<code>true false</code>
<code>old style one</code>	<code>= {\(truth value)}</code>	<code>true false</code>

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
0I23456789	Old style with the old 1
0123456789	Lining

<code>newcm</code>	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 <code>regular</code> 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.
<code>newcm sans</code>	
<code>newcm mono</code>	
<code>newcm regular</code>	
<code>newcm regular sans</code>	
<code>newcm regular mono</code>	

2 Maths

`LINGUISCIX-FONTS` sets maths fonts also. I have used `lua-unicode-math` package which is faster and which is said to be the future of maths in \LaTeX . But, as of now it is highly experimental. If you want to stick to the stable `unicode-math` package. The trick is simply to load the same before loading `LINGUISCIX`. That will suppress the loading of

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

lua-unicode-math. In order to control the settings related to maths, the following keys can be used.

<code>math</code>	<code>= {\langle math font \rangle}</code>
<code>math features</code>	<code>= {\langle math font features \rangle}</code>
<code>math bold</code>	<code>= {\langle bold math font \rangle}</code>
<code>math bold features</code>	<code>= {\langle bold math font features \rangle}</code>

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.

<code>bourbaki's empty set</code>	<code>= {\langle truth value \rangle}</code>	<code>true false</code>
-----------------------------------	--	---------------------------

In (L^A)T_EX, the default shape of the ‘empty set’ symbol is: ‘ \emptyset ’, but the symbol used by the Bourbaki group is still considered more correct and preferred by many (including me). New Computer Modern Math fonts provide it 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: ‘ \emptyset ’ and not: ‘ \emptyset ’. In order to change this behaviour, one may use this key and set it to false for getting the slashed-zero of original (L^A)T_EX. Hail plumbers union, *IYKYK!* ;-)

This package provides a suit for creating interlinear glosses. It is supported by T_EX 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 LINGUIS $\overline{\text{T}}$ X-Leipzig to support a set of glosses. The glosses created by the package use the new code of the L^AT_EX 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.

\backslash glx	$\{(comma\ separated\ list\ of\ glosses)\}$
\backslash glx*	$\{(comma\ separated\ list\ of\ glosses)\}$

These simple commands take a comma separated list as their argument. All the items from the list are glosses (either created by the user or provided by a package). Cases of the items given in the list are ignored. Spaces around the items are ignored. The regular unstarred command prints the glosses related to 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}

\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}

```

The expansions of PST and PRS (from LINGUIS $\overline{\text{T}}$ X-Leipzig package) are past and present respectively. 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.

\backslash newgloss	$\{(gloss)\} \{(expansion)\}$
\backslash renewgloss	$\{(gloss)\} \{(expansion)\}$

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

\backslash 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. E.g.:

Glossary

PRS: present	8		PST: past	8
--------------------	---	--	-----------------	---

`\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.

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 | expansion`

The `format` key is used for setting the format of either the gloss or the expansion. It's a meta key that takes other key-val pair in the argument. The nested keys control the formatting of the respective elements.

`gloss` = *{(formatting commands for glosses)}* `\textsc{#1}`
`expansion` = *{(formatting commands for glosses)}*

These keys only work inside the meta key `format`. They set the commands that print either the gloss or the expansion. `#1` refers to the printed text of them. No special formatting is applied to expansions by default, but glosses are by default printed in `\textsc`.

`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.

<u>columns</u>	= $\{ \langle \textit{number of columns} \rangle \}$	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.	
<u>page numbers</u>	= $\{ \langle \textit{truth value} \rangle \}$	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.	
<u>sectioning</u>	= $\{ \langle \textit{section level} \rangle \}$	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.	
<u>section number</u>	= $\{ \langle \textit{truth value} \rangle \}$	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.	
<u>no bold</u>	= $\{ \langle \textit{truth value} \rangle \}$	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.	
<u>separator</u>	= $\{ \langle \textit{separator between glosses or expansions} \rangle \}$	
	This is a context-sensitive key. If used with <code>\glx</code> , then it sets the separator between the glosses (<code>,_</code> is the default). If used with <code>\glx*</code> , it sets the separator between the expansions (<code>,_</code> is the default) and if used with the <code>\listofglosses</code> , it sets the separator between glosses and their expansions (<code>:_</code> is the default).	
<u>entry separator</u>	= $\{ \langle \textit{separator between pairs of glosses and expansions} \rangle \}$	
	Each pair of gloss and its expansion is separated using a token list controlled by this key. The default is <code>\par</code> .	

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, LINGUIS $\overline{\text{C}}$ I $\overline{\text{X}}$ -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 phi: eɪ}` \longrightarrow [a_ɪ p^hi: e_ɪ] whereas the starred version prints it in slashes for phonemic transcription, e.g.: `\ipatext*{aɪ phi: eɪ}` \longrightarrow /a_ɪ p^hi: e_ɪ/.

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 [a_ɪ p^hi: e_ɪ] and /a_ɪ p^hi: e_ɪ/ respectively:

```
{\lngxipa [aɪ phi: eɪ]}
{\lngxipa /aɪ phi: 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 LINGUIS $\overline{\text{C}}$ I $\overline{\text{X}}$ -FONTS and LINGUIS $\overline{\text{C}}$ I $\overline{\text{X}}$ -ipa.

Family	LINGUIS $\overline{\text{C}}$ I $\overline{\text{X}}$ -FONTS	LINGUIS $\overline{\text{C}}$ I $\overline{\text{X}}$ -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
<i>Continued on the next page...</i>		

Family	LINGUIS _{CL} X-FONTS	LINGUIS _{CL} X-IPA
	text italic	ipa italic
	text italic features	ipa italic features
	text bold italic	ipa bold italic
	text bold italic features	ipa bold italic features
	text slanted	ipa slanted
	text slanted features	ipa slanted features
	text bold slanted	ipa bold slanted
	text bold slanted features	ipa bold slanted features
	text swash	ipa swash
	text swash features	ipa swash features
	text bold swash	ipa bold swash
	text bold swash features	ipa bold swash features
	text small caps	ipa small caps
	text small caps features	ipa small caps features
Sans serif	text sans 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
<i>Continued on the next page...</i>		

Family	LINGUISṬṬX-FONTS	LINGUISṬṬX-IPA
	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
<i>End of the table...</i>		

Table 1: Font keys provided by LINGUISṬṬX-FONTS and LINGUISṬṬX-IPA

Apart from these, both the packages provide the following keys for appending to the extra features for the respective fonts:

- text main extra features
- text sans extra features
- text mono extra features
- ipa main extra features
- ipa sans extra features
- ipa mono extra features

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, Lua^AT_EX, 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 L^AT_EX-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: L^AT_EX 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 `\items` as I, I.I, I.I.I and I.I.I.I. 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.

<hr/> <code>languages</code> <hr/>	<code>{\langle list of languages \rangle}</code>
<code>\loadlanguages</code>	<code>{\langle list of languages \rangle}</code>
	This key works with the central key-parser of <code>LINGUISTIX</code> , i.e., <code>\linguistix</code> . It accepts one argument that is a list of languages user wants to load. Unlike <code>babel</code> , the first element of this list is set as the main language for the document. The command <code>\loadlanguages</code> has the identical behaviour. In fact, it is a wrapper around the key.
<hr/> <code>\providelanguage</code> <hr/>	<code>{\langle language options \rangle} {\langle language name \rangle}</code>
	This is a wrapper command over <code>\babelprovide</code> . The first argument is passed to the optional argument of <code>\babelprovide</code> and the second one to the mandatory argument of the same. For more information, please read <code>babel</code> 's manual.
	Languages supported by <code>LINGUISTIX-LANGUAGES</code> are loaded with a package with that language's name. If it is absent, the package produces a warning.
<hr/> <code>native numbering</code> <hr/>	<code>= {\langle strict/logical/off \rangle}</code>
	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.
<hr/> <code>strict</code> <hr/>	The 'strict' plug changes the <code>\lngx_counter:n</code> command to the counter of the main language of the document. That way all the counters are printed in the main language.
<hr/> <code>logical</code> <hr/>	This plug changes the meaning of <code>\lngx_counter:n</code> to the <code>\localecounter</code> command provided by <code>babel</code> . 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 <code>TeX</code> '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 <code>TeX</code> 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 <code>TeX</code> .
<hr/> <code>off</code> <hr/>	It is equivalent of the <code>noop</code> plug when the other two are not used at all. It is only required when you want to go back to <code>L^ATeX</code> defaults. E.g., if you have turned strict native numbering in some language and you want it to go back to <code>L^ATeX</code> defaults, you may use this.

This is a small package that provides commands for printing logos of the LINGUISŢIX 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 LINGUISŢIX bundle is printed with this command, e.g., `\lngxlogo[fonts] → LINGUISŢIX-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.

This is an extension package to the existing NFSS scheme of L^AT_EX. The NFSS mainly works on the four facets of the text, i.e., encoding, family, shape and 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 LuaL^AT_EX.

```

\documentclass{article}

\begin{document}
\makeatletter
\fontencoding{OT1}\sffamily\itshape\bfseries
\selectfont
\fontencoding\ | \fontfamily\ | \fontseries\ | \fontshape\quad
\normalfont
\fontencoding\ | \fontfamily\ | \fontseries\ | \fontshape
\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 `LINGUISTX` 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. `LINGUISTX-NFSS` is useful in such cases. It introduces the concept of ‘super’ font family. It shouldn’t be confused with `LATEX 2ε`’s ‘meta’ font family. It refers to `rm`, `sf` or `tt` in the kernel. This package provides control over these facets. Let’s have a look at the macros it provides.

<hr/>	
<code>\IfEncodingTF</code>	<code>★ {\langle encoding \rangle} {\langle true code \rangle} {\langle false code \rangle}</code>
<code>\IfEncodingT</code>	<code>★ {\langle encoding \rangle} {\langle true code \rangle}</code>
<code>\IfEncodingF</code>	<code>★ {\langle encoding \rangle} {\langle false code \rangle}</code>
<code>\CurrentEncoding</code>	<code>★</code> If the current encoding matches with the given <code>\langle encoding \rangle</code> , it selects the true branch; false otherwise. The <code>\CurrentEncoding</code> macro expands to the current encoding.
<hr/>	
<code>\IfMetaFamilyTF</code>	<code>★ {\langle meta family \rangle} {\langle true code \rangle} {\langle false code \rangle}</code>
<code>\IfMetaFamilyT</code>	<code>★ {\langle meta family \rangle} {\langle true code \rangle}</code>
<code>\IfMetaFamilyF</code>	<code>★ {\langle meta family \rangle} {\langle false code \rangle}</code>
<code>\CurrentMetaFamily</code>	<code>★</code> If the current meta family matches with the given <code>\langle meta family \rangle</code> , it selects the true branch; false otherwise. The <code>\CurrentMetaFamily</code> macro expands to the current meta family.
<hr/>	
<code>\IfSuperFamilyTF</code>	<code>★ {\langle super family \rangle} {\langle true code \rangle} {\langle false code \rangle}</code>
<code>\IfSuperFamilyT</code>	<code>★ {\langle super family \rangle} {\langle true code \rangle}</code>
<code>\IfSuperFamilyF</code>	<code>★ {\langle super family \rangle} {\langle false code \rangle}</code>
<code>\CurrentSuperFamily</code>	<code>★</code> If the current super family matches with the given <code>\langle super family \rangle</code> , it selects the true branch; false otherwise. The <code>\CurrentSuperFamily</code> macro expands to the current super family.
<hr/>	
<code>\IfSeriesTF</code>	<code>★ {\langle series \rangle} {\langle true code \rangle} {\langle false code \rangle}</code>
<code>\IfSeriesT</code>	<code>★ {\langle series \rangle} {\langle true code \rangle}</code>
<code>\IfSeriesF</code>	<code>★ {\langle series \rangle} {\langle false code \rangle}</code>
<code>\CurrentSeries</code>	<code>★</code> If the current series matches with the given <code>\langle series \rangle</code> , it selects the true branch and false otherwise. The <code>\CurrentSeries</code> macro expands to the current series.
<hr/>	
<code>\IfShapeTF</code>	<code>★ {\langle shape \rangle} {\langle true code \rangle} {\langle false code \rangle}</code>
<code>\IfShapeT</code>	<code>★ {\langle shape \rangle} {\langle true code \rangle}</code>
<code>\IfShapeF</code>	<code>★ {\langle shape \rangle} {\langle false code \rangle}</code>
<code>\CurrentShape</code>	<code>★</code> If the current series matches with the given <code>\langle shape \rangle</code> , it selects the true branch and false otherwise. The <code>\CurrentShape</code> macro expands to the current shape.
<hr/>	
<code>\superfontfamily</code>	<code>\{\langle family ID \rangle\} {\langle rm=\langle rm NFSS \rangle, sf=\langle sf NFSS \rangle, tt=\langle tt NFSS \rangle\}</code>

Every super font family has a `\langle family ID \rangle`, even the default one (i.e., `default`). This command creates a super family with the given `\langle family ID \rangle`s. The `\langle meta family keys \rangle` 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=\{\langle key \rangle\}` option to it and use the `\langle key \rangle` in the suitable `\langle meta family key \rangle`. Note that using all these keys is *not* mandatory. A super family may have ≤ 3 keys.

<code>\softsuperfontfamily</code>	<code>{\langle ID \rangle}{\langle encoding, family, series, shape \rangle}</code>
<code>\softersuperfontfamily</code>	<code>{\langle ID \rangle}</code>
<code>\softtestsuperfontfamily</code>	<code>{\langle ID \rangle}</code>

These commands loads the super font family with the given $\langle ID \rangle$. 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.

<code>\softnormalfont</code>	<code>{\langle encoding, family, series, shape \rangle}</code>
<code>\softernormalfont</code>	
<code>\softestnormalfont</code>	

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? :-)

L^AT_EX₃ interface for programmers

In this section, we take a look at the public L^AT_EX₃ commands of the bundle. These can be considered stable and can be used in production code.

LINGUISTIX-BASE

[Documentation](#) | [Implementation](#)

<code>\lngx_set_keys:n</code>	$\langle keyval list \rangle$
-------------------------------	-------------------------------

This is the base command for `\linguistix`. It takes a comma separated list of $\langle keyval list \rangle$ and parses it.

LINGUIS \mathbb{T} X-FIXPEX

[Documentation](#) | [Implementation](#)

No L^AT_EX₃ function provided by this package.

LINGUIS \mathbb{T} X-FONTS

[Documentation](#) | [Implementation](#)

<code>\g_lngx_old_style_bool</code>	These are the two booleans that are used to check if the old style numbers, the old style one (i.e., ‘r’) and Bourbaki’s empty set symbol (i.e., ‘ \emptyset ’) is asked by the user.
<code>\g_lngx_old_style_one_bool</code>	
<code>\g_lngx_bourbaki_bool</code>	

<code>\lngx_set_main_font:nn</code>	<code>{\features}</code> <code>{\font}</code>
<code>\lngx_set_main_font:VV</code>	<code>{\features}</code> <code>{\font}</code>
<code>\lngx_set_sans_font:nn</code>	<code>{\features}</code> <code>{\font}</code>
<code>\lngx_set_sans_font:VV</code>	<code>{\features}</code> <code>{\font}</code>
<code>\lngx_set_mono_font:nn</code>	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 (lua)-unicode-math, i.e., <code>\setmainfont</code> , <code>\setsansfont</code> , <code>\setmonofont</code> and <code>\setmathfont</code> . The <code>\features</code> are passed to the optional argument and the <code>\font</code> is passed to the mandatory argument of the respective command from the aforementioned list.
<code>\lngx_set_mono_font:VV</code>	
<code>\lngx_set_math_font:nn</code>	
<code>\lngx_set_math_font:VV</code>	

<code>\lngx_other_main_font:nnn</code>	<code>{\language}</code> <code>{\features}</code> <code>{\font}</code>
<code>\lngx_other_main_font:nee</code>	<code>{\language}</code> <code>{\features}</code> <code>{\font}</code>
<code>\lngx_other_sans_font:nnn</code>	<code>{\language}</code> <code>{\features}</code> <code>{\font}</code>
<code>\lngx_other_sans_font:nee</code>	These commands take three arguments. These are wrapper commands around the font-setting commands of babel. The <code>\features</code> are passed to the optional argument and the <code>\font</code> is passed to the mandatory argument of the respective command from the aforementioned list.
<code>\lngx_other_mono_font:nnn</code>	
<code>\lngx_other_mono_font:nee</code>	

LINGUIS \mathbb{T} X-GLOSSING

[Documentation](#) | [Implementation](#)

<code>\lngx_gloss_format:n</code>	<code>{\gloss}</code>
<code>\lngx_expansion_format:n</code>	<code>{\expansion}</code>

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.

<code>\lngx_gloss_new:nn</code>	<code>{\gloss}</code> <code>{\expansion}</code>
---------------------------------	---

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

<code>\lngx_gloss_list:</code>	This functions prints the list of glosses and is equated with <code>\listofglosses</code> .
--------------------------------	---

<code>lngx_multicols</code>	<code>{\section title}</code>
-----------------------------	-------------------------------

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.

LINGUISṪIX-ipa

[Documentation](#) | [Implementation](#)

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

<code>\lngx_set_main_ipa_font:nn</code>	<code>{⟨features⟩} {⟨font⟩}</code>
<code>\lngx_set_main_ipa_font:VV</code>	
<code>\lngx_main_ipa:</code>	These functions set the IPA fonts for the serif variants. The <code>⟨font⟩</code> is set with <code>⟨features⟩</code>
<code>lngx_ipa_rm_nfss</code>	for the serif IPA. The command to switch to this family is <code>\lngx_main_ipa:</code> . It can be

accessed with the NFSS family `lngx_ipa_rm_nfss`.

<code>\lngx_set_sans_ipa_font:nn</code>	<code>{⟨features⟩} {⟨font⟩}</code>
<code>\lngx_set_sans_ipa_font:VV</code>	
<code>\lngx_sans_ipa:</code>	These functions set the IPA fonts for the sans variants. The <code>⟨font⟩</code> is set with <code>⟨features⟩</code>
<code>lngx_ipa_sf_nfss</code>	for the sans IPA. The command to switch to this family is <code>\lngx_sans_ipa:</code> . It can be

accessed with the NFSS family `lngx_ipa_sf_nfss`.

<code>\lngx_set_mono_ipa_font:nn</code>	<code>{⟨features⟩} {⟨font⟩}</code>
<code>\lngx_set_mono_ipa_font:VV</code>	
<code>\lngx_mono_ipa:</code>	These functions set the IPA fonts for the mono variants. The <code>⟨font⟩</code> is set with <code>⟨features⟩</code>
<code>lngx_ipa_tt_nfss</code>	for the mono IPA. The command to switch to this family is <code>\lngx_mono_ipa:</code> . It can be

accessed with the NFSS family `lngx_ipa_nfss_nfss`.

<code>\lngx_ipa:</code>	The <code>\lngx_ipa:</code> command loads the super family <code>lngx_ipa</code> (see the documentation of
<code>lngx_ipa</code>	LINGUISṪIX-NFSS). The <code>\lngx_ipa:</code> function has a user-side command <code>\lngxipa</code> too.

LINGUISṪIX-LANGUAGES

[Documentation](#) | [Implementation](#)

Here are the L3 functions defined for LINGUISṪIX-LANGUAGES.

<code>\g_lngx_main_language_tl</code>	A <code>tl</code> that globally stores the main language of the document.
---------------------------------------	---

<code>\g_lngx_languages_clist</code>	A <code>clist</code> that globally stores the languages that are used.
--------------------------------------	--

<code>\lngx_languages:nn</code>	<code>{⟨language options⟩} {⟨language name⟩}</code>
<code>\lngx_languages:VV</code>	<code>⟨language options tl⟩ ⟨language tl⟩</code>

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

<code>\lngx_load_languages:n</code>	<code>{⟨list of languages⟩}</code>
-------------------------------------	------------------------------------

This function loads the languages in LINGUISṪIX sense.

<code>\lngx_counter:n</code>	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.

<code>\lngx_misc_reset:</code>	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.

There are only two L^AT_EX₃ 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 `!3color` module. It can be accessed using this variable. The color looks like: .

This subsection discusses the programming interface LINGUIS \mathcal{T} **X**-NFSS provides.

`\c_lngx_default_rmdefault_tl` * These `tl`s expand to the default values of the fonts set at the `\begin{document}/end`
`\c_lngx_default_sfdefault_tl` * hook. These are not supposed to be changed and hence they are set with the `c` prefix.
`\c_lngx_default_ttdefault_tl` *

`\l_lngx_current_encoding_tl` * These `tl`s expand to the current values of encoding, meta family, super family,
`\l_lngx_current_meta_family_tl` * series and shape respectively. Note that these are updated time to time by the
`\l_lngx_current_super_family_tl` * commands that change them (package-internal or L^AT_EX-internal).
`\l_lngx_current_series_tl` *
`\l_lngx_current_shape_tl` *

`\lngx_if_encoding_p:n` * $\{\langle encoding \rangle\}$
`\lngx_if_encoding:nTF` * $\{\langle encoding \rangle\}\{\langle true code \rangle\}\{\langle false code \rangle\}$
`\lngx_if_meta_family_p:n` * $\{\langle meta font family \rangle\}$
`\lngx_if_meta_family:nTF` * $\{\langle meta font family \rangle\}\{\langle true code \rangle\}\{\langle false code \rangle\}$
`\lngx_if_super_family_p:n` * $\{\langle super font family \rangle\}$
`\lngx_if_super_family:nTF` * $\{\langle super font family \rangle\}\{\langle true code \rangle\}\{\langle false code \rangle\}$
`\lngx_if_series_p:n` * $\{\langle series \rangle\}$
`\lngx_if_series:nTF` * $\{\langle series \rangle\}\{\langle true code \rangle\}\{\langle false code \rangle\}$
`\lngx_if_shape_p:n` * $\{\langle shape \rangle\}$
`\lngx_if_shape:nTF` * $\{\langle shape \rangle\}\{\langle true code \rangle\}\{\langle false code \rangle\}$

`\lngx_if_meta_family_rm_p:` *
`\lngx_if_meta_family_rm:TF` * $\{\langle true code \rangle\}\{\langle false code \rangle\}$
`\lngx_if_meta_family_sf_p:` *
`\lngx_if_meta_family_sf:TF` * $\{\langle true code \rangle\}\{\langle false code \rangle\}$
`\lngx_if_meta_family_tt_p:` *
`\lngx_if_meta_family_tt:TF` * $\{\langle true code \rangle\}\{\langle false code \rangle\}$

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 * {\langle true code \rangle}{\langle false code \rangle}
\lngx_if_series_bf_p: *
\lngx_if_series_bf:TF * {\langle true code \rangle}{\langle false code \rangle}

```

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 * {\langle true code \rangle}{\langle false code \rangle}
\lngx_if_shape_it_p: *
\lngx_if_shape_it:TF * {\langle true code \rangle}{\langle false code \rangle}
\lngx_if_shape_sc_p: *
\lngx_if_shape_sc:TF * {\langle true code \rangle}{\langle false code \rangle}
\lngx_if_shape_ssc_p: *
\lngx_if_shape_ssc:TF * {\langle true code \rangle}{\langle false code \rangle}
\lngx_if_shape_sl_p: *
\lngx_if_shape_sl:TF * {\langle true code \rangle}{\langle false code \rangle}
\lngx_if_shape_sw_p: *
\lngx_if_shape_sw:TF * {\langle true code \rangle}{\langle false code \rangle}
\lngx_if_shape_ulc_p: *
\lngx_if_shape_ulc:TF * {\langle true code \rangle}{\langle false code \rangle}

```

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 {\langle family ID \rangle} {\langle rm=\langle rm NFSS \rangle, sf=\langle sf NFSS \rangle, tt=\langle tt NFSS \rangle \rangle}

```

This function takes an $\langle ID \rangle$ and sets the `rm`, `sf`, `tt` values as requested by the user and creates a super font family.

```

\lngx_soft_super_font_family:nn {\langle ID \rangle}{\langle encoding, family, series, shape \rangle}
\lngx_softer_super_font_family:n {\langle ID \rangle}
\lngx_softest_super_font_family:n {\langle ID \rangle}

```

The `\lngx_soft_super_font_family:nn` sets super family marked by the $\langle ID \rangle$ 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 {\langle ID \rangle}
\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.

LINGUIS $\overline{\text{T}}$ IX

Provide the package with its basic information.

```
1  \langle *package \rangle
2  \ProvidesExplPackage{linguistix}
3                        {2026-04-27}
4                        {v0.9b}
5                        {%
6                          The ‘Linguis $\overline{\text{T}}$ IX’ bundle: Enhanced
7                          support for linguistics.%
8                        }
```

When one loads LINGUIS $\overline{\text{T}}$ IX, all the packages of the bundle are loaded automatically. That’s the only content of the umbrella package LINGUIS $\overline{\text{T}}$ IX. All the packages are loaded conditionally (i.e., only if not loaded already).

```
9
10 \IfPackageLoadedF { linguistix-base } {
11   \RequirePackage { linguistix-base }
12 }
13 \IfPackageLoadedF { linguistix-fonts } {
14   \RequirePackage { linguistix-fonts }
15 }
16 \IfPackageLoadedF { linguistix-glossing } {
17   \RequirePackage { linguistix-glossing }
18 }
19 \IfPackageLoadedF { linguistix-ipa } {
20   \RequirePackage { linguistix-ipa }
21 }
22 \IfPackageLoadedF { linguistix-languages } {
23   \RequirePackage { linguistix-languages }
24 }
25 \IfPackageLoadedF { linguistix-leipzig } {
26   \RequirePackage { linguistix-leipzig }
27 }
28 \IfPackageLoadedF { linguistix-logos } {
29   \RequirePackage { linguistix-logos }
30 }
31 \IfPackageLoadedF { linguistix-nfss } {
32   \RequirePackage { linguistix-nfss }
33 }
34 \rangle package \rangle
```


Set the essentials of the package.

```

35 <*base>
36 \ProvidesExplPackage{linguistix-base}
37     {2026-04-27}
38     {v0.9b}
39     {%
40         The base package of the ‘LinguisTiX’
41         bundle.%
42     }
```

\lngx_set_keys:n I use the `l3keys` module of L^AT_EX₃ 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.

```

43
44 \cs_new_protected:Npn \lngx_set_keys:n #1 {
45     \keys_set:nn { lngx_keys } { #1 }
46 }
```

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

\linguistix I equate this command with a user-side macro here and end the LINGUIS**TiX**-BASE package.

```

47
48 \cs_gset_eq:NN \linguistix \lngx_set_keys:n
49 </base>
```

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

The `unicode-math` and `lua-unicode-math` packages define `\gla` command 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 `(lua)-unicode-math-\gla`. This is not relevant to people who are not using `expex`. Thus, the settings are loaded only conditionally.

```

50 <fixpex>
51 \ProvidesExplPackage{linguistix-fixpex}
52     {2026-04-27}
53     {v0.9b}
54     {%
55         To fix the clash between 'expex' and
56         (lua-unicode-math).%
57     }

```

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

```

58
59 \msg_new:nnn { fixpex } { pkg_not_loaded } {
60     The~ 'LinguisTiX-fixpex'~ package~ is~ a~ first-aid~
61     for~ resolving~ the~ clash~ between~
62     '(lua)-unicode-math'\ and~ 'expex'.~ It~ should~ only~
63     be~ used~ if~ at~ least~ one~ of~ the~ two~ is~ loaded.~
64     Here~ 'LinguisTiX-fixpex'\ is~ not~ needed~ as~ you~
65     '#1'~ is~ not~ loaded.
66 }

```

I first start the hook `begindocument/before`.

```

67
68 \hook_gput_code:nnn { begindocument / before } { . } {

```

The `(lua)-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.

```

69     \IfPackageLoadedTF { expex } {
70         \exp_args:Ne
71         \IfPackageLoadedTF {
72             \sys_if_engine luatex:TF {
73                 \IfPackageLoadedF { unicode-math } {
74                     unicode-math
75                 } {
76                     lua-unicode-math
77                 }
78             } {
79                 unicode-math
80             }
81         } {
82             \hook_gput_code:nnn { begindocument / end } { . } {

```

\umgla This replicates the `(lua)-unicode-math-\gla` for future use.

```

83         \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).

```

84         \cs_gset_eq:NN \gla    \glw@gla
85     }

```

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

```

86     } {
87         \msg_info:nnn { fixpex } { pkg_not_loaded } {
88             (lua)-unicode-math
89         }
90     }

```

Similarly, I do it for expex.

```

91     } {
92         \msg_info:nnn { fixpex } { pkg_not_loaded } {
93             expex
94         }
95     }
96 }
97 </fixpex>

```

Package essentials first.

```

98  \font
99  \ProvidesExplPackage{linguistix-fonts}
100      {2026-04-27}
101      {v0.9b}
102      {%
103          The font-assistant package of the
104          'LinguistTiX' bundle.%
105      }

```

Then, I load unicode-math or lua-unicode-math (depending on the engine used), LINGUISTIX-NFSS and LINGUISTIX-BASE (if they are not already loaded).

```

106
107  \IfPackageLoadedF { linguistix-base } {
108      \RequirePackage { linguistix-base }
109  }
110
111  \sys_if_engine_luatex:TF {
112      \IfPackageLoadedF { unicode-math } {
113          \IfPackageLoadedF { lua-unicode-math } {
114              \RequirePackage { fontspec, lua-unicode-math }
115          }
116      }
117  } {
118      \IfPackageLoadedF { unicode-math } {
119          \RequirePackage { unicode-math }
120      }
121  }
122
123  \IfPackageLoadedF { linguistix-fixpex } {
124      \RequirePackage { linguistix-fixpex }
125  }

```

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

```

126
127  \NewCommandCopy \ogLaTeX \LaTeX
128
129  \RenewDocumentCommand \LaTeX { } {%
130      L\kern-.81ex\relax
131      \raisebox{.6ex}{\textsc{a}}\kern-.23ex\relax
132      \hbox{T}\kern-.4ex\relax
133      \raisebox{-.5ex}{E}\kern-.3ex\relax
134      X%
135  }

```

(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.

```

136
137  \keys_define:nn { lngx_keys } {
138      old~ style~ numbers
139      .bool_gset:N          = {

```

```

140     \g_lngx_old_style_bool
141 },
142 old~ style~ one
143 .bool_gset:N          = {
144     \g_lngx_old_style_one_bool
145 },
146 bourbaki's~ empty~ set
147 .bool_gset:N          = {
148     \g_lngx_bourbaki_bool
149 }
150 }

```

(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 **tl**s 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>.)

```

151
152 \prop_gclear_new:N \g_lngx_text_main_fonts_prop
153 \tl_gclear_new:N \g_lngx_text_main_font_features_tl
154
155 \clist_map_inline:nn {
156     upright,
157     bold~ upright,
158     italic,
159     bold~ italic,
160     slanted,
161     bold~ slanted,
162     swash,
163     bold~ swash,
164     small~ caps
165 } {

```

All the keys listed here use a global token list to have an appending list of features. The variables are declared here.

```

166 \group_begin:
167 \str_clear:N \l_tmpa_str
168 \str_set:Nn \l_tmpa_str { #1 }
169 \str_replace_all:Nnn \l_tmpa_str { ~ } { _ }
170 \tl_gclear_new:c {
171     g_lngx_text_main_features_ \l_tmpa_str _tl
172 }
173 \group_end:

```

All the keys here are prefixed with the word **text** in order to distinguish them from the keys provided by the **LINGUIS_{TL}X-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.

```

174 \keys_define:nn { lngx_keys } {
175   text~ #1
176   .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`.

```

177   \group_begin:
178   \str_clear:N \l_tmpa_str
179   \str_set:Ne \l_tmpa_str {
180     \text_titlecase_all:n { #1 }
181     Font
182   }
183   \str_replace_all:Nnn \l_tmpa_str { ~ } { }

```

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

```

184   \prop_gput:Nne \g__lngx_text_main_fonts_prop
185                 { text~ #1 }
186                 { \str_use:N \l_tmpa_str = { ##1 } }
187   \group_end:
188 },

```

Same is repeated for features, but we add the value of the keys to a continuously appending token list first and then expand it in the `prop` variable.

```

189   text~ #1~ features
190   .code:n          = {
191     \group_begin:
192     \str_clear:N \l_tmpa_str
193     \str_clear:N \l_tmpb_str
194     \str_set:Ne \l_tmpa_str {
195       \text_titlecase_all:n { #1 }
196       Features
197     }
198     \str_set:Nn \l_tmpb_str { #1 }
199     \str_replace_all:Nnn \l_tmpa_str { ~ } { }
200     \str_replace_all:Nnn \l_tmpb_str { ~ } { _ }
201     \tl_gput_right:ce {
202       g__lngx_text_main_features_ \l_tmpb_str _tl
203     } {
204       \tl_if_empty:cF {
205         g__lngx_text_main_features_ \l_tmpb_str _tl
206       } { , }
207       \exp_not:n { ##1 }
208     }
209     \prop_gput:Nne \g__lngx_text_main_fonts_prop
210                   { text~ #1~ features } {
211       \str_use:N \l_tmpa_str = {
212         \exp_not:v {
213           g__lngx_text_main_features_ \l_tmpb_str _tl
214         }
215       }
216     }

```

```

217     \group_end:
218   }
219 }
220 }

```

(End of definition for `\g__lngx_text_main_fonts_prop` and others. These functions are documented on page [II](#).)

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

```

221
222 \tl_gclear_new:N \g__lngx_text_main_features_extra_tl
223
224 \keys_define:nn { lngx_keys } {
225   text~ main~ extra~ features
226     .code:n = {
227       \tl_gput_right:N \g__lngx_text_main_features_extra_tl {
228         \tl_if_empty:NF \g__lngx_text_main_features_extra_tl {
229           ,
230         }
231       \exp_not:n { #1 }
232     }
233   \prop_gput:Nne \g__lngx_text_main_fonts_prop
234     { text~ main~ extra~ features } {
235     \exp_not:V \g__lngx_text_main_features_extra_tl
236   }
237 }
238 }

```

(End of definition for `text main extra features`. This function is documented on page [I3](#).)

```

\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 extra features
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 extra 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.

```

239
240 \prop_gclear_new:N \g__lngx_text_sans_fonts_prop
241 \tl_gclear_new:N \g__lngx_text_sans_font_features_tl
242
243 \prop_gclear_new:N \g__lngx_text_mono_fonts_prop
244 \tl_gclear_new:N \g__lngx_text_mono_font_features_tl
245
246 \clist_map_inline:nn {
247   sans,
248   mono
249 } {
250   \tl_gclear_new:c {
251     g__lngx_text_ #1 _features_extra_tl
252   }
253   \clist_map_inline:nn {
254     upright,
255     bold~ upright,
256     italic,
257     bold~ italic,
258     slanted,
259     bold~ slanted,
260     swash,
261     bold~ swash,
262     small~ caps
263   } {
264     \group_begin:
265     \str_clear:N \l_tmpa_str
266     \str_set:Nn \l_tmpa_str { ##1 }
267     \str_replace_all:Nnn \l_tmpa_str { ~ } { _ }
268     \tl_gclear_new:c {
269       g__lngx_text_ #1 _features_ \l_tmpa_str _tl
270     }
271     \group_end:
272     \keys_define:nn { lngx_keys } {
273       text~ #1~ ##1
274       .code:n = {
275         \group_begin:
276         \str_clear:N \l_tmpa_str
277         \str_set:Ne \l_tmpa_str {
278           \text_titlecase_all:n { ##1 }
279           Font
280         }
281         \str_replace_all:Nnn \l_tmpa_str { ~ } { }
282         \prop_gput:cne { g__lngx_text_ #1 _fonts_prop }
283           { text~ #1~ ##1 } {
284           \str_use:N \l_tmpa_str = {
285             \exp_not:n { #####1 }
286           }
287         }
288         \group_end:
289       },
290       text~ #1~ ##1~ features

```



```

291 .code:n = {
292   \group_begin:
293   \str_clear:N \l_tmpa_str
294   \str_clear:N \l_tmpb_str
295   \str_set:N \l_tmpa_str {
296     \text_titlecase_all:n { ##1 }
297     Features
298   }
299   \str_set:Nn \l_tmpb_str { ##1 }
300   \str_replace_all:Nnn \l_tmpa_str { ~ } { }
301   \str_replace_all:Nnn \l_tmpb_str { ~ } { _ }
302   \tl_gput_right:ce {
303     g__lngx_text_ #1 _features_ \l_tmpb_str _tl
304   } {
305     \tl_if_empty:cF {
306       g__lngx_text_ #1 _features_ \l_tmpb_str _tl
307     } { , }
308     \exp_not:n { #####1 }
309   }
310   \prop_gput:cne { g__lngx_text_ #1 _fonts_prop }
311     { text~ #1~ ##1~ features } {
312     \str_use:N \l_tmpa_str = {
313       \exp_not:v {
314         g__lngx_text_ #1 _features_ \l_tmpb_str _tl
315       }
316     }
317   }
318   \group_end:
319 }
320 }
321 }
322 \keys_define:nn { lngx_keys } {
323   text~ #1~ extra~ features
324   .code:n = {
325     \tl_gput_right:ce {
326       g__lngx_text_ #1 _features_extra_tl
327     } {
328       \tl_if_empty:cF {
329         g__lngx_text_ #1 _features_extra_tl
330       } { , }
331       \exp_not:n { ##1 }
332     }
333     \prop_gput:cne { g__lngx_text_ #1 _fonts_prop }
334       { text~ #1~ extra~ features } {
335       \exp_not:v { g__lngx_text_ #1 _features_extra_tl }
336     }
337   }
338 }
339 }

```

(End of definition for `\g__lngx_text_sans_fonts_prop` and others. These functions are documented on page 13.)

```

\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.

```

340
341 \clist_map_inline:nn {
342   main,
343   sans,
344   mono
345 } {
346   \keys_define:nn { lngx_keys } {
347     text~ #1~ font
348     .tl_gset:c          = { g__lngx_text_ #1 _font_tl }
349   }
350 }

```

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

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

```

\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
351
352 \prop_gclear_new:N \g__lngx_math_fonts_prop
353 \tl_gclear_new:N \g__lngx_math_features_tl
354
355 \prop_gclear_new:N \g__lngx_math_bold_fonts_prop
356 \tl_gclear_new:N \g__lngx_math_bold_features_tl
357
358 \keys_define:nn { lngx_keys } {
359   math
360   .tl_gset:N          = \g__lngx_math_font_tl,
361   math~ bold
362   .tl_gset:N          = \g__lngx_math_bold_font_tl,
363   math~ features
364   .prop_gput:N        = \g__lngx_math_fonts_prop,
365   math~ bold~ features
366   .prop_gput:N        = \g__lngx_math_bold_fonts_prop
367 }

```

(End of definition for `\g__lngx_math_fonts_prop` and others. These functions are documented on page 7.)

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.

```

368
369 \keys_define:nn { lngx_keys } {
370   newcm
371   .meta:n          = {
372     text~ main~ font    = { NewCM10-Book.otf },
373     text~ sans~ font    = { NewCMSans10-Book.otf },
374     text~ mono~ font    = { NewCMMono10-Book.otf },
375     math                = { NewCMMath-Book.otf },
376     math~ bold          = { NewCMMath-Bold.otf }
377   }
378 }

```

(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.

```

379
380 \keys_define:nn { lngx_keys } {
381   newcm~ sans

```

```

382 .meta:n          = {
383   text~ main~ font = { NewCMSans10-Book.otf },
384   text~ sans~ font = { NewCMSans10-Book.otf },
385   text~ mono~ font = { NewCMMono10-Book.otf },
386   math           = { NewCMSansMath-Regular.otf },
387   math~ bold     = { NewCMSansMath-Regular.otf }
388 }
389 }

```

(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.

```

390
391 \keys_define:nn { lngx_keys } {
392   newcm~ mono
393   .meta:n          = {
394     text~ main~ font = { NewCMMono10-Book.otf },
395     text~ sans~ font = { NewCMSans10-Book.otf },
396     text~ mono~ font = { NewCMMono10-Book.otf },
397     math           = { NewCMSansMath-Regular.otf },
398     math~ bold     = { NewCMSansMath-Regular.otf }
399   }
400 }

```

(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.

```

401
402 \keys_define:nn { lngx_keys } {
403   newcm~ regular
404   .meta:n          = {
405     text~ main~ font = { NewCM10-Regular.otf },
406     text~ sans~ font = { NewCMSans10-Regular.otf },
407     text~ mono~ font = { NewCMMono10-Regular.otf },
408     math           = { NewCMMath-Regular.otf },
409     math~ bold     = { NewCMMath-Bold.otf }
410   }
411 }

```

(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.

```

412
413 \keys_define:nn { lngx_keys } {
414   newcm~ regular~ sans
415   .meta:n          = {
416     text~ main~ font = { NewCMSans10-Regular.otf },
417     text~ sans~ font = { NewCMSans10-Regular.otf },
418     text~ mono~ font = { NewCMMono10-Regular.otf },
419     math           = { NewCMMath-Regular.otf },
420     math~ bold     = { NewCMMath-Bold.otf }
421   }
422 }

```

(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.

```

423
424 \keys_define:nn { lngx_keys } {
425   newcm~ regular~ mono
426   .meta:n          = {
427     text~ main~ font      = { NewCMMono10-Regular.otf },
428     text~ sans~ font      = { NewCMSans10-Regular.otf },
429     text~ mono~ font      = { NewCMMono10-Regular.otf },
430     math                = { NewCMMath-Regular.otf },
431     math~ bold           = { NewCMMath-Bold.otf },
432   }
433 }
```

(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.

```

434
435 \lngx_set_keys:n {
436   bourbaki's~ empty~ set,
```

Then we load the `old style numbers` boolean.

```

437   old~ style~ numbers,
438   newcm
439 }
```

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

```

440
441 \IfPackageLoadedF { linguistix-languages } {
442   \cs_new_protected:Npn \lngx_set_main_font:nn #1#2 {
443     \setmainfont [ #1 ] { #2 }
444   }
445   \cs_new_protected:Npn \lngx_set_sans_font:nn #1#2 {
446     \setsansfont [ #1 ] { #2 }
447   }
448   \cs_new_protected:Npn \lngx_set_mono_font:nn #1#2 {
449     \setmonofont [ #1 ] { #2 }
450   }
451 }
```

A wrapper command is provided for loading math fonts.

```

452
453 \cs_new_protected:Npn \lngx_set_math_font:nn #1#2 {
454   \setmathfont [ #1 ] { #2 }
455 }
456
457 \cs_new_protected:Npn \lngx_set_math_bold_font:nn #1#2 {
458   \IfPackageLoadedT { lua-unicode-math } {
459     \DeclareMathVersion { bold }
460   }
```

```

461 \setmathfont [
462   #1,
463   version          = { bold }
464 ] { #2 }
465 }

```

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

```

466
467 \cs_generate_variant:Nn \lngx_set_main_font:nn { VV }
468 \cs_generate_variant:Nn \lngx_set_sans_font:nn { VV }
469 \cs_generate_variant:Nn \lngx_set_mono_font:nn { VV }
470 \cs_generate_variant:Nn \lngx_set_math_font:nn { VV }
471 \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 20.)

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.

```

\__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
472
473 \clist_map_inline:nn {
474   main,
475   sans,
476   mono
477 } {
478   \cs_new_protected:cpn {
479     __lngx_build_ #1 _font_features:
480   } {
481     \prop_map_inline:cn { g__lngx_text_ #1 _fonts_prop } {
482       \tl_gput_right:cn {
483         g__lngx_text_ #1 _font_features_tl
484       } { ####2, }
485     }
486   }
487 }
488
489 \cs_new_protected:Npn \__lngx_build_math_features: {
490   \prop_map_inline:Nn \g__lngx_math_fonts_prop {
491     \tl_gput_right:Nn \g__lngx_math_features_tl {
492       { ##2 }
493     }
494   }
495 }
496
497 \cs_new_protected:Npn \__lngx_build_math_bold_features: {
498   \prop_map_inline:Nn \g__lngx_math_bold_fonts_prop {
499     \tl_gput_right:Nn \g__lngx_math_bold_features_tl {
500       { ##2 }
501     }
502   }
503 }

```

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

Now I start the pre-begindocument hook.

```

504
505 \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.

```

506 \lngx_set_keys:n {
507   text~ main~ extra~
508   features          = {
509     \bool_if:NT \g_lngx_old_style_bool {
510       Numbers        = { OldStyle },
511       \bool_if:NT \g_lngx_old_style_one_bool {
512         CharacterVariant = { 6 }
513       }
514     }
515   },
516   text~ sans~ extra~
517   features          = {
518     \bool_if:NT \g_lngx_old_style_bool {
519       Numbers        = { OldStyle },
520       \bool_if:NT \g_lngx_old_style_one_bool {
521         CharacterVariant = { 6 }
522       }
523     }
524   }
525 }

```

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

```

526 \__lngx_build_main_font_features:
527 \lngx_set_main_font:VV
528   \g__lngx_text_main_font_features_tl
529   \g__lngx_text_main_font_tl
530 \__lngx_build_sans_font_features:
531 \lngx_set_sans_font:VV
532   \g__lngx_text_sans_font_features_tl
533   \g__lngx_text_sans_font_tl
534 \__lngx_build_mono_font_features:
535 \lngx_set_mono_font:VV
536   \g__lngx_text_mono_font_features_tl
537   \g__lngx_text_mono_font_tl
538 \__lngx_build_math_features:
539 \lngx_set_math_font:VV \g__lngx_math_features_tl
540                       \g__lngx_math_font_tl
541 \IfPackageLoadedT { unicode-math } {
542   \__lngx_build_math_bold_features:
543   \lngx_set_math_bold_font:VV
544     \g__lngx_math_bold_features_tl
545     \g__lngx_math_bold_font_tl
546 }
547 }
548 </font>

```

```

549 <*glossing>
550 \ProvidesExplPackage{linguistix-glossing}
551     {2026-04-27}
552     {v0.9b}
553     {%
554     Accessible glossing with Linguistix%
555     }

```

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

```

556
557 \IfPackageLoadedF { multicol } {
558   \RequirePackage { multicol }
559 }

```

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

```

560
561 \bool_new:N      \l_lngx_expansion_bool
562 \tl_clear_new:N \l_lngx_gloss_separator_tl
563 \tl_clear_new:N \l_lngx_expansion_separator_tl
564 \tl_clear_new:N \l_lngx_glossary_separator_tl
565 \dim_zero_new:N \l_lngx_i_have_dim
566 \dim_zero_new:N \l_lngx_i_need_dim
567 \dim_zero_new:N \l_lngx_remain_dim
568 \dim_zero_new:N \l_lngx_i_hack_dim
569 \int_gzero_new:N \g__lngx_page_ref_int
570 \str_clear_new:N \l_lngx_gls_language_str
571 \str_clear_new:N \l__lngx_gls_sorting_order_str
572 \str_clear_new:N \l__lngx_gls_expansion_case_str
573 \str_clear_new:N \l__lngx_glossary_style_str
574 \str_clear_new:N \l__lngx_separator_str
575 \seq_gclear_new:N \g__lngx_gls_use_order_seq
576
577 \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.

```

578
579 \IfPDFManagementActiveT {
580   \socket_if_exist:nT { hyp / link / GoTo / Contents } {
581     \socket_new_plug:nnn { hyp / link / GoTo / Contents }
582       { text } {
583         \pdfstringdef \__lngx_tmp_text: { #2 }
584         \pdfannot_dict_put:nne { link / GoTo } { Contents } {
585           ( \__lngx_tmp_text: )
586         }
587       }
588   }
589 }

```

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

```

590
591 \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.

```

592
593 \socket_new_plug:nnn { lngx / description / gloss } { on } {
594   \cs_set:Npn \__lngx_gloss_description: { Gloss~ }
595 }
596
597 \socket_new_plug:nnn { lngx / description / gloss }
598   { off } {
599   \cs_set_eq:NN \__lngx_gloss_description: \prg_do_nothing:
600 }
601
602 \socket_assign_plug:nn { lngx / description / gloss }
603   { off }
604
605 \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.

```

606
607 \NewTaggingSocket { lngx / gloss } { 2 }
608
609 \NewTaggingSocketPlug { lngx / gloss } { default } {
610   \mode_leave_vertical:
611   \tag_mc_end:
612   \exp_args:Ne
613   \tag_struct_begin:n {
614     tag                = { Span },
615     E                  = {
616       \__lngx_gloss_description: #2
617     }
618   }
619   \tag_mc_begin:n {
620     tag                = { Span }
621   }

```

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).

```

622 \IfPackageLoadedTF { hyperref } {
623   \group_begin:
624   \str_clear:N \l_tmpa_str
625   \str_set:Nn \l_tmpa_str { #1 }
626   \exp_args:Ne \hypersetup {
627     linkcolor      = {

```



```

628     \exp_not:V \g__lngx_gloss_link_color_str
629   }
630 }

```

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

```

631 \IfPDFManagementActiveT {
632   \socket_if_exist:nT { hyp / link / GoTo / Contents } {
633     \socket_assign_plug:nn {
634       hyp / link / GoTo / Contents
635     } { text }
636   }
637 }
638 \lngx_gloss_format:n {
639   \hyperlink { lngx_ #1 _glossary } { #1 }
640 }
641 \group_end:
642 } {

```

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

```

643   \lngx_gloss_format:n { #1 }
644 }
645 \tag_mc_end:
646 \tag_struct_end:
647 \tag_mc_begin:n { }
648 }

```

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

```

649
650 \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.

```

651
652 \keys_define:nn { lngx_glossing } {
653   format
654   .meta:nn          = { lngx / gloss / format } { #1 },

```

(End of definition for `format`. This function is documented on page 9.)

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

```

\g__lngx_gloss_link_color_str
655 link~ color
656 .str_gset:N          = \g__lngx_gloss_link_color_str,
657 link~ color
658 .initial:n           = { black },

```

(End of definition for `link color` and `\g__lngx_gloss_link_color_str`. This function is documented on page 9.)

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.

```

659 sort
660 .choices:nn          = { alphabetical, use } {
661   \str_set_eq:NN \l__lngx_gls_sorting_order_str
662                 \l_keys_choice_str
663 },
664 sort
665 .initial:n           = { alphabetical },

```

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

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.

```

666   expansion~ case
667   .choices:nn          = {
668     lowercase, title~ case~ all, title~ case~ first
669   } {
670     \str_set_eq:NN \l__lngx_gls_expansion_case_str
671                   \l_keys_choice_str
672   },
673   expansion~ case
674   .initial:n           = { lowercase },

```

(End of definition for *expansion case* and `\l__lngx_gls_expansion_case_str`. This function is documented on page 9.)

style `\l__lngx_glossary_style_str` The glossary can be printed in two styles given below. The default is **block**.

```

675   style
676   .choices:nn          = { block, inline } {
677     \str_set_eq:NN \l__lngx_glossary_style_str
678                   \l_keys_choice_str
679   },
680   style
681   .initial:n           = { block },

```

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

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.

```

682   columns
683   .int_set:N           = \l__lngx_glossary_columns_int,
684   columns
685   .initial:n           = { 2 },

```

(End of definition for *columns* and `\l__lngx_glossary_columns_int`. This function is documented on page 10.)

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

```

686   page~ numbers
687   .bool_set:N          =
688     \l__lngx_glosses_page_number_bool,
689   page~ numbers
690   .initial:n           = { true },

```

(End of definition for *page numbers* and `\l__lngx_glosses_page_number_bool`. This function is documented on page 10.)

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.

```

691   sectioning
692   .str_set:N           = \l__lngx_gls_sectioning_str,
693   sectioning
694   .initial:n           = { section },

```

(End of definition for sectioning and \l__lngx_gls_sectioning_str. This function is documented on page 10.)

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

```

695 section~ number
696 .bool_set:N          = \l__lngx_gls_section_number_bool,
697 section~ number
698 .initial:n           = { false },

```

(End of definition for section number and \l__lngx_gls_section_number_bool. This function is documented on page 10.)

no bold 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).

```

699 no~ bold
700 .bool_set_inverse:N   = \l__lngx_gls_bold_bool,
701 no~ bold
702 .initial:n           = { false },

```

(End of definition for no bold and \l__lngx_gls_bold_bool. This function is documented on page 10.)

separator 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.

```

703 separator
704 .code:n              = {
705   \tl_set:cn {
706     l_lngx_
707     \str_use:N \l__lngx_separator_str
708     _separator_tl
709   } { #1 }
710 },
711 separator
712 .initial:n           = { ,~ },

```

(End of definition for separator and \l__lngx_separator_tl. This function is documented on page 10.)

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

```

713 entry~ separator
714 .tl_set:N            = \l__lngx_entry_separator_tl,
715 entry~ separator
716 .initial:n           = { \par }
717 }

```

(End of definition for entry separator and \l__lngx_entry_separator_tl. This function is documented on page 10.)

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

```

718
719 \IfPDFManagementActiveT {

```

```

720 \str_set:Ne \l_lngx_gls_language_str {
721   \GetDocumentProperties { document / lang }
722 }
723 }

```

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

```

\lngx_gloss_format:n
724
725 \keys_define:nn { lngx / gloss / format } {
726   gloss
727   .cs_gset_protected:Np = \lngx_gloss_format:n #1,
728   gloss
729   .initial:n           = { \textsc { #1 } },

```

(End of definition for `gloss` and `\lngx_gloss_format:n`. These functions are documented on page 9.)

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

```

\lngx_expansion_format:n
730   expansion
731   .cs_gset_protected:Np = \lngx_expansion_format:n #1,
732   expansion
733   .initial:n           = { #1 }
734 }

```

(End of definition for `expansion` and `\lngx_expansion_format:n`. These functions are documented on page 9.)

\setupglossing A wrapper around these options is provided.

```

735
736 \NewDocumentCommand \setupglossing { m } {
737   \keys_set:nn { lngx_glossing } { #1 }
738 }

```

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

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

```

\lngx_gloss_new:nn
739
740 \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.

```

741   \group_begin:
742   \str_clear:N \l_tmpa_str
743   \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).

```

744   \tl_gclear_new:c {
745     g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
746   }
747   \seq_gclear_new:c {
748     g_lngx_ \str_use:N \l_tmpa_str _pages_seq
749   }
750   \tl_gset:cn {
751     g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
752   } { #2 }

```

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

```

753 \cs_new_protected:cpn {
754   __lngx_gloss_ \str_use:N \l_tmpa_str :
755 } {

```

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`. This is done only if the `\DocumentMetadata` command is used.

```

756   \IfDocumentMetadataTF {
757     \exp_args:Nee \UseTaggingSocket
758       { lngx / gloss }
759       { \str_use:N \l_tmpa_str }
760       { #2 }
761   } {
762     \IfPackageLoadedTF { hyperref } {
763       \group_begin:
764       \exp_args:Ne \hypersetup {
765         linkcolor = {
766           \exp_not:V \g__lngx_gloss_link_color_str
767         }
768       }
769       \IfPDFManagementActiveT {
770         \socket_if_exist:nT {
771           hyp / link / GoTo / Contents
772         } {
773           \socket_assign_plug:nn {
774             hyp / link / GoTo / Contents
775           } { text }
776         }
777       }
778       \lngx_gloss_format:n {
779         \hyperlink { lngx_ #1 _glossary } { #1 }
780       }
781       \group_end:
782     } {
783       \lngx_gloss_format:n { #1 }
784     }
785   }

```

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. 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 to the sequences that respectively store the page numbers of glosses and the order in which they were used. Imagine if a gloss is used twice on a page, it doesn't make sense to add the same page number twice. Similarly, if a gloss is used, it is added to the sequence of used glosses. It doesn't make sense to add it 10 times again and removing the 9 duplicates later.

```

786   \int_gincr:N \g__lngx_page_ref_int
787   \exp_args:Ne
788   \label { lngx_gloss_ \int_use:N \g__lngx_page_ref_int }
789   \cs_if_exist:cT {

```

```

790     r @ lngx_gloss_ \int_use:N \g__lngx_page_ref_int
791 } {
792   \group_begin:
793   \tl_clear:N \l_tmpa_tl
794   \tl_set:N \l_tmpa_tl {
795     \exp_not:N \use_ii:nnnnn
796     \use:c {
797       r @ lngx_gloss_ \int_use:N \g__lngx_page_ref_int
798     }
799   }
800   \seq_if_in:cVF {
801     g_lngx_ \str_use:N \l_tmpa_str _pages_seq
802   } \l_tmpa_tl {
803     \seq_gput_right:ce {
804       g_lngx_ \str_use:N \l_tmpa_str _pages_seq
805     } {
806       \exp_not:N \use_ii:nnnnn
807       \use:c {
808         r @ lngx_gloss_ \int_use:N \g__lngx_page_ref_int
809       }
810     }
811   }
812   \group_end:
813 }
814 \seq_if_in:NeF \g__lngx_gls_use_order_seq {
815   \str_use:N \l_tmpa_str
816 } {
817   \seq_gput_right:Ne \g__lngx_gls_use_order_seq
818     { \str_use:N \l_tmpa_str }
819 }
820 }
821 \group_end:
822 }
823
824 \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 L^AT_EX₃ equivalent for this is provided.

```

825
826 \NewDocumentCommand \renewgloss { m m } {
827   \cs_undefine:c { __lngx_gloss_ #1 : }
828   \lngx_gloss_new:nn { #1 } { #2 }
829 }

```

(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.

```

830
831 \NewDocumentCommand \glx { s O{ } m } {
832   \group_begin:
833   \IfBooleanT { #1 } {
834     \bool_set_true:N \l_lngx_expansion_bool
835     \str_set:Nn \l__lngx_separator_str { expansion }
836     \keys_set:nn { lngx_glossing } {
837       separator = { , \c_space_tl }
838     }
839   }
840   \keys_set:nn { lngx_glossing } { #2 }
841   \tl_clear:N \l_tmpa_tl
842   \seq_clear:N \l_tmpa_seq
843   \seq_set_from_clist:Nn \l_tmpa_seq { #3 }
844   \seq_pop_left:NN \l_tmpa_seq \l_tmpa_tl
845   \str_set:Ne \l_tmpa_str {
846     \exp_args:Ne \str_lowercase:n {
847       \tl_use:N \l_tmpa_tl
848     }
849   }
850   \bool_if:NTF \l_lngx_expansion_bool {
851     \str_case:Vn \l__lngx_gls_expansion_case_str {
852       { lowercase } {
853         \text_lowercase:n {
854           \tl_use:c {
855             g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
856           }
857         }
858       }
859       { title~ case~ all } {
860         \text_titlecase_all:n {
861           \tl_use:c {
862             g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
863           }
864         }
865       }
866       { title~ case~ first } {
867         \text_titlecase_first:n {
868           \tl_use:c {
869             g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
870           }
871         }
872       }
873     } {
874       \use:c { __lngx_gloss_ \str_use:N \l_tmpa_str : }
875     }
876   }

```

```

877 \seq_if_empty:NF \l_tmpa_seq {
878   \seq_map_inline:Nn \l_tmpa_seq {
879     \group_begin:
880     \str_clear:N \l_tmpa_str
881     \str_set:Ne \l_tmpa_str {
882       \exp_args:Ne \str_lowercase:n { ##1 }
883     }
884     \bool_if:NTF \l_lngx_expansion_bool {
885       \str_case:Vn \l__lngx_gls_expansion_case_str {
886         { lowercase } {
887           \tl_use:N \l_lngx_expansion_separator_tl
888           \text_lowercase:n {
889             \tl_use:c {
890               g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
891             }
892           }
893         }
894         { title~ case~ all } {
895           \tl_use:N \l_lngx_expansion_separator_tl
896           \text_titlecase_all:n {
897             \tl_use:c {
898               g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
899             }
900           }
901         }
902         { title~ case~ first } {
903           \tl_use:N \l_lngx_expansion_separator_tl
904           \text_titlecase_first:n {
905             \tl_use:c {
906               g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
907             }
908           }
909         }
910       }
911     } {
912       \tl_use:N \l_lngx_gloss_separator_tl
913       \use:c { __lngx_gloss_ \str_use:N \l_tmpa_str : }
914     }
915     \group_end:
916   }
917 }
918 \group_end:
919 }

```

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

`__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).

```

920
921 \cs_new_protected:Npn \__lngx_dotfill:nnn #1#2#3 {
922   %% Courtesy: Jonathan P. Spratte

```



```

923 %% topanswers.xyz/tex?q=8155#a7758 (LPPL)
924 \l__lngx_entry_separator_tl
925 \smallskip
926 \group_begin:
927 \rightskip          = 0pt plus -1fil \prg_do_nothing:
928 \parfillskip        = 0pt plus 1fil \prg_do_nothing:
929 \leftskip           = 1em plus 1fil \prg_do_nothing:
930 \finalhyphendemerits = 0           \prg_do_nothing:
931 \parindent          = -1em         \prg_do_nothing:
932 \bool_if:NT \l__lngx_gls_bold_bool { \textbf } {
933   \lngx_gloss_format:n {
934     #1
935   }
936   \tl_use:N \l__lngx_glossary_separator_tl
937 }
938 #2
939 \bool_if:NT \l__lngx_glosses_page_number_bool {
940   \mode_leave_vertical:
941   \quad
942   \IfDocumentMetadataT {
943     \tag_mc_end:
944     \tag_struct_begin:n {
945       tag                = { Span },
946       actualtext         = { }
947     }
948     \tag_mc_begin:n {
949       tag                = { Span }
950     }
951   }
952   \cleaders
953     \hbox to 0.44em { \hss . \hss }
954     \hskip 0.5cm plus 1fill \prg_do_nothing:
955   \IfDocumentMetadataT {
956     \tag_mc_end:
957     \tag_struct_end:
958     \tag_mc_begin:n { }
959   }
960   \quad
961   \kern 0pt \prg_do_nothing:
962   \em #3
963 }
964 \l__lngx_entry_separator_tl
965 \group_end:
966 }

```

(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.

```

967
968 \NewDocumentEnvironment { lngx_multicols } { m } {
969   \int_compare:nNnTF { 1 } < {
970     \int_use:N \l__lngx_glossary_columns_int
971   } {

```

```

972 \begin { multicol s } {
973 \int _use:N \l __lngx_glossary_columns_int
974 } [ #1 ]
975 } { #1 }
976 \noindent
977 } {
978 \int _compare:nNnT { 1 } < {
979 \int _use:N \l __lngx_glossary_columns_int
980 } {
981 \end { multicol s }
982 }
983 }

```

(End of definition for `lngx_multicol s`. This function is documented on page 20.)

\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.

```

984
985 \cs_new_protected:Npn \lngx_gloss_list: {
986 \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.

```

987 \group_begin:
988 \seq_clear:N \l_tmpa_seq
989 \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 L^AT_EX₃'s mechanism for sorting strings.

```

990 \str_case:Vn \l__lngx_gls_sorting_order_str {
991 { alphabetical } {
992 \seq_sort:Nn \l_tmpa_seq {
993 \str_compare:nNnTF { ##1 } > { ##2 } {
994 \sort_return_swapped:
995 } {
996 \sort_return_same:
997 }
998 }
999 }
1000 }

```

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.

```

1001 \str_if_eq:VnTF \l__lngx_glossary_style_str { inline } {
1002 \group_begin:
1003 \keys_set:nn { lngx_glossing } {
1004 separator = { \c_colon_str \c_space_tl },
1005 entry~ separator = { ,~ }
1006 }

```

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.

```

1007 \tl_clear:N \l_tmpa_tl
1008 \str_clear:N \l_tmpa_str
1009 \seq_pop_left:NN \l_tmpa_seq \l_tmpa_tl
1010 \str_set:NV \l_tmpa_str \l_tmpa_tl
1011 \IfDocumentMetadataT {
1012   \tag_mc_end:
1013   \tag_struct_begin:n {
1014     tag = { Span },
1015   }
1016   \tag_mc_begin:n {
1017     tag = { Span }
1018   }
1019 }
1020 \MakeLinkTarget * {
1021   lngx_ \str_use:N \l_tmpa_str _glossary
1022 }
1023 \bool_if:NT \l__lngx_gls_bold_bool { \textbf } {
1024   \lngx_gloss_format:n {
1025     \tl_use:N \l_tmpa_tl
1026     \tl_use:N \l_lngx_glossary_separator_tl
1027   }
1028 }
1029 \IfDocumentMetadataT {
1030   \tag_mc_end:
1031   \tag_struct_end:
1032 }

```

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

```

1033 \str_case:Vn \l__lngx_gls_expansion_case_str {
1034   { lowercase } {
1035     \lngx_expansion_format:n {
1036       \text_lowercase:n {
1037         \tl_use:c {
1038           g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
1039         }
1040       }
1041     }
1042   }
1043   { title~ case~ all } {
1044     \lngx_expansion_format:n {
1045       \text_titlecase_all:n {
1046         \tl_use:c {
1047           g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
1048         }
1049       }
1050     }
1051   }
1052   { title~ case~ first } {
1053     \lngx_expansion_format:n {
1054       \text_titlecase_first:n {
1055         \tl_use:v {
1056           g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
1057         }
1058       }
1059     }
1060   }
1061 }

```

```

1058     }
1059   }
1060 }
1061 }

```

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.

```

1062   \seq_if_empty:NF \l_tmpa_seq {
1063     \seq_map_inline:Nn \l_tmpa_seq {
1064       \group_begin:
1065       \tl_use:N \l__lngx_entry_separator_tl
1066       \MakeLinkTarget * { lngx_ ##1 _glossary }
1067       \textbf {
1068         \lngx_gloss_format:n {
1069           ##1
1070           \tl_use:N \l_lngx_glossary_separator_tl
1071         }
1072       }
1073       \str_case:Vn \l__lngx_gls_expansion_case_str {
1074         { lowercase } {
1075           \lngx_expansion_format:n {
1076             \text_lowercase:n {
1077               \exp_not:v { g_lngx_ ##1 _expansion_tl }
1078             }
1079           }
1080         }
1081         { title~ case~ all } {
1082           \lngx_expansion_format:n {
1083             \text_titlecase_all:n {
1084               \exp_not:v { g_lngx_ ##1 _expansion_tl }
1085             }
1086           }
1087         }
1088         { title~ case~ first } {
1089           \lngx_expansion_format:n {
1090             \text_titlecase_first:n {
1091               \exp_not:v { g_lngx_ ##1 _expansion_tl }
1092             }
1093           }
1094         }
1095       }
1096       \group_end:
1097     }
1098   }
1099   \group_end:
1100 } {

```

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`.

```

1101   \ProvideDocumentCommand \glossaryname { } { Glossary }

```

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

```

II02 \begin { lngx_multicols } {
II03 \str_if_eq:VnF \l__lngx_gls_sectioning_str { null } {
II04 \use:e {
II05 \exp_not:N \use:c
II06 { \str_use:N \l__lngx_gls_sectioning_str }
II07 \bool_if:NF \l__lngx_gls_section_number_bool { * }
II08 { \exp_not:N \glossaryname }
II09 }
II10 }
II11 }
II12 \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.

```

III3 \group_begin:
III4 \seq_clear:N \l_tmpb_seq
III5 \seq_map_inline:cn { g_lngx_ ##1 _pages_seq } {

```

The pages are hyperlinked with the internal PDF names.

```

III6 \seq_put_right:Ne \l_tmpb_seq { ####1 }
III7 }

```

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

```

III8 \__lngx_dotfill:nnn {
III9 \MakeLinkTarget * { lngx_ ##1 _glossary }
III10 ##1
III11 } {

```

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

```

II22 \str_case:Vn \l__lngx_gls_expansion_case_str {
II23 { lowercase } {
II24 \lngx_expansion_format:n {
II25 \text_lowercase:n {
II26 \exp_not:v { g_lngx_ ##1 _expansion_tl }
II27 }
II28 }
II29 }
II30 { title~ case~ all } {
II31 \lngx_expansion_format:n {
II32 \text_titlecase_all:n {
II33 \exp_not:v { g_lngx_ ##1 _expansion_tl }
II34 }
II35 }
II36 }
II37 { title~ case~ first } {
II38 \lngx_expansion_format:n {
II39 \text_titlecase_first:n {
II40 \exp_not:v { g_lngx_ ##1 _expansion_tl }
II41 }
II42 }
II43 }
II44 }
II45 } {

```

The list of page numbers is printed.

```

1146         \seq_use:Nn \l_tmpb_seq { ,~ }
1147     }
1148     \group_end:
1149 }
1150 \end { lngx_multicols }
1151 }
1152 \group_end:
1153 }

```

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

`\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 \LaTeX_3 function for printing the glosses.

```

1154
1155 \NewDocumentCommand \listofglosses { 0 { } } {
1156     \group_begin:
1157     \str_set:Nn \l_lngx_separator_str { glossary }
1158     \keys_set:nn { lngx_glossing } {
1159         separator = { \c_colon_str \c_space_tl }
1160     }
1161     \keys_set:nn { lngx_glossing } { #1 }
1162     \lngx_gloss_list:
1163     \group_end:
1164 }
1165 \</glossing>

```

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

```

1166 \ipa
1167 \ProvidesExplPackage{linguistix-ipa}
1168     {2026-04-27}
1169     {v0.9b}
1170     {%
1171     A package for typesetting the IPA
1172     (International Phonetic Alphabet) from
1173     the ‘LinguisTiX’ bundle.%
1174     }

```

Then, I load unicode-math or lua-unicode-math (depending on the engine used), LINGUISTIX-NFSS and LINGUISTIX-BASE (if they are not already loaded).

```

1175
1176 \sys_if_engine luatex:TF {
1177     \IfPackageLoadedF { unicode-math } {
1178         \IfPackageLoadedF { lua-unicode-math } {
1179             \RequirePackage { fontspec, lua-unicode-math }
1180         }
1181     }
1182 } {
1183     \IfPackageLoadedF { unicode-math } {
1184         \RequirePackage { unicode-math }
1185     }
1186 }
1187
1188 \IfPackageLoadedF { linguistix-base } {
1189     \RequirePackage { linguistix-base }
1190 }
1191
1192 \IfPackageLoadedF { linguistix-nfss } {
1193     \RequirePackage { linguistix-nfss }
1194 }
1195
1196 \IfPackageLoadedF { linguistix-fixpex } {
1197     \RequirePackage { linguistix-fixpex }
1198 }

```

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

```

1199
1200 \NewDocumentCommand \ipatext { s m } {
1201     \IfBooleanTF { #1 } {
1202         {
1203             \lngxipa
1204             / #2 /
1205         }
1206     } {
1207         {
1208             \lngxipa
1209             [ #2 ]
1210         }
1211     }
1212 }

```

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

```

\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.

```

I2I3
I2I4 \prop_gclear_new:N \g__lngx_ipa_main_fonts_prop
I2I5 \tl_gclear_new:N \g__lngx_ipa_main_font_features_tl
I2I6
I2I7 \clist_map_inline:nn {
I2I8   upright,
I2I9   bold~ upright,
I2I10  italic,
I2I11  bold~ italic,
I2I12  slanted,
I2I13  bold~ slanted,
I2I14  swash,
I2I15  bold~ swash,
I2I16  small~ caps
I2I17 } {
I2I18
I2I19 All the keys here are prefixed with the word ipa in order to distinguish them from the
I2I20 keys provided by the LINGUISTICX-FONTS package. These keys have identical method as
I2I21 their text counterparts, though.
I2I22
I2I23 \group_begin:
I2I24 \str_clear:N \l_tmpa_str
I2I25 \str_set:Nn \l_tmpa_str { #1 }
I2I26 \str_replace_all:Nnn \l_tmpa_str { ~ } { _ }
I2I27 \tl_gclear_new:c {
I2I28   g__lngx_ipa_main_features_ \l_tmpa_str _tl
I2I29 }
I2I30 \group_end:
I2I31 \keys_define:nn { lngx_keys } {
I2I32   ipa~ #1
I2I33   .code:n = {
I2I34     \group_begin:
I2I35     \str_clear:N \l_tmpa_str
I2I36     \str_set:Nn \l_tmpa_str {
I2I37       \text_titlecase_all:n { #1 }
I2I38       Font
I2I39     }
I2I40     \str_replace_all:Nnn \l_tmpa_str { ~ } { }
I2I41     \prop_gput:Nne \g__lngx_ipa_main_fonts_prop
I2I42       { ipa~ #1 }
I2I43       { \str_use:N \l_tmpa_str = { ##1 } }
I2I44     \group_end:
I2I45   },
I2I46   ipa~ #1~ features
I2I47   .code:n = {
I2I48     \group_begin:
I2I49     \str_clear:N \l_tmpa_str
I2I50     \str_clear:N \l_tmpb_str
I2I51     \str_set:Nn \l_tmpa_str {
I2I52       \text_titlecase_all:n { #1 }
I2I53       Features
I2I54     }
I2I55     \str_set:Nn \l_tmpb_str { #1 }
I2I56     \str_replace_all:Nnn \l_tmpa_str { ~ } { }
I2I57

```



```

1262 \str_replace_all:Nnn \l_tmpb_str { ~ } { _ }
1263 \tl_gput_right:ce {
1264   g__lngx_ipa_main_features_ \l_tmpb_str _tl
1265 } {
1266   \tl_if_empty:cF {
1267     g__lngx_ipa_main_features_ \l_tmpb_str _tl
1268   } { , }
1269   \exp_not:n { ##1 }
1270 }
1271 \prop_gput:Nne \g__lngx_ipa_main_fonts_prop
1272   { ipa~ #1~ features } {
1273   \str_use:N \l_tmpa_str = {
1274     \exp_not:v {
1275       g__lngx_ipa_main_features_ \l_tmpb_str _tl
1276     }
1277   }
1278 }
1279 \group_end:
1280 }
1281 }
1282 }

```

(End of definition for `\g__lngx_ipa_main_fonts_prop` and others. These functions are documented on page [II.](#))

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

```

1283
1284 \tl_gclear_new:N \g__lngx_ipa_main_features_extra_tl
1285
1286 \keys_define:nn { lngx_keys } {
1287   ipa~ main~ extra~ features
1288   .code:n = {
1289     \tl_gput_right:Nn \g__lngx_ipa_main_features_extra_tl {
1290       \tl_if_empty:NF \g__lngx_ipa_main_features_extra_tl {
1291         ,
1292       }
1293       \exp_not:n { #1 }
1294     }
1295     \prop_gput:Nne \g__lngx_ipa_main_fonts_prop
1296       { ipa~ main~ extra~ features } {
1297       \exp_not:V \g__lngx_ipa_main_features_extra_tl
1298     }
1299   }
1300 }

```

(End of definition for `ipa main extra features`. This function is documented on page [13.](#))

```

\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 extra features
    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 extra 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.

```

I301
I302 \prop_gclear_new:N \g__lngx_ipa_sans_fonts_prop
I303 \tl_gclear_new:N \g__lngx_ipa_sans_font_features_tl
I304
I305 \prop_gclear_new:N \g__lngx_ipa_mono_fonts_prop
I306 \tl_gclear_new:N \g__lngx_ipa_mono_font_features_tl
I307
I308 \clist_map_inline:nn {
I309   sans,
I310   mono
I311 } {
I312   \tl_gclear_new:c {
I313     g__lngx_ipa_ #1 _features_extra_tl
I314   }
I315   \clist_map_inline:nn {
I316     upright,
I317     bold~ upright,
I318     italic,
I319     bold~ italic,
I320     slanted,
I321     bold~ slanted,
I322     swash,
I323     bold~ swash,
I324     small~ caps
I325   } {
I326     \group_begin:
I327     \str_clear:N \l_tmpa_str
I328     \str_set:Nn \l_tmpa_str { ##1 }
I329     \str_replace_all:Nnn \l_tmpa_str { ~ } { _ }
I330     \tl_gclear_new:c {
I331       g__lngx_ipa_ #1 _features_ \l_tmpa_str _tl
I332     }
I333     \group_end:
I334     \keys_define:nn { lngx_keys } {
I335       ipa~ #1~ ##1
I336       .code:n = {
I337         \group_begin:
I338         \str_clear:N \l_tmpa_str
I339         \str_set:Ne \l_tmpa_str {
I340           \text_titlecase_all:n { ##1 }
I341           Font
I342         }
I343         \str_replace_all:Nnn \l_tmpa_str { ~ } { }
I344         \prop_gput:cne { g__lngx_ipa_ #1 _fonts_prop }
I345           { ipa~ #1~ ##1 } {
I346           \str_use:N \l_tmpa_str = {
I347             \exp_not:n { #####1 }
I348           }
I349         }
I350       \group_end:
I351     },
I352     ipa~ #1~ ##1~ features

```

```

1353 .code:n = {
1354   \group_begin:
1355   \str_clear:N \l_tmpa_str
1356   \str_clear:N \l_tmpb_str
1357   \str_set:Ne \l_tmpa_str {
1358     \ipa_titlecase_all:n { ##1 }
1359     Features
1360   }
1361   \str_set:Nn \l_tmpb_str { ##1 }
1362   \str_replace_all:Nnn \l_tmpa_str { ~ } { }
1363   \str_replace_all:Nnn \l_tmpb_str { ~ } { _ }
1364   \tl_gput_right:ce {
1365     g__lngx_ipa_ #1 _features_ \l_tmpb_str _tl
1366   } {
1367     \tl_if_empty:cF {
1368       g__lngx_ipa_ #1 _features_ \l_tmpb_str _tl
1369     } { , }
1370     \exp_not:n { #####1 }
1371   }
1372   \prop_gput:cne { g__lngx_ipa_ #1 _fonts_prop }
1373     { ipa~ #1~ ##1~ features } {
1374     \str_use:N \l_tmpa_str = {
1375       \exp_not:v {
1376         g__lngx_ipa_ #1 _features_ \l_tmpb_str _tl
1377       }
1378     }
1379   }
1380   \group_end:
1381 }
1382 }
1383 }
1384 \keys_define:nn { lngx_keys } {
1385   ipa~ #1~ extra~ features
1386   .code:n = {
1387     \tl_gput_right:ce {
1388       g__lngx_ipa_ #1 _features_extra_tl
1389     } {
1390       \tl_if_empty:cF {
1391         g__lngx_ipa_ #1 _features_extra_tl
1392       } { , }
1393       \exp_not:n { ##1 }
1394     }
1395     \prop_gput:cne { g__lngx_ipa_ #1 _fonts_prop }
1396       { ipa~ #1~ extra~ features } {
1397       \exp_not:v { g__lngx_ipa_ #1 _features_extra_tl }
1398     }
1399   }
1400 }
1401 }

```

(End of definition for `\g__lngx_ipa_sans_fonts_prop` and others. These functions are documented on page 13.)

```

\g__lngx_ipa_main_font_tl
\g__lngx_ipa_sans_font_tl
\g__lngx_ipa_mono_font_tl
ipa main font
ipa sans font
ipa mono font

```

These keys provide keys to set fonts for IPA.

```

I402
I403 \clist_map_inline:nn {
I404   main,
I405   sans,
I406   mono
I407 } {
I408   \keys_define:nn { lngx_keys } {
I409     ipa~ #1~ font
I410     .tl_gset:c          = { g__lngx_ipa_ #1 _font_tl }
I411   }
I412 }

```

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

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.

```

I413
I414 \keys_define:nn { lngx_keys } {
I415   ipa~ newcm
I416   .meta:n          = {
I417     ipa~ main~ extra~
I418     features        = {
I419       Renderer      = {HarfBuzz},
I420       StylisticSet  = {05}
I421     },
I422     ipa~ sans~ extra~
I423     features        = {
I424       Renderer      = {HarfBuzz},
I425       StylisticSet  = {05}
I426     },
I427     ipa~ mono~ extra~
I428     features        = {
I429       Renderer      = {HarfBuzz},
I430       StylisticSet  = {05}
I431     },
I432     ipa~ main~ font  = { NewCM10-Book.otf },
I433     ipa~ sans~ font  = { NewCMSans10-Book.otf },
I434     ipa~ mono~ font  = { NewCMMono10-Book.otf }
I435   }
I436 }

```

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

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

```

I437
I438 \keys_define:nn { lngx_keys } {
I439   ipa~ newcm~ sans
I440   .meta:n          = {
I441     ipa~ main~ extra~
I442     features        = {
I443       Renderer      = {HarfBuzz},
I444       StylisticSet  = {05}

```

```

1445     },
1446     ipa~ sans~ extra~
1447     features                = {
1448         Renderer            = {HarfBuzz},
1449         StylisticSet        = {05}
1450     },
1451     ipa~ mono~ extra~
1452     features                = {
1453         Renderer            = {HarfBuzz},
1454         StylisticSet        = {05}
1455     },
1456     ipa~ main~ font         = { NewCMSans10-Book.otf },
1457     ipa~ sans~ font         = { NewCMSans10-Book.otf },
1458     ipa~ mono~ font         = { NewCMMono10-Book.otf }
1459 }
1460 }

```

(End of definition for *ipa newcm sans*. This function is documented on page [II](#).)

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

```

1461 \keys_define:nn { lngx_keys } {
1462   ipa~ newcm~ mono
1463   .meta:n                = {
1464     ipa~ main~ extra~
1465     features              = {
1466         Renderer          = {HarfBuzz},
1467         StylisticSet      = {05}
1468     },
1469     ipa~ sans~ extra~
1470     features              = {
1471         Renderer          = {HarfBuzz},
1472         StylisticSet      = {05}
1473     },
1474     ipa~ mono~ extra~
1475     features              = {
1476         Renderer          = {HarfBuzz},
1477         StylisticSet      = {05}
1478     },
1479     ipa~ main~ font       = { NewCMMono10-Book.otf },
1480     ipa~ sans~ font       = { NewCMSans10-Book.otf },
1481     ipa~ mono~ font       = { NewCMMono10-Book.otf }
1482   }
1483 }
1484 }

```

(End of definition for *ipa newcm mono*. This function is documented on page [II](#).)

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

```

1485 \keys_define:nn { lngx_keys } {
1486   ipa~ newcm~ regular
1487   .meta:n                = {
1488     ipa~ main~ extra~

```

```

1490     features                = {
1491         Renderer              = {HarfBuzz},
1492         StylisticSet          = {05}
1493     },
1494     ipa~ sans~ extra~
1495     features                = {
1496         Renderer              = {HarfBuzz},
1497         StylisticSet          = {05}
1498     },
1499     ipa~ mono~ extra~
1500     features                = {
1501         Renderer              = {HarfBuzz},
1502         StylisticSet          = {05}
1503     },
1504     ipa~ main~ font          = { NewCM10-Regular.otf },
1505     ipa~ sans~ font          = { NewCMSans10-Regular.otf },
1506     ipa~ mono~ font          = { NewCMMono10-Regular.otf }
1507 }
1508 }

```

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

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.

```

1509 \keys_define:nn { lngx_keys } {
1510     ipa~ newcm~ regular~ sans
1511     .meta:n                = {
1512         ipa~ main~ extra~
1513         features            = {
1514             Renderer        = {HarfBuzz},
1515             StylisticSet     = {05}
1516         },
1517         ipa~ sans~ extra~
1518         features            = {
1519             Renderer        = {HarfBuzz},
1520             StylisticSet     = {05}
1521         },
1522         ipa~ mono~ extra~
1523         features            = {
1524             Renderer        = {HarfBuzz},
1525             StylisticSet     = {05}
1526         },
1527         ipa~ main~ font      = { NewCMSans10-Regular.otf },
1528         ipa~ sans~ font      = { NewCMSans10-Regular.otf },
1529         ipa~ mono~ font      = { NewCMMono10-Regular.otf }
1530     }
1531 }
1532 }

```

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

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.

```

1533

```

```

1534 \keys_define:nn { lngx_keys } {
1535   ipa~ newcm~ regular~ mono
1536   .meta:n          = {
1537     ipa~ main~ extra~
1538     features        = {
1539       Renderer      = {HarfBuzz},
1540       StylisticSet  = {05}
1541     },
1542     ipa~ sans~ extra~
1543     features        = {
1544       Renderer      = {HarfBuzz},
1545       StylisticSet  = {05}
1546     },
1547     ipa~ mono~ extra~
1548     features        = {
1549       Renderer      = {HarfBuzz},
1550       StylisticSet  = {05}
1551     },
1552     ipa~ main~ font  = { NewCMMono10-Regular.otf },
1553     ipa~ sans~ font  = { NewCMSans10-Regular.otf },
1554     ipa~ mono~ font  = { NewCMMono10-Regular.otf }
1555   }
1556 }

```

(End of definition for *ipa newcm regular mono*. This function is documented on page 11.)

We set the `ipa newcm` key by default.

```

1557
1558 \lngx_set_keys:n {ipa~ newcm}

```

`\lngx_set_main_ipa_font:nn` 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.

`\lngx_main_ipa:` 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)

`lngx_ipa_rm_nfss` at several other places, so use them with caution.

`\lngx_set_sans_ipa_font:nn`

`\lngx_sans_ipa:`

`lngx_ipa_sf_nfss`

```

1559
1560 \cs_new_protected:Npn \lngx_set_main_ipa_font:nn #1#2 {
1561   \setfontfamily \lngx_main_ipa: [
1562     #1,
1563     NFSSFamily      = { lngx_ipa_rm_nfss }
1564   ] { #2 }
1565 }
1566
1567 \cs_new_protected:Npn \lngx_set_sans_ipa_font:nn #1#2 {
1568   \setfontfamily \lngx_sans_ipa: [
1569     #1,
1570     NFSSFamily      = { lngx_ipa_sf_nfss }
1571   ] { #2 }
1572 }
1573
1574 \cs_new_protected:Npn \lngx_set_mono_ipa_font:nn #1#2 {
1575   \setfontfamily \lngx_mono_ipa: [
1576     #1,
1577     NFSSFamily      = { lngx_ipa_tt_nfss }
1578   ] { #2 }

```

```

1579 }
1580
1581 \cs_generate_variant:Nn \lngx_set_main_ipa_font:nn { VV }
1582 \cs_generate_variant:Nn \lngx_set_sans_ipa_font:nn { VV }
1583 \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 21.)

lngx_ipa Here, I create a ‘super font family’ with `\lngx_super_font_family:nn`, a macro provided by LINGUIS~~TI~~X-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.

```

1584
1585 \lngx_super_font_family:nn { lngx_ipa } {
1586   rm          = { lngx_ipa_rm_nfss },
1587   sf          = { lngx_ipa_sf_nfss },
1588   tt          = { lngx_ipa_tt_nfss }
1589 }

```

(End of definition for `lngx_ipa`. This function is documented on page 21.)

\lngxipa I use `\lngx_softer_super_font_family:n` provided by LINGUIS~~TI~~X-NFSS for defining this
\lngx_ipa: switch to the IPA.

```

1590
1591 \cs_new_protected:Npn \lngx_ipa: {
1592   \lngx_softer_super_font_family:n { lngx_ipa }
1593 }
1594
1595 \cs_gset_eq:NN \lngxipa \lngx_ipa:

```

(End of definition for `\lngxipa` and `\lngx_ipa:`. These functions are documented on page 11.)

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

```

1596
1597 \clist_map_inline:nn {
1598   main,
1599   sans,
1600   mono
1601 } {
1602   \cs_new_protected:cpn {
1603     lngx_build_#1_ipa_font_features:
1604   } {
1605     \prop_map_inline:cn { g__lngx_ipa_#1_fonts_prop } {
1606       \tl_gput_right:cn {
1607         g__lngx_ipa_#1_font_features_tl
1608       } { ####2 }
1609     }
1610   }
1611 }
1612
1613 \hook_gput_code:nnn { begindocument / before } { . } {
1614   \lngx_build_main_ipa_font_features:
1615   \lngx_set_main_ipa_font:VV

```



```

1616     \g__lngx_ipa_main_font_features_tl
1617     \g__lngx_ipa_main_font_tl
1618   \lngx_build_sans_ipa_font_features:
1619   \lngx_set_sans_ipa_font:VV
1620     \g__lngx_ipa_sans_font_features_tl
1621     \g__lngx_ipa_sans_font_tl
1622   \lngx_build_mono_ipa_font_features:
1623   \lngx_set_mono_ipa_font:VV
1624     \g__lngx_ipa_mono_font_features_tl
1625     \g__lngx_ipa_mono_font_tl
1626   }
1627 </ipa>

```

```

r628 (*lang)
r629 \ProvidesExplPackage{linguistix-languages}
r630 {2026-04-27}
r631 {v0.9b}
r632 {%
r633 An assistant package for automatically
r634 loading fonts and more settings for
r635 languages.%
r636 }

```

LINGUISTIX-BASE is loaded (if not already done) for the key-value parser.

```

r637
r638 \IfPackageLoadedF { linguistix-base } {
r639 \RequirePackage { linguistix-base }
r640 }

```

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

```

r641
r642 \IfPackageLoadedF { babel } {
r643 \RequirePackage [ provide * = * ] { babel }
r644 }

```

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

```

r645
r646 \tl_new:N \g_lngx_main_language_tl

```

(End of definition for `\g_lngx_main_language_tl`. This function is documented on page 21.)

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

```

r647
r648 \clist_new:N \g_lngx_languages_clist

```

(End of definition for `\g_lngx_languages_clist`. This function is documented on page 21.)

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

`\providelanguage`

```

r649
r650 \cs_new_protected:Npn \lngx_languages:nn #1#2 {
r651 \babelprovide [ #1 ] { #2 }
r652 }
r653
r654 \cs_generate_variant:Nn \lngx_languages:nn { VV }
r655 \cs_gset_eq:NN \providelanguage \lngx_languages:nn

```

(End of definition for `\lngx_languages:nn` and `\providelanguage`. These functions are documented on page 21.)

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.

```

r656
r657 \IfPackageLoadedTF { linguistix-fonts } {
r658   \cs_gset_protected:Npn \lngx_set_main_font:nn #1#2 {
r659     \bafont { rm } [ #1 ] { #2 }
r660   }
r661   \cs_gset_protected:Npn \lngx_set_sans_font:nn #1#2 {
r662     \bafont { sf } [ #1 ] { #2 }
r663   }
r664   \cs_gset_protected:Npn \lngx_set_mono_font:nn #1#2 {
r665     \bafont { tt } [ #1 ] { #2 }
r666   }
r667 } {
r668   \cs_new_protected:Npn \lngx_set_main_font:nn #1#2 {
r669     \bafont { rm } [ #1 ] { #2 }
r670   }
r671   \cs_new_protected:Npn \lngx_set_sans_font:nn #1#2 {
r672     \bafont { sf } [ #1 ] { #2 }
r673   }
r674   \cs_new_protected:Npn \lngx_set_mono_font:nn #1#2 {
r675     \bafont { tt } [ #1 ] { #2 }
r676   }
r677 }

```

\lngx_other_main_font:nnn The following macros set fonts for other languages using the `\bafont` command.

\lngx_other_sans_font:nnn

\lngx_other_mono_font:nnn

```

r678
r679 \cs_gset_protected:Npn \lngx_other_main_font:nnn #1#2#3 {
r680   \bafont [ #1 ] { rm } [ #2 ] { #3 }
r681 }
r682
r683 \cs_gset_protected:Npn \lngx_other_sans_font:nnn #1#2#3 {
r684   \bafont [ #1 ] { sf } [ #2 ] { #3 }
r685 }
r686
r687 \cs_gset_protected:Npn \lngx_other_mono_font:nnn #1#2#3 {
r688   \bafont [ #1 ] { tt } [ #2 ] { #3 }
r689 }
r690
r691 \cs_generate_variant:Nn \lngx_other_main_font:nnn { nee }
r692 \cs_generate_variant:Nn \lngx_other_sans_font:nnn { nee }
r693 \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 20.)

\lngx_load_languages:n I provide a simple macro that only does the job of loading languages, both in L^AT_EX₃
\loadlanguages style, as well as the in the plain style.

```

r694
r695 \cs_new_protected:Npn \lngx_load_languages:n #1 {
r696   \lngx_set_keys:n { languages = { #1 } }
r697 }
r698
r699 \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 21.)

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

`\lngx_counter:n`

```

1700
1701 \cs_gset_eq:NN \lngx_counter:n \arabic

```

(End of definition for `\lngx_counter:n`. This function is documented on page 21.)

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

```

1702
1703 \cs_set:Npn \thechapter {
1704   \lngx_counter:n { chapter }
1705 }
1706 \cs_set:Npn \thesection {
1707   \lngx_counter:n { section }
1708 }
1709 \cs_set:Npn \thesubsection {
1710   \lngx_counter:n { subsection }
1711 }
1712 \cs_set:Npn \thesubsubsection {
1713   \lngx_counter:n { subsubsection }
1714 }
1715 \cs_set:Npn \theparagraph {
1716   \lngx_counter:n { section }
1717 }
1718 \cs_set:Npn \thesubparagraph {
1719   \lngx_counter:n { section }
1720 }
1721 \cs_set:Npn \thepage {
1722   \lngx_counter:n { page }
1723 }
1724 \cs_set:Npn \thefigure {
1725   \lngx_counter:n { figure }
1726 }
1727 \cs_set:Npn \thetable {
1728   \lngx_counter:n { table }
1729 }
1730 \cs_set:Npn \thefootnote {
1731   \lngx_counter:n { footnote }
1732 }
1733 \cs_set:Npn \thempfootnote {
1734   \lngx_counter:n { mpfootnote }
1735 }
1736 \cs_set:Npn \theequation {
1737   \lngx_counter:n { equation }
1738 }

```

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

```

1739
1740 \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).

```

1741
1742 \socket_new_plug:nnn { lngx / native-numbering }
1743         { strict } {
1744     \cs_gset_eq:Nc \lngx_counter:n {
1745         \tl_use:N \g_lngx_main_language_tl counter
1746     }
1747 }

```

(End of definition for *strict*. This function is documented on page 15.)

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 T_EX-logically).

```

1748
1749 \socket_new_plug:nnn { lngx / native-numbering }
1750         { logical } {
1751     \cs_gset_protected:Npn \lngx_counter:n ##1 {
1752         \localecounter { digits } { ##1 }
1753     }
1754 }

```

(End of definition for *logical*. This function is documented on page 15.)

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.

```

1755
1756 \socket_new_plug:nnn { lngx / native-numbering } { off } {
1757     \cs_gset_eq:NN \lngx_counter:n \arabic
1758 }

```

(End of definition for *off*. This function is documented on page 15.)

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.

```

1759
1760 \cs_generate_variant:Nn \socket_assign_plug:nn { ne }
1761
1762 \keys_define:nn { lngx_keys } {
1763     native~ numbering
1764     .choices:nn      = { strict,logical,off } {
1765         \socket_assign_plug:ne { lngx / native-numbering } {
1766             \str_use:N \l_keys_choice_str
1767         }
1768         \socket_use:n { lngx / native-numbering }
1769     },

```

Similarly, we set the default value to `on`.

```

1770     native~ numbering
1771     .default:n       = { strict }
1772 }

```

(End of definition for *native numbering*. This function is documented on page 15.)

`\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.

```

1773
1774 \cs_new_protected:Npn \lngx_misc_reset: {
1775   \cs_set:Npn \theenumii { \alph { enumii } }
1776   \cs_set:Npn \labelenumii { ( \theenumii ) }
1777   \cs_set:Npn \theenumiii { \roman { enumiii } }
1778   \cs_set:Npn \labelenumiii { \theenumiii . }
1779   \cs_set:Npn \theenumiv { \Alph { enumiv } }
1780   \cs_set:Npn \labelenumiv { \theenumiv . }
1781   \IfPackageLoadedT { expex } {
1782     \lingset { labeltype = alpha }
1783   }
1784   \cs_gset_eq:NN \emph \textit
1785 }

```

(End of definition for `\lngx_misc_reset:`. This function is documented on page 21.)

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

```

1786
1787 \msg_new:nnn { linguistix-languages } { no_support } {
1788   '#1'~ is~ not~ supported.\\
1789   If~ you~ want~ it~ to~ be~ supported,~ please~ report~
1790   to~ the~ maintainers.
1791 }

```

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

```

1792
1793 \keys_define:nn { lngx_keys } {
1794   languages
1795   .code:n          = {

```

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

```

1796   \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.

```

1797   \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.

```

1798   \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.

```

1799   \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.

```

1800   \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 `LuaATeX`, 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.

```

1801      \tl_set:Nc \l_tmpb_tl {
1802          main,

```

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

```

1803      import
1804  }

```

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

```

1805      \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.

```

1806      \file_if_exist:nTF { linguistix - \l_tmpa_tl . sty } {
1807          \exp_args:Nc \RequirePackage
1808              { linguistix - \l_tmpa_tl }
1809      } {

```

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.

```

1810          \msg_warning:nnV { linguistix-languages }
1811                          { no_support }
1812                          \l_tmpa_tl
1813      }

```

The temporary `tls` are cleared.

```

1814      \tl_clear:N \l_tmpa_tl
1815      \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.

```

1816      \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`.

```

1817      \clist_map_inline:Nn \l_tmpa_clist {
1818          \clist_pop:NN \l_tmpa_clist \l_tmpa_tl
1819          \tl_set:Nc \l_tmpb_tl { import }
1820          \lngx_languages:VV \l_tmpb_tl \l_tmpa_tl
1821          \file_if_exist:nTF {
1822              linguistix - \l_tmpa_tl . sty
1823          } {
1824              \exp_args:Nc \RequirePackage
1825                  { linguistix - \l_tmpa_tl }
1826          } {
1827              \msg_warning:nnV { linguistix-languages }
1828                              { no_ldf }
1829                              \l_tmpa_tl
1830          }
1831          \tl_clear:N \l_tmpa_tl
1832          \tl_clear:N \l_tmpb_tl
1833      }
1834  }

```

```
1835     }  
1836   }  
1837 }  
1838 </lang>
```

(End of definition for `languages`. This function is documented on page [I5](#).)


```

1839 <*logos>
1840 \ProvidesExplPackage{linguistix-logos}
1841         {2026-04-27}
1842         {v0.9b}
1843         {%
1844         Logos of the ‘LinguisTiX’ bundle.%
1845         }

```

The fontspec package (if not already loaded).

```

1846
1847 \IfPackageLoadedF { fontspec } {
1848     \RequirePackage { fontspec }
1849 }

```

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

```

1850
1851 \newfontfamily \lngx_logo_font: [
1852     UprightFont          = { NewCMUncial10-Book.otf },
1853     UprightFeatures      = {
1854         SizeFeatures      = {
1855             {
1856                 Size          = {-8},
1857                 Font          = {NewCMUncial08-Book.otf}
1858             },
1859             {
1860                 Size          = {8-},
1861                 Font          = {NewCMUncial10-Book.otf}
1862             },
1863         }
1864     },
1865     BoldFont             = { NewCMUncial10-Bold.otf },
1866     BoldFeatures         = {
1867         SizeFeatures      = {
1868             {
1869                 Size          = {-8},
1870                 Font          = {NewCMUncial08-Bold.otf}
1871             },
1872             {
1873                 Size          = {8-},
1874                 Font          = {NewCMUncial10-Bold.otf}
1875             },
1876         }
1877     }
1878 ]{ NewCMUncial10-Book.otf }

```

(End of definition for \lngx_logo_font:. This function is documented on page 22.)

lngx_purple_color The following defines the lngx_purple_color.

```

1879
1880 \color_set:nn { lngx_purple_color } { blue ! 50 ! red }

```

(End of definition for lngx_purple_color. This function is documented on page 22.)

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

```

1881
1882 \NewDocumentCommand \lngxlogo { 0{} } {%
1883   \group_begin:
1884   \lngx_logo_font:
1885   LinguisTi
1886   \color_group_begin:
1887   \color_select:n { lngx_purple_color }
1888   X
1889   \color_group_end:
1890   \IfBlankF { #1 } { - #1 }
1891   \group_end:
1892 }

```

(End of definition for `\lngxlogo`. This function is documented on page 16.)

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

```

1893
1894 \cs_new:Npn \lngxpkg {
1895   \IfPackageLoadedTF { hyperref } {
1896     \texorpdfstring {
1897       \lngxlogo
1898     } {
1899       LinguisTiX
1900     }
1901   } {
1902     \lngxlogo
1903   }
1904 }

```

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

```

1905
1906 \clist_map_inline:nn {
1907   base,examples,fixpex,fonts,ipa,languages,logos,nfss,
1908   marathi,british,american,english,greek,malayalam,glossing,
1909   leipzig
1910 } {

```

`#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.

```

1911   \cs_new:cpn { lngx #1 logo } {
1912     \texorpdfstring {
1913       \lngxlogo [ #1 ]
1914     } {
1915       LinguisTiX - #1
1916     }
1917   }
1918 }
1919 </logos>

```

LINGUIS*Ti*X-NFSS

Documentation | L^AT_EX₃-interface

```

1920 < *nfss >

```

```

1921 \ProvidesExplPackage{linguistix-nfss}
1922     {2026-04-27}
1923     {v0.9b}
1924     {%
1925         An extension to the core NFSS commands
1926         from the ‘LinguisTiX’ bundle.%
1927     }

```

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

```

1928
1929 \tl_new:N \l__lngx_normalfont_tmp_tl
1930 \tl_new:N \l__lngx_selectfont_tmp_tl
1931 \tl_new:N \l__lngx_family_tmp_tl
1932 \tl_new:N \l__lngx_nfss_tmp_tl

```

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

```

1933
1934 \tl_new:N \l_lngx_current_encoding_tl
1935 \tl_new:N \l_lngx_current_meta_family_tl
1936 \tl_new:N \l_lngx_current_super_family_tl
1937 \tl_new:N \l_lngx_current_series_tl
1938 \tl_new:N \l_lngx_current_shape_tl

```

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

```

1939
1940 \hook_gput_code:nnn { begindocument / end } { . } {
1941     \tl_const:Ne \c_lngx_default_rmdefault_tl { \rmdefault }
1942     \tl_const:Ne \c_lngx_default_sfdefault_tl { \sfdefault }
1943     \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 22.)

`\l_lngx_current_encoding_tl` First, I set the value `default` for the initial super font family.
`\l_lngx_current_meta_family_tl` `\tl_set:Nn \l_lngx_current_super_family_tl { default }`
`\l_lngx_current_super_family_tl`
`\l_lngx_current_series_tl` The current encoding is saved in the relevant t_l.
`\l_lngx_current_shape_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.

```

1948 \tl_set:Ne \l_lngx_current_meta_family_tl {
1949     \@currentmetafamily % new from ltnews42, thanks Frank!
1950 }

```

Here, the series and shape t_ls are set to their defaults.

```

1951 \tl_set:Nn \l_lngx_current_series_tl { md }
1952 \tl_set:Nn \l_lngx_current_shape_tl { up }
1953 }

```

(End of definition for `\l_lngx_current_encoding_tl` and others. These functions are documented on page 22.)

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

```

1954
1955 \hook_gput_code:nnn { cmd / selectfont / before } { . } {
1956   \tl_set:Ne \l__lngx_selectfont_tmp_tl { \f@encoding }
1957 }

```

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`.

```

1958
1959 \hook_gput_code:nnn { cmd / selectfont / after } { . } {
1960   \tl_set_eq:NN \l_lngx_current_encoding_tl
1961               \l__lngx_selectfont_tmp_tl
1962   \tl_clear:N   \l__lngx_selectfont_tmp_tl
1963 }

```

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.

```

1964
1965 \hook_gput_code:nnn { cmd / rmfamily / before } { . } {
1966   \tl_set:Nn \l_lngx_current_meta_family_tl { rm }
1967   \tl_set:Ne \l__lngx_family_tmp_tl { \f@encoding }
1968 }
1969
1970 \hook_gput_code:nnn { cmd / rmfamily / after } { . } {
1971   \tl_set:Nn \l_lngx_current_meta_family_tl { rm }
1972   \tl_set_eq:NN \l_lngx_current_encoding_tl
1973               \l__lngx_family_tmp_tl
1974   \tl_clear:N   \l__lngx_family_tmp_tl
1975 }
1976
1977 \hook_gput_code:nnn { cmd / sffamily / before } { . } {
1978   \tl_set:Nn \l_lngx_current_meta_family_tl { sf }
1979   \tl_set:Ne \l__lngx_family_tmp_tl { \f@encoding }
1980 }
1981
1982 \hook_gput_code:nnn { cmd / sffamily / after } { . } {
1983   \tl_set:Nn \l_lngx_current_meta_family_tl { sf }
1984   \tl_set_eq:NN \l_lngx_current_encoding_tl
1985               \l__lngx_family_tmp_tl
1986   \tl_clear:N   \l__lngx_family_tmp_tl
1987 }
1988
1989 \hook_gput_code:nnn { cmd / ttfamily / before } { . } {
1990   \tl_set:Nn \l_lngx_current_meta_family_tl { tt }
1991   \tl_set:Ne \l__lngx_family_tmp_tl { \f@encoding }
1992 }
1993
1994 \hook_gput_code:nnn { cmd / ttfamily / after } { . } {

```

```

1995 \tl_set:Nn \l_lngx_current_meta_family_tl { tt }
1996 \tl_set_eq:NN \l_lngx_current_encoding_tl
1997         \l__lngx_family_tmp_tl
1998 \tl_clear:N \l__lngx_family_tmp_tl
1999 }

```

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

```

2000
2001 \hook_gput_code:nnn { cmd / mdseries / after } { . } {
2002     \tl_set:Nn \l_lngx_current_series_tl { md }
2003 }
2004
2005 \hook_gput_code:nnn { cmd / bfseries / after } { . } {
2006     \tl_set:Nn \l_lngx_current_series_tl { bf }
2007 }

```

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

```

2008
2009 \hook_gput_code:nnn { cmd / upshape / after } { . } {
2010     \tl_set:Nn \l_lngx_current_shape_tl { up }
2011 }
2012
2013 \hook_gput_code:nnn { cmd / itshape / after } { . } {
2014     \tl_set:Nn \l_lngx_current_shape_tl { it }
2015 }
2016
2017 \hook_gput_code:nnn { cmd / scshape / after } { . } {
2018     \tl_set:Nn \l_lngx_current_shape_tl { sc }
2019 }
2020
2021 \hook_gput_code:nnn { cmd / sscshape / after } { . } {
2022     \tl_set:Nn \l_lngx_current_shape_tl { ssc }
2023 }
2024
2025 \hook_gput_code:nnn { cmd / slshape / after } { . } {
2026     \tl_set:Nn \l_lngx_current_shape_tl { sl }
2027 }
2028
2029 \hook_gput_code:nnn { cmd / swshape / after } { . } {
2030     \tl_set:Nn \l_lngx_current_shape_tl { sw }
2031 }
2032
2033 \hook_gput_code:nnn { cmd / ulcshape / after } { . } {
2034     \tl_set:Nn \l_lngx_current_shape_tl { ulc }
2035 }

```

`\lngx_if_encoding_p:n` I provide a conditional for checking the current encoding with the given argument.

`\lngx_if_encoding:nTF`

```

2036
2037 \prg_new_conditional:Nnn \lngx_if_encoding:n {
2038     p,
2039     T,

```

```

2040 F,
2041 TF
2042 } {
2043 \tl_if_eq:NnTF \l_lngx_current_encoding_tl { #1 } {
2044 \prg_return_true:
2045 } {
2046 \prg_return_false:
2047 }
2048 }
2049

```

(End of definition for `\lngx_if_encoding:nTF`. This function is documented on page 22.)

`\IfEncodingTF` For non- \LaTeX contexts, these simpler alternatives are provided.
`\IfEncodingT`
`\IfEncodingF`

```

2050
2051 \cs_new_eq:NN \IfEncodingTF \lngx_if_encoding:nTF
2052 \cs_new_eq:NN \IfEncodingT \lngx_if_encoding:nT
2053 \cs_new_eq:NN \IfEncodingF \lngx_if_encoding:nF

```

(End of definition for `\IfEncodingTF`, `\IfEncodingT`, and `\IfEncodingF`. These functions are documented on page 18.)

`\lngx_if_meta_family_p:n` A conditional for checking the meta family with the given argument.
`\lngx_if_meta_family:nTF`

```

2054
2055 \prg_new_conditional:Nnn \lngx_if_meta_family:n {
2056 P,
2057 T,
2058 F,
2059 TF
2060 } {
2061 \tl_if_eq:NnTF \l_lngx_current_meta_family_tl { #1 } {
2062 \prg_return_true:
2063 } {
2064 \prg_return_false:
2065 }
2066 }

```

(End of definition for `\lngx_if_meta_family:nTF`. This function is documented on page 22.)

`\IfMetaFamilyTF` User-facing conditionals for meta family.
`\IfMetaFamilyT`
`\IfMetaFamilyF`

```

2067
2068 \cs_new_eq:NN \IfMetaFamilyTF \lngx_if_meta_family:nTF
2069 \cs_new_eq:NN \IfMetaFamilyT \lngx_if_meta_family:nT
2070 \cs_new_eq:NN \IfMetaFamilyF \lngx_if_meta_family:nF

```

(End of definition for `\IfMetaFamilyTF`, `\IfMetaFamilyT`, and `\IfMetaFamilyF`. These functions are documented on page 18.)

`\lngx_if_super_family_p:n` A conditional for checking the super family with the given argument.
`\lngx_if_super_family:nTF`

```

2071
2072 \prg_new_conditional:Nnn \lngx_if_super_family:n {
2073 P,
2074 T,
2075 F,
2076 TF

```

```

2077 } {
2078   \tl_if_eq:NnTF \l_lngx_current_super_family_tl { #1 } {
2079     \prg_return_true:
2080   } {
2081     \prg_return_false:
2082   }
2083 }

```

(End of definition for `\lngx_if_super_family:nTF`. This function is documented on page 22.)

`\IfSuperFamilyTF` User-facing conditionals for super family.

`\IfSuperFamilyT`
`\IfSuperFamilyF`

```

2084
2085 \cs_new_eq:NN \IfSuperFamilyTF \lngx_if_super_family:nTF
2086 \cs_new_eq:NN \IfSuperFamilyT \lngx_if_super_family:nT
2087 \cs_new_eq:NN \IfSuperFamilyF \lngx_if_super_family:nF

```

(End of definition for `\IfSuperFamilyTF`, `\IfSuperFamilyT`, and `\IfSuperFamilyF`. These functions are documented on page 18.)

`\lngx_if_series_p:n` A conditional for checking the current series with the given argument.

`\lngx_if_series:nTF`

```

2088
2089 \prg_new_conditional:Nnn \lngx_if_series:n {
2090   P,
2091   T,
2092   F,
2093   TF
2094 } {
2095   \tl_if_eq:NnTF \l_lngx_current_series_tl { #1 } {
2096     \prg_return_true:
2097   } {
2098     \prg_return_false:
2099   }
2100 }

```

(End of definition for `\lngx_if_series:nTF`. This function is documented on page 22.)

`\IfSeriesTF` Its user-side macros.

`\IfSeriesT`
`\IfSeriesF`

```

2101
2102 \cs_new_eq:NN \IfSeriesTF \lngx_if_series:nTF
2103 \cs_new_eq:NN \IfSeriesT \lngx_if_series:nT
2104 \cs_new_eq:NN \IfSeriesF \lngx_if_series:nF

```

(End of definition for `\IfSeriesTF`, `\IfSeriesT`, and `\IfSeriesF`. These functions are documented on page 18.)

`\lngx_if_shape_p:n` A conditional for checking the current shape with the current argument.

`\lngx_if_shape:nTF`

```

2105
2106 \prg_new_conditional:Nnn \lngx_if_shape:n {
2107   P,
2108   T,
2109   F,
2110   TF
2111 } {
2112   \tl_if_eq:NnTF \l_lngx_current_shape_tl { #1 } {
2113     \prg_return_true:

```

```

2114 } {
2115   \prg_return_false:
2116 }
2117 }

```

(End of definition for `\lngx_if_shape:nTF`. This function is documented on page 22.)

\IfShapeTF User-side macros for the same.

```

2118 \IfShapeT
2119 \IfShapeF
2120 \cs_new_eq:NN \IfShapeTF \lngx_if_shape:nTF
2121 \cs_new_eq:NN \IfShapeT \lngx_if_shape:nT
2122 \cs_new_eq:NN \IfShapeF \lngx_if_shape:nF

```

(End of definition for `\IfShapeTF`, `\IfShapeT`, and `\IfShapeF`. These functions are documented on page 18.)

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.

```

2122 \cs_generate_variant:Nn \prg_new_conditional:Nnn { cnn }

```

\lngx_if_meta_family_rm_p: These are separate conditionals for `rm`, `sf` and `tt` families. They don't require arguments.

\lngx_if_meta_family_rm:TF No user side commands are provided for these.

```

2124 \lngx_if_meta_family_sf_p:
2125 \lngx_if_meta_family_sf:TF
2126 \lngx_if_meta_family_tt_p:
2127 \lngx_if_meta_family_tt:TF
2128 \clist_map_inline:nn {
2129   rm,
2130   sf,
2131   tt
2132 } {
2133   \prg_new_conditional:cnn { lngx_if_meta_family_ #1 : } {
2134     p, T, F, TF
2135   } {
2136     \tl_if_eq:NnTF \l_lngx_current_meta_family_tl { #1 } {
2137       \prg_return_true:
2138     } {
2139       \prg_return_false:
2140     }
2141   }

```

(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 22.)

\lngx_if_series_md_p: Separate conditionals for both the series.

```

2140 \lngx_if_series_md:TF
2141 \lngx_if_series_bf_p:
2142 \lngx_if_series_bf:TF
2143 \clist_map_inline:nn {
2144   md,
2145   bf
2146 } {
2147   \prg_new_conditional:cnn { lngx_if_series_ #1 : } {
2148     p, T, F, TF
2149   } {
2150     \tl_if_eq:NnTF \l_lngx_current_series_tl { #1 } {
2151       \prg_return_true:
2152     } {
2153       \prg_return_false:
2154     }
2155   }

```



```

2151     \prg_return_false:
2152   }
2153 }
2154 }

```

(End of definition for `\lngx_if_series_md:TF` and `\lngx_if_series_bf:TF`. These functions are documented on page 23.)

`\lngx_if_shape_up_p:` Separate conditionals for all the shapes.

```

\lngx_if_shape_up_p:TF
\lngx_if_shape_up:TF
\lngx_if_shape_it_p:TF
\lngx_if_shape_it:TF
\lngx_if_shape_sc_p:TF
\lngx_if_shape_sc:TF
\lngx_if_shape_ssc_p:TF
\lngx_if_shape_ssc:TF
\lngx_if_shape_sl_p:TF
\lngx_if_shape_sl:TF
\lngx_if_shape_sw_p:TF
\lngx_if_shape_sw:TF
\lngx_if_shape_ulc_p:TF
\lngx_if_shape_ulc:TF
2155
2156 \clist_map_inline:nn {
2157   up,
2158   it,
2159   sc,
2160   ssc,
2161   sl,
2162   sw,
2163   ulc
2164 } {
2165   \prg_new_conditional:cnn { lngx_if_shape_ #1 : } {
2166     p, T, F, TF
2167   } {
2168     \tl_if_eq:NnTF \l_lngx_current_shape_tl { #1 } {
2169       \prg_return_true:
2170     } {
2171       \prg_return_false:
2172     }
2173   }
2174 }

```

(End of definition for `\lngx_if_shape_up:TF` and others. These functions are documented on page 23.)

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 tls 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.

```

2175
2176 \clist_map_inline:nn {
2177   rm,
2178   sf,
2179   tt
2180 } {
2181   \keys_define:nn { lngx_nfss } {
2182     #1
2183     .code:n = {
2184       \tl_gclear_new:c {
2185         g_lngx_ \l__lngx_nfss_tmp_tl _ #1 default _tl
2186       }
2187       \tl_gset:cn {
2188         g_lngx_ \l__lngx_nfss_tmp_tl _ #1 default _tl
2189       } { ##1 }
2190     }
2191   }
2192 }

```

`\lngx_super_font_family:nn` I first set the temporary `tl` with the name of the super font family retrieved from the first argument.
`\superfontfamily`

```
2193
2194 \cs_new_protected:Npn \lngx_super_font_family:nn #1#2 {
2195   \tl_set:Nx \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.

```
2196   \keys_set:nn { lngx_nfss } { #2 }
2197   \tl_clear:N \l__lngx_nfss_tmp_tl
2198 }
2199
2200 \cs_gset_eq:NN \superfontfamily
2201   \lngx_super_font_family:nn
```

(End of definition for `\lngx_super_font_family:nn` and `\superfontfamily`. These functions are documented on page 23.)

`\lngx_soft_super_font_family:nn` I set the `tl` that saves the current font family to the first argument.
`\softsuperfontfamily`

```
2202
2203 \cs_new_protected:Npn \lngx_soft_super_font_family:nn #1#2 {
2204   \tl_set:Nx \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.

```
2205   \clist_map_inline:nn {
2206     rm,
2207     sf,
2208     tt
2209   } {
2210     \tl_if_empty:cF { g_lngx_ #1 _ ##1 default_tl } {
2211       \cs_set:cpe { ##1 default } {
2212         \tl_use:c { g_lngx_ #1 _ ##1 default _tl }
2213       }
2214     }
2215   }
```

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

```
2216   \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.

```
2217   \clist_map_inline:nn { #2 } {
2218     \str_case:nn { ##1 } {
2219       { encoding } {
2220         \exp_args:NV \fontencoding
2221           \l__lngx_current_encoding_tl
2222       }
2223       { family } {
2224         \use:c {
2225           \l__lngx_current_meta_family_tl family
```

```

2226     }
2227     \exp_args:NV \fontencoding
2228                 \l_lngx_current_encoding_tl
2229     \selectfont
2230 }
2231 { series } {
2232     \use:c {
2233         \l_lngx_current_series_tl series
2234     }
2235 }
2236 { shape } {
2237     \use:c {
2238         \l_lngx_current_shape_tl shape
2239     }
2240 }
2241 }
2242 }
2243 }
2244
2245 \cs_gset_eq:NN \softsuperfontfamily
2246                 \lngx_soft_super_font_family:nn

```

(End of definition for `\lngx_soft_super_font_family:nn` and `\softsuperfontfamily`. These functions are documented on page 23.)

`\lngx_softer_super_font_family:n` This function excludes the encoding and resets all the other attributes. It comes with a user-side macro.

```

2247
2248 \cs_new_protected:Npn \lngx_softer_super_font_family:n #1 {
2249     \lngx_soft_super_font_family:nn { #1 } {
2250         family,
2251         series,
2252         shape
2253     }
2254 }
2255
2256 \cs_gset_eq:NN \softersuperfontfamily
2257                 \lngx_softer_super_font_family:n

```

(End of definition for `\lngx_softer_super_font_family:n` and `\softersuperfontfamily`. These functions are documented on page 23.)

`\lngx_softest_super_font_family:n` This function resets all the attributes. It is available as a user-side macro.

```

2258
2259 \cs_new_protected:Npn \lngx_softest_super_font_family:n #1 {
2260     \lngx_soft_super_font_family:nn { #1 } {
2261         encoding,
2262         family,
2263         series,
2264         shape
2265     }
2266 }
2267
2268 \cs_gset_eq:NN \softestsuperfontfamily
2269                 \lngx_softest_super_font_family:n

```

(End of definition for `\lngx_softest_super_font_family:n` and `\softestsuperfontfamily`. These functions are documented on page 23.)

`\lngx_soft_normal_font:n` 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.
`\softnormalfont`

```

2270
2271 \cs_new_protected:Npn \lngx_soft_normal_font:n #1 {
2272   \tl_set:Nc \l_lngx_current_super_family_tl { default }
2273   \clist_map_inline:nn {
2274     rm,
2275     sf,
2276     tt
2277   } {
2278     \cs_set:cpe { ##1 default } {
2279       \tl_use:c { c_lngx_default_ ##1 default _tl }
2280     }
2281   }
2282   \normalfont
2283   \clist_map_inline:nn { #1 } {
2284     \str_case:nn { ##1 } {
2285       { encoding } {
2286         \exp_args:NV \fontencoding
2287           \l_lngx_current_encoding_tl
2288       }
2289       { family } {
2290         \use:c {
2291           \l_lngx_current_meta_family_tl family
2292         }
2293         \exp_args:NV \fontencoding
2294           \l_lngx_current_encoding_tl
2295         \selectfont
2296       }
2297       { series } {
2298         \use:c {
2299           \l_lngx_current_series_tl series
2300         }
2301       }
2302       { shape } {
2303         \use:c {
2304           \l_lngx_current_shape_tl shape
2305         }
2306       }
2307     }
2308   }
2309 }
2310
2311 \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 23.)

`\lngx_softer_normal_font:` This is a parallel to the ‘softer’ super family command for the default super family.
`\softernormalfont`

```

2312
2313 \cs_new_protected:Npn \lngx_softer_normal_font: {

```

```

2314 \lmgx_soft_normal_font:n {
2315     family,
2316     series,
2317     shape
2318 }
2319 }
2320
2321 \cs_gset_eq:NN \softernormalfont \lmgx_softer_normal_font:

```

(End of definition for `\lmgx_softer_normal_font:` and `\softernormalfont`. These functions are documented on page 23.)

`\lmgx_softest_normal_font:` This is a parallel to the ‘softest’ super family command for the default super family.
`\softestnormalfont`

```

2322
2323 \cs_new_protected:Npn \lmgx_softest_normal_font: {
2324     \lmgx_soft_normal_font:n {
2325         encoding,
2326         family,
2327         series,
2328         shape
2329     }
2330 }
2331
2332 \cs_gset_eq:NN \softestnormalfont \lmgx_softest_normal_font:

```

(End of definition for `\lmgx_softest_normal_font:` and `\softestnormalfont`. These functions are documented on page 23.)

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

```

2333 \cs_new:Npn \CurrentEncoding {
2334     \tl_use:N \l_lmgx_current_encoding_tl
2335 }
2336 \cs_new:Npn \CurrentMetaFamily {
2337     \tl_use:N \l_lmgx_current_meta_family_tl
2338 }
2339 \cs_new:Npn \CurrentSuperFamily {
2340     \tl_use:N \l_lmgx_current_super_family_tl
2341 }
2342 \cs_new:Npn \CurrentSeries {
2343     \tl_use:N \l_lmgx_current_series_tl
2344 }
2345 \cs_new:Npn \CurrentShape {
2346     \tl_use:N \l_lmgx_current_shape_tl
2347 }
2348 </nfss>

```

(End of definition for `\CurrentEncoding` and others. These functions are documented on page 18.)

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).

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols

`\` 62, 63, 64, 1788

A

`\addto` 21
`\Alph` 1779
`\alph` 1775
`\arabic` 68, 69, 1701, 1757
`\AssignTaggingSocketPlug` 650

B

`\babelfont` 1659, 1662, 1665, 1669, 1672, 1675, 1680, 1684, 1688
`\babelprovide` 1651
`\begin` 972, 1102
bool commands:
 `\bool_gset_false:N` 986
 `\bool_if:NTF` 509, 511, 518, 520, 850, 884, 932, 939, 1023, 1107
 `\bool_new:N` 561
 `\bool_set_true:N` 834
bourbaki's `\empty_set` 7, 136

C

`\cleaders` 952
clist commands:
 `\clist_gset:Nn` 1796
 `\clist_if_empty:NTF` 1798, 1816
 `\clist_map_inline:Nn` 1817
 `\clist_map_inline:nn` 155, 246, 253, 341, 473, 1217, 1308, 1315, 1403, 1597, 1906, 2125, 2141, 2156, 2176, 2205, 2217, 2273, 2283
 `\clist_new:N` 1648
 `\clist_pop:NN` 1799, 1818
 `\clist_set_eq:NN` 1797
 `\l_tmpa_clist` 1797, 1798, 1799, 1816, 1817, 1818
color commands:
 `\color_group_begin:` 1886
 `\color_group_end:` 1889
 `\color_select:n` 1887
 `\color_set:nn` 1880
columns 10, 682
cs commands:
 `\cs_generate_variant:Nn` . 467, 468, 469, 470, 471, 1581, 1582, 1583, 1654, 1691, 1692, 1693, 1760, 2123
 `\cs_gset_eq:NN` . . 48, 83, 84, 824, 1595, 1655, 1699, 1701, 1744, 1757, 1784, 2200, 2245, 2256, 2268, 2311, 2321, 2332
 `\cs_gset_protected:Npn` 1658, 1661, 1664, 1679, 1683, 1687, 1751

\cs_if_exist:NTF	789
\cs_new:Npn	1894, 1911, 2333, 2336, 2339, 2342, 2345
\cs_new_eq:NN	2051, 2052, 2053, 2068, 2069, 2070, 2085, 2086, 2087, 2102, 2103, 2104, 2119, 2120, 2121
\cs_new_protected:Npn	46, 44, 442, 445, 448, 453, 457, 478, 489, 497, 740, 753, 921, 985, 1560, 1567, 1574, 1591, 1602, 1650, 1668, 1671, 1674, 1695, 1774, 2194, 2203, 2248, 2259, 2271, 2313, 2323
\cs_set:Npe	2211, 2278
\cs_set:Npn	594, 1703, 1706, 1709, 1712, 1715, 1718, 1721, 1724, 1727, 1730, 1733, 1736, 1775, 1776, 1777, 1778, 1779, 1780
\cs_set_eq:NN	599
\cs_undefine:N	827
\CurrentEncoding	18, 2333
\CurrentMetaFamily	18, 2333
\CurrentSeries	18, 2333
\CurrentShape	18, 2333
\CurrentSuperFamily	18, 2339
D	
\DeclareMathVersion	459
dim commands:	
\dim_zero_new:N	565, 566, 567, 568
\DocumentMetadata	45
E	
\em	962
\emph	1784
\encodingdefault	1946
\end	981, 1150
entry_separator	10, 713
exp commands:	
\exp_args:Ne	70, 612, 626, 764, 787, 846, 882, 1807, 1824
\exp_args:Nee	45, 757
\exp_args:NV	2220, 2227, 2286, 2293
\exp_not:N	795, 806, 1105, 1108
\exp_not:n	207, 212, 231, 235, 285, 308, 313, 331, 335, 628, 766, 1077, 1084, 1091, 1126, 1133, 1140, 1269, 1274, 1293, 1297, 1347, 1370, 1375, 1393, 1397
expansion	9, 730
expansion_case	9, 666
F	
file commands:	
\file_if_exist:nTF	1806, 1821
\finalhyphendemerits	930
\fontencoding	2220, 2227, 2286, 2293
format	9, 651
G	
\GetDocumentProperties	721
\gla	83, 84

<code>gloss</code>	9, <u>724</u>
<code>\glossaryname</code>	52, 1101, 1108
<code>\glx</code>	8, 10, <u>830</u>
<code>\glx*</code>	8, 10
group commands:	
<code>\group_begin:</code>	166, 177, 191, 264, 275, 292, 623, 741, 763, 792, 832, 879, 926, 987, 1002, 1064, 1113, 1156, 1228, 1239, 1253, 1326, 1337, 1354, 1883
<code>\group_end:</code>	173, 187, 217, 271, 288, 318, 641, 781, 812, 821, 915, 918, 965, 1096, 1099, 1148, 1152, 1163, 1235, 1249, 1279, 1333, 1350, 1380, 1891

H

<code>\hbox</code>	132, 953
hook commands:	
<code>\hook_gput_code:nnn</code>	68, 82, 505, 1613, 1940, 1955, 1959, 1965, 1970, 1977, 1982, 1989, 1994, 2001, 2005, 2009, 2013, 2017, 2021, 2025, 2029, 2033
<code>\hskip</code>	954
<code>\hss</code>	953
<code>\hyperlink</code>	639, 779
<code>\hypersetup</code>	626, 764

I

<code>\IfBlankF</code>	1890
<code>\IfBooleanT</code>	833
<code>\IfBooleanTF</code>	1201
<code>\IfDocumentMetadataT</code>	942, 955, 1011, 1029
<code>\IfDocumentMetadataTF</code>	756
<code>\IfEncodingF</code>	18, <u>2050</u>
<code>\IfEncodingT</code>	18, <u>2050</u>
<code>\IfEncodingTF</code>	18, <u>2050</u>
<code>\IfMetaFamilyF</code>	18, <u>2067</u>
<code>\IfMetaFamilyT</code>	18, <u>2067</u>
<code>\IfMetaFamilyTF</code>	18, <u>2067</u>
<code>\IfPackageLoadedF</code>	10, 13, 16, 19, 22, 25, 28, 31, 73, 107, 112, 113, 118, 123, 441, 557, 1177, 1178, 1183, 1188, 1192, 1196, 1638, 1642, 1847
<code>\IfPackageLoadedT</code>	458, 541, 1781
<code>\IfPackageLoadedTF</code>	69, 71, 622, 762, 1657, 1895
<code>\IfPDFManagementActiveT</code>	579, 631, 719, 769
<code>\IfSeriesF</code>	18, <u>2101</u>
<code>\IfSeriesT</code>	18, <u>2101</u>
<code>\IfSeriesTF</code>	18, <u>2101</u>
<code>\IfShapeF</code>	18, <u>2118</u>
<code>\IfShapeT</code>	18, <u>2118</u>
<code>\IfShapeTF</code>	18, <u>2118</u>
<code>\IfSuperFamilyF</code>	18, <u>2084</u>
<code>\IfSuperFamilyT</code>	18, <u>2084</u>
<code>\IfSuperFamilyTF</code>	18, <u>2084</u>
int commands:	
<code>\int_compare:nNnTF</code>	969, 978
<code>\int_gincr:N</code>	786

\int_gzero_new:N	569
\int_use:N	788, 790, 797, 808, 970, 973, 979
ipa_bold_italic	<u>I2</u> , <u>I2I3</u>
ipa_bold_italic_features	<u>I2</u> , <u>I2I3</u>
ipa_bold_slanted	<u>I2</u> , <u>I2I3</u>
ipa_bold_slanted_features	<u>I2</u> , <u>I2I3</u>
ipa_bold_swash	<u>I2</u> , <u>I2I3</u>
ipa_bold_swash_features	<u>I2</u> , <u>I2I3</u>
ipa_bold_upright	<u>II</u> , <u>I2I3</u>
ipa_bold_upright_features	<u>II</u> , <u>I2I3</u>
ipa commands:	
\ipa_titlecase_all:n	<u>I358</u>
ipa_italic	<u>I2</u> , <u>I2I3</u>
ipa_italic_features	<u>I2</u> , <u>I2I3</u>
ipa_main_extra_features	<u>I3</u> , <u>I283</u>
ipa_main_font	<u>II</u> , <u>I402</u>
ipa_mono_bold_italic	<u>I2</u> , <u>I30I</u>
ipa_mono_bold_italic_features	<u>I2</u> , <u>I30I</u>
ipa_mono_bold_slanted	<u>I2</u> , <u>I30I</u>
ipa_mono_bold_slanted_features	<u>I3</u> , <u>I30I</u>
ipa_mono_bold_swash	<u>I3</u> , <u>I30I</u>
ipa_mono_bold_swash_features	<u>I3</u> , <u>I30I</u>
ipa_mono_bold_upright	<u>I2</u> , <u>I30I</u>
ipa_mono_bold_upright_features	<u>I2</u> , <u>I30I</u>
ipa_mono_extra_features	<u>I3</u> , <u>I30I</u>
ipa_mono_font	<u>I2</u> , <u>I402</u>
ipa_mono_italic	<u>I2</u> , <u>I30I</u>
ipa_mono_italic_features	<u>I2</u> , <u>I30I</u>
ipa_mono_slanted	<u>I2</u> , <u>I30I</u>
ipa_mono_slanted_features	<u>I2</u> , <u>I30I</u>
ipa_mono_small_caps	<u>I3</u> , <u>I30I</u>
ipa_mono_small_caps_features	<u>I3</u> , <u>I30I</u>
ipa_mono_swash	<u>I3</u> , <u>I30I</u>
ipa_mono_swash_features	<u>I3</u> , <u>I30I</u>
ipa_mono_upright	<u>I2</u> , <u>I30I</u>
ipa_mono_upright_features	<u>I2</u> , <u>I30I</u>
ipa_newcm	<u>II</u> , <u>I4I3</u>
ipa_newcm_mono	<u>II</u> , <u>I46I</u>
ipa_newcm_regular	<u>II</u> , <u>I485</u>
ipa_newcm_regular_mono	<u>II</u> , <u>I533</u>
ipa_newcm_regular_sans	<u>II</u> , <u>I509</u>
ipa_newcm_sans	<u>II</u> , <u>I437</u>
ipa_sans_bold_italic	<u>I2</u> , <u>I30I</u>
ipa_sans_bold_italic_features	<u>I2</u> , <u>I30I</u>

<code>ipa_sans_bold_slanted</code>	I2 , I30I
<code>ipa_sans_bold_slanted_features</code>	I2 , I30I
<code>ipa_sans_bold_swash</code>	I2 , I30I
<code>ipa_sans_bold_swash_features</code>	I2 , I30I
<code>ipa_sans_bold_upright</code>	I2 , I30I
<code>ipa_sans_bold_upright_features</code>	I2 , I30I
<code>ipa_sans_extra_features</code>	I3 , I30I
<code>ipa_sans_font</code>	I2 , I402
<code>ipa_sans_italic</code>	I2 , I30I
<code>ipa_sans_italic_features</code>	I2 , I30I
<code>ipa_sans_slanted</code>	I2 , I30I
<code>ipa_sans_slanted_features</code>	I2 , I30I
<code>ipa_sans_small_caps</code>	I2 , I30I
<code>ipa_sans_small_caps_features</code>	I2 , I30I
<code>ipa_sans_swash</code>	I2 , I30I
<code>ipa_sans_swash_features</code>	I2 , I30I
<code>ipa_sans_upright</code>	I2 , I30I
<code>ipa_sans_upright_features</code>	I2 , I30I
<code>ipa_slanted</code>	I2 , I2I3
<code>ipa_slanted_features</code>	I2 , I2I3
<code>ipa_small_caps</code>	I2 , I2I3
<code>ipa_small_caps_features</code>	I2 , I2I3
<code>ipa_swash</code>	I2 , I2I3
<code>ipa_swash_features</code>	I2 , I2I3
<code>ipa_upright</code>	II , I2I3
<code>ipa_upright_features</code>	II , I2I3
<code>\ipatext</code>	II , II99
<code>\ipatext*</code>	II , II99

K

<code>\kern</code>	I30 , I3I , I32 , I33 , 96I
keys commands:	
<code>\l_keys_choice_str</code>	662 , 67I , 678 , I766
<code>\keys_define:nn</code>	I37 , I74 , 224 , 272 , 322 , 346 , 358 , 369 , 380 , 39I , 402 , 4I3 , 424 , 652 , 725 , I236 , I286 , I334 , I384 , I408 , I4I4 , I438 , I462 , I486 , I5I0 , I534 , I762 , I793 , 2I8I
<code>\keys_set:nn</code>	45 , 737 , 836 , 840 , I003 , II58 , II6I , 2I96

L

<code>\label</code>	45 , 788
<code>\labelenumii</code>	I776
<code>\labelenumiii</code>	I778
<code>\labelenumiv</code>	I780
<code>languages</code>	I5 , I792
<code>\LaTeX</code>	5 , I26
<code>\leftskip</code>	929

\lingset	1782
\linguistix	5, 19, 47
link_color	9, 655
\listofglosses	8, 10, 20, 1154
lngx commands:	
\g_lngx_bourbaki_bool	20, 136
\lngx_build_main_ipa_font_features:	1614
\lngx_build_mono_ipa_font_features:	1622
\lngx_build_sans_ipa_font_features:	1618
\lngx_counter:n	15, 21, 68, 69, 1700, 1701, 1704, 1707, 1710, 1713, 1716, 1719, 1722, 1725, 1728, 1731, 1734, 1737, 1744, 1751, 1757
\l_lngx_current_encoding_tl	22, 1934, 1944, 1960, 1972, 1984, 1996, 2043, 2221, 2228, 2287, 2294, 2334
\l_lngx_current_meta_family_tl	22, 1935, 1944, 1966, 1971, 1978, 1983, 1990, 1995, 2061, 2133, 2225, 2291, 2337
\l_lngx_current_series_tl	22, 1937, 1944, 2002, 2006, 2095, 2148, 2233, 2299, 2343
\l_lngx_current_shape_tl	22, 1938, 1944, 2010, 2014, 2018, 2022, 2026, 2030, 2034, 2112, 2168, 2238, 2304, 2346
\l_lngx_current_super_family_tl	22, 1936, 1944, 2078, 2204, 2272, 2340
\c_lngx_default_rmdefault_tl	22, 1939
\c_lngx_default_sfdefault_tl	22, 1939
\c_lngx_default_ttdefault_tl	22, 1939
\l_lngx_expansion_bool	561, 834, 850, 884
\lngx_expansion_format:n	20, 730, 731, 1035, 1044, 1053, 1075, 1082, 1089, 1124, 1131, 1138
\l_lngx_expansion_separator_tl	563, 887, 895, 903
\lngx_gloss_format:n	20, 638, 643, 724, 727, 778, 783, 933, 1024, 1068
\g_lngx_gloss_link_color_str	40
\lngx_gloss_list:	20, 984, 985, 1162
\lngx_gloss_new:nn	20, 46, 739, 740, 824, 828
\l_lngx_gloss_separator_tl	562, 912
\l_lngx_glossary_separator_tl	564, 936, 1026, 1070
\l_lngx_gls_language_str	570, 720
\l_lngx_i_hack_dim	568
\l_lngx_i_have_dim	565
\l_lngx_i_need_dim	566
\lngx_if_encoding:n	2037
\lngx_if_encoding:nTF	22, 2036, 2051, 2052, 2053
\lngx_if_encoding:p:n	22, 2036
\lngx_if_meta_family:n	2055
\lngx_if_meta_family:nTF	22, 2054, 2068, 2069, 2070
\lngx_if_meta_family:p:n	22, 2054
\lngx_if_meta_family_rm:TF	22, 2124
\lngx_if_meta_family_rm:p:	22, 2124
\lngx_if_meta_family_sf:TF	22, 2124
\lngx_if_meta_family_sf:p:	22, 2124

\lngx_if_meta_family_tt:TF	22, <u>2124</u>
\lngx_if_meta_family_tt_p:	22, <u>2124</u>
\lngx_if_series:n	2089
\lngx_if_series:nTF	22, <u>2088</u> , 2102, 2103, 2104
\lngx_if_series_bf:TF	23, <u>2140</u>
\lngx_if_series_bf_p:	23, <u>2140</u>
\lngx_if_series_md:TF	23, <u>2140</u>
\lngx_if_series_md_p:	23, <u>2140</u>
\lngx_if_series_p:n	22, <u>2088</u>
\lngx_if_shape:n	2106
\lngx_if_shape:nTF	22, <u>2105</u> , 2119, 2120, 2121
\lngx_if_shape_it:TF	23, <u>2155</u>
\lngx_if_shape_it_p:	23, <u>2155</u>
\lngx_if_shape_p:n	22, <u>2105</u>
\lngx_if_shape_sc:TF	23, <u>2155</u>
\lngx_if_shape_sc_p:	23, <u>2155</u>
\lngx_if_shape_sl:TF	23, <u>2155</u>
\lngx_if_shape_sl_p:	23, <u>2155</u>
\lngx_if_shape_ssc:TF	23, <u>2155</u>
\lngx_if_shape_ssc_p:	23, <u>2155</u>
\lngx_if_shape_sw:TF	23, <u>2155</u>
\lngx_if_shape_sw_p:	23, <u>2155</u>
\lngx_if_shape_ulc:TF	23, <u>2155</u>
\lngx_if_shape_ulc_p:	23, <u>2155</u>
\lngx_if_shape_up:TF	23, <u>2155</u>
\lngx_if_shape_up_p:	23, <u>2155</u>
\lngx_if_super_family:n	2072
\lngx_if_super_family:nTF	22, <u>2071</u> , 2085, 2086, 2087
\lngx_if_super_family_p:n	22, <u>2071</u>
lngx_ipa	21, <u>1584</u>
\lngx_ipa:	21, <u>1590</u> , 1591, 1595
lngx_ipa_rm_nfss	21, <u>1559</u>
lngx_ipa_sf_nfss	21, <u>1559</u>
lngx_ipa_tt_nfss	21, <u>1559</u>
\lngx_languages:nn	21, <u>1649</u> , 1650, 1654, 1655, 1805, 1820
\g_lngx_languages_clist	21, <u>1647</u> , 1796, 1797
\lngx_load_languages:n	21, <u>1694</u> , 1695, 1699
\lngx_logo_font:	22, <u>1850</u> , 1851, 1884
\lngx_main_ipa:	21, <u>1559</u> , 1561
\g_lngx_main_language_tl	21, <u>1645</u> , 1745, 1800
\lngx_misc_reset:	21, <u>1773</u> , 1774
\lngx_mono_ipa:	21, <u>1559</u> , 1575
lngx_multicols	20, 967
\g_lngx_old_style_bool	20, <u>136</u> , 509, 518

<code>\g_lngx_old_style_one_bool</code>	20, 136, 511, 520
<code>\lngx_other_main_font:nnn</code>	20, 1678, 1679, 1691
<code>\lngx_other_mono_font:nnn</code>	20, 1678, 1687, 1693
<code>\lngx_other_sans_font:nnn</code>	20, 1678, 1683, 1692
<code>lngx_purple_color</code>	22, 1879
<code>\l_lngx_remain_dim</code>	567
<code>\lngx_sans_ipa:</code>	21, 1559, 1568
<code>\l_lngx_separator_tl</code>	54
<code>\lngx_set_keys:n</code>	19, 43, 44, 48, 435, 506, 1558, 1696
<code>\lngx_set_main_font:nn</code>	20, 440, 442, 467, 527, 1658, 1668
<code>\lngx_set_main_ipa_font:nn</code>	21, 1559, 1560, 1581, 1615
<code>\lngx_set_math_bold_font:nn</code>	457, 471, 543
<code>\lngx_set_math_font:nn</code>	20, 440, 453, 470, 539
<code>\lngx_set_mono_font:nn</code>	20, 440, 448, 469, 535, 1664, 1674
<code>\lngx_set_mono_ipa_font:nn</code>	21, 1559, 1574, 1583, 1623
<code>\lngx_set_sans_font:nn</code>	20, 440, 445, 468, 531, 1661, 1671
<code>\lngx_set_sans_ipa_font:nn</code>	21, 1559, 1567, 1582, 1619
<code>\lngx_soft_normal_font:n</code>	23, 2270, 2271, 2311, 2314, 2324
<code>\lngx_soft_super_font_family:nn</code>	23, 2202, 2203, 2246, 2249, 2260
<code>\lngx_softer_normal_font:</code>	23, 2312, 2313, 2321
<code>\lngx_softer_super_font_family:n</code>	23, 1592, 2247, 2248, 2257
<code>\lngx_softest_normal_font:</code>	23, 2322, 2323, 2332
<code>\lngx_softest_super_font_family:n</code>	23, 2258, 2259, 2269
<code>\lngx_super_font_family:nn</code>	23, 1585, 2193, 2194, 2201
<code>\g_lngx_trigger_aux_file_bool</code>	986

lngx internal commands:

<code>\g__lngx_bold_math_font_features_tl</code>	472
<code>__lngx_build_bold_math_font_features:</code>	472
<code>__lngx_build_main_font_features:</code>	472, 526
<code>__lngx_build_math_bold_features:</code>	497, 542
<code>__lngx_build_math_features:</code>	489, 538
<code>__lngx_build_math_font_features:</code>	472
<code>__lngx_build_mono_font_features:</code>	472, 534
<code>__lngx_build_sans_font_features:</code>	472, 530
<code>__lngx_dotfill:nnn</code>	920, 921, 1118
<code>\l__lngx_entry_separator_tl</code>	713, 924, 964, 1065
<code>\l__lngx_family_tmp_tl</code>	1931, 1967, 1973, 1974, 1979, 1985, 1986, 1991, 1997, 1998
<code>__lngx_gloss_description:</code>	40, 592, 594, 599, 616
<code>\g__lngx_gloss_link_color_str</code>	628, 655, 766
<code>\l__lngx_glossary_columns_int</code>	20, 682, 970, 973, 979
<code>\l__lngx_glossary_style_str</code>	573, 675, 1001
<code>\l__lngx_glosses_page_number_bool</code>	686, 939
<code>\l__lngx_gls_bold_bool</code>	699, 932, 1023
<code>\l__lngx_gls_expansion_case_str</code>	572, 666, 851, 885, 1033, 1073, 1122

\l__lngx_gls_section_number_bool	695, 1107
\l__lngx_gls_sectioning_str	691, 1103, 1106
\l__lngx_gls_sorting_order_str	571, 659, 990
\g__lngx_gls_use_order_seq	575, 814, 817, 989
\g__lngx_ipa_main_features_extra_tl	1284, 1289, 1290, 1297
\g__lngx_ipa_main_font_features_tl	1213, 1616
\g__lngx_ipa_main_font_tl	1402, 1617
\g__lngx_ipa_main_fonts_prop	1213, 1295
\g__lngx_ipa_mono_font_features_tl	1301, 1624
\g__lngx_ipa_mono_font_tl	1402, 1625
\g__lngx_ipa_mono_fonts_prop	1301
\g__lngx_ipa_sans_font_features_tl	1301, 1620
\g__lngx_ipa_sans_font_tl	1402, 1621
\g__lngx_ipa_sans_fonts_prop	1301
\g__lngx_math_bold_features_tl	351, 499, 544
\g__lngx_math_bold_font_tl	362, 545
\g__lngx_math_bold_fonts_prop	351, 498
\g__lngx_math_features_tl	351, 491, 539
\g__lngx_math_font_features_tl	472
\g__lngx_math_font_tl	360, 540
\g__lngx_math_fonts_prop	351, 490
\l__lngx_nfss_tmp_tl	1932, 2185, 2188, 2195, 2197
\l__lngx_normalfont_tmp_tl	1929
\g__lngx_page_ref_int	45, 569, 786, 788, 790, 797, 808
\l__lngx_selectfont_tmp_tl	1930, 1956, 1961, 1962
\l__lngx_separator_str	574, 577, 707, 835, 1157
\l__lngx_separator_tl	703
\g__lngx_text_main_features_extra_tl	222, 227, 228, 235
\g__lngx_text_main_font_features_tl	151, 472, 528
\g__lngx_text_main_font_tl	340, 529
\g__lngx_text_main_fonts_prop	151, 233
\g__lngx_text_mono_font_features_tl	239, 472, 536
\g__lngx_text_mono_font_tl	340, 537
\g__lngx_text_mono_fonts_prop	239
\g__lngx_text_sans_font_features_tl	239, 472, 532
\g__lngx_text_sans_font_tl	340, 533
\g__lngx_text_sans_fonts_prop	239
__lngx_tmp_text:	583, 585
\lngxipa	11, 17, 21, 1203, 1208, 1590
\lngxlogo	16, 1881, 1897, 1902, 1913
\lngxpkg	1894
\loadlanguages	15, 1694
\localecounter	1752
logical	15, 1748

M

<code>\MakeLinkTarget</code>	1020, 1066, 1119
<code>\MakeLinkTarget*</code>	50, 53
<code>math</code>	7, 351
<code>math_{bold}</code>	7, 351
<code>math_{bold}_{features}</code>	7, 351
<code>math_{features}</code>	7, 351
mode commands:	
<code>\mode_leave_vertical:</code>	610, 940
msg commands:	
<code>\msg_info:nnn</code>	87, 92
<code>\msg_new:nnn</code>	59, 1787
<code>\msg_warning:nnn</code>	1810, 1827
<code>\multicol</code>	39

N

<code>native_{numbering}</code>	15, 1759
<code>newcm</code>	6, 368
<code>newcm_{mono}</code>	6, 390
<code>newcm_{regular}</code>	6, 401
<code>newcm_{regular}_{mono}</code>	6, 423
<code>newcm_{regular}_{sans}</code>	6, 412
<code>newcm_{sans}</code>	6, 379
<code>\NewCommandCopy</code>	127
<code>\NewDocumentCommand</code>	736, 826, 831, 1155, 1200, 1882
<code>\NewDocumentEnvironment</code>	968
<code>\newfontfamily</code>	1851
<code>\newgloss</code>	8, 20, 739
<code>\NewTaggingSocket</code>	607
<code>\NewTaggingSocketPlug</code>	609
<code>no_{bold}</code>	10, 699
<code>\noindent</code>	976
<code>\normalfont</code>	2216, 2282

O

<code>off</code>	15, 1755
<code>\ogLaTeX</code>	5, 126
<code>old_{style}_{numbers}</code>	6, 136
<code>old_{style}_{one}</code>	6, 136

P

<code>page_{numbers}</code>	10, 686
<code>\par</code>	716
<code>\parfillskip</code>	928
<code>\parindent</code>	931
pdfannot commands:	
<code>\pdfannot_dict_put:nnn</code>	584
<code>\pdfstringdef</code>	583

prg commands:

- \prg_do_nothing: 599, 927, 928, 929, 930, 931, 954, 961
- \prg_new_conditional:Nnn 2037, 2055, 2072, 2089, 2106, 2123, 2130, 2145, 2165
- \prg_return_false: 2046, 2064, 2081, 2098, 2115, 2136, 2151, 2171
- \prg_return_true: 2044, 2062, 2079, 2096, 2113, 2134, 2149, 2169

prop commands:

- \prop_gclear_new:N 152, 240, 243, 352, 355, 1214, 1302, 1305
- \prop_gput:Nnn 184, 209, 233, 282, 310, 333, 1246, 1271, 1295, 1344, 1372, 1395
- \prop_map_inline:Nn 481, 490, 498, 1605
- \ProvideDocumentCommand 1101
- \providelanguage 15, 1649
- \ProvidesExplPackage 2, 36, 51, 99, 550, 1167, 1629, 1840, 1921

Q

\quad 941, 960

R

- \raisebox 131, 133
- \ref 45
- \relax 130, 131, 132, 133
- \RenewDocumentCommand 129
- \renewgloss 8, 46, 825
- \RequirePackage 11, 14, 17, 20, 23, 26, 29, 32, 108, 114, 119, 124, 558, 1179, 1184, 1189, 1193, 1197, 1639, 1643, 1807, 1824, 1848
- \rightskip 927
- \rmdefault 1941
- \roman 1777

S

- \section 42
- section_number 10, 695
- sectioning 10, 691
- \selectfont 2229, 2295
- separator 10, 703

seq commands:

- \seq_clear:N 842, 988, 1114
- \seq_gclear_new:N 575, 747
- \seq_gput_right:Nn 803, 817
- \seq_if_empty:NTF 877, 1062
- \seq_if_in:NnTF 800, 814
- \seq_map_inline:Nn 878, 1063, 1112, 1115
- \seq_pop_left:NN 844, 1009
- \seq_put_right:Nn 1116
- \seq_remove_duplicates:N 45
- \seq_set_eq:NN 989
- \seq_set_from_clist:Nn 843
- \seq_sort:Nn 992
- \seq_use:Nn 1146

\l_tmpa_seq	842, 843, 844, 877, 878, 988, 989, 992, 1009, 1062, 1063, 1112
\l_tmpb_seq	1114, 1116, 1146
\setfontfamily	1561, 1568, 1575
\setmainfont	443
\setmathfont	454, 461
\setmonofont	449
\setsansfont	446
\setupglossing	9, 735
\sfdefault	1942
\smallskip	925
socket commands:	
\socket_assign_plug:nn	602, 633, 773, 1760, 1765
\socket_if_exist:nTF	580, 632, 770
\socket_new:nn	591, 1740
\socket_new_plug:nnn	581, 593, 597, 1742, 1749, 1756
\socket_use:n	605, 1768
\softernormalfont	19, 2312
\softersuperfontfamily	19, 2247
\softestnormalfont	19, 2322
\softestsuperfontfamily	19, 2258
\softnormalfont	19, 2270
\softsuperfontfamily	19, 2202
sort	9, 659
sort commands:	
\sort_return_same:	996
\sort_return_swapped:	994
str commands:	
\c_colon_str	1004, 1159
\str_case:nn	851, 885, 990, 1033, 1073, 1122, 2218, 2284
\str_clear:N	167, 178, 192, 193, 265, 276, 293, 294, 624, 742, 880, 1008, 1229, 1240, 1254, 1255, 1327, 1338, 1355, 1356
\str_clear_new:N	570, 571, 572, 573, 574
\str_compare:nNnTF	993
\str_if_eq:nnTF	1001, 1103
\str_lowercase:n	44, 743, 846, 882
\str_replace_all:Nnn	169, 183, 199, 200, 267, 281, 300, 301, 1231, 1245, 1261, 1262, 1329, 1343, 1362, 1363
\str_set:Nn	168, 179, 194, 198, 266, 277, 295, 299, 577, 625, 720, 743, 835, 845, 881, 1010, 1157, 1230, 1241, 1256, 1260, 1328, 1339, 1357, 1361
\str_set_eq:NN	661, 670, 677
\str_use:N	186, 211, 284, 312, 707, 745, 748, 751, 754, 759, 801, 804, 815, 818, 855, 862, 869, 875, 890, 898, 906, 913, 1021, 1038, 1047, 1056, 1106, 1248, 1273, 1346, 1374, 1766
\l_tmpa_str	167, 168, 169, 171, 178, 179, 183, 186, 192, 194, 199, 211, 265, 266, 267, 269, 276, 277, 281, 284, 293, 295, 300, 312, 624, 625, 742, 743, 745, 748, 751, 754, 759, 801, 804, 815, 818, 845, 855, 862, 869, 875, 880, 881, 890, 898, 906, 913, 1008, 1010, 1021, 1038, 1047, 1056, 1229, 1230, 1231, 1233, 1240, 1241, 1245, 1248, 1254, 1256, 1261, 1273, 1327, 1328, 1329, 1331, 1338, 1339, 1343, 1346, 1355, 1357, 1362, 1374

<code>\l_tmpb_str</code>	I93, I98, 200, 202, 205, 213, 294, 299, 301, 303, 306, 314, I255, I260, I262, I264, I267, I275, I356, I361, I363, I365, I368, I376
<code>strict</code>	I5, I74I
<code>style</code>	9, 675
<code>\superfontfamily</code>	I8, 2193
sys commands:	
<code>\sys_if_engine_luatex:TF</code>	72, III, II76

T

tag commands:	
<code>\tag_mc_begin:n</code>	619, 647, 948, 958, IO16
<code>\tag_mc_end:</code>	6II, 645, 943, 956, IO12, IO30
<code>\tag_struct_begin:n</code>	613, 944, IO13
<code>\tag_struct_end:</code>	646, 957, IO3I
T _E X and L ^A T _E X 2 _ε commands:	
<code>\@currentmetafamily</code>	I949
<code>\f@encoding</code>	I956, I967, I979, I99I
<code>\glw@gla</code>	84
<code>\texorpdfstring</code>	I896, I912
<code>text_␣bold_␣italic</code>	I2, I5I
<code>text_␣bold_␣italic_␣features</code>	I2, I5I
<code>text_␣bold_␣slanted</code>	I2, I5I
<code>text_␣bold_␣slanted_␣features</code>	I2, I5I
<code>text_␣bold_␣swash</code>	I2, I5I
<code>text_␣bold_␣swash_␣features</code>	I2, I5I
<code>text_␣bold_␣upright</code>	II, I5I
<code>text_␣bold_␣upright_␣features</code>	II, I5I
text commands:	
<code>\text_lowercase:n</code>	853, 888, IO36, IO76, II25
<code>\text_titlecase_all:n</code>	I80, I95, 278, 296, 860, 896, IO45, IO83, II32, I242, I257, I340
<code>\text_titlecase_first:n</code>	867, 904, IO54, IO90, II39
<code>text_␣italic</code>	I2, I5I
<code>text_␣italic_␣features</code>	I2, I5I
<code>text_␣main_␣extra_␣features</code>	I3, 22I
<code>text_␣main_␣font</code>	II, 340
<code>text_␣mono_␣bold_␣italic</code>	I2, 239
<code>text_␣mono_␣bold_␣italic_␣features</code>	I2, 239
<code>text_␣mono_␣bold_␣slanted</code>	I2, 239
<code>text_␣mono_␣bold_␣slanted_␣features</code>	I3, 239
<code>text_␣mono_␣bold_␣swash</code>	I3, 239
<code>text_␣mono_␣bold_␣swash_␣features</code>	I3, 239
<code>text_␣mono_␣bold_␣upright</code>	I2, 239
<code>text_␣mono_␣bold_␣upright_␣features</code>	I2, 239
<code>text_␣mono_␣extra_␣features</code>	I3, 239
<code>text_␣mono_␣font</code>	I2, 340
<code>text_␣mono_␣italic</code>	I2, 239

<code>text_mono_italic_features</code>	I2 , 239
<code>text_mono_slanted</code>	I2 , 239
<code>text_mono_slanted_features</code>	I2 , 239
<code>text_mono_small_caps</code>	I3 , 239
<code>text_mono_small_caps_features</code>	I3 , 239
<code>text_mono_swash</code>	I3 , 239
<code>text_mono_swash_features</code>	I3 , 239
<code>text_mono_upright</code>	I2 , 239
<code>text_mono_upright_features</code>	I2 , 239
<code>text_sans_bold_italic</code>	I2 , 239
<code>text_sans_bold_italic_features</code>	I2 , 239
<code>text_sans_bold_slanted</code>	I2 , 239
<code>text_sans_bold_slanted_features</code>	I2 , 239
<code>text_sans_bold_swash</code>	I2 , 239
<code>text_sans_bold_swash_features</code>	I2 , 239
<code>text_sans_bold_upright</code>	I2 , 239
<code>text_sans_bold_upright_features</code>	I2 , 239
<code>text_sans_extra_features</code>	I3 , 239
<code>text_sans_font</code>	I2 , 340
<code>text_sans_italic</code>	I2 , 239
<code>text_sans_italic_features</code>	I2 , 239
<code>text_sans_slanted</code>	I2 , 239
<code>text_sans_slanted_features</code>	I2 , 239
<code>text_sans_small_caps</code>	I2 , 239
<code>text_sans_small_caps_features</code>	I2 , 239
<code>text_sans_swash</code>	I2 , 239
<code>text_sans_swash_features</code>	I2 , 239
<code>text_sans_upright</code>	I2 , 239
<code>text_sans_upright_features</code>	I2 , 239
<code>text_slanted</code>	I2 , 151
<code>text_slanted_features</code>	I2 , 151
<code>text_small_caps</code>	I2 , 151
<code>text_small_caps_features</code>	I2 , 151
<code>text_swash</code>	I2 , 151
<code>text_swash_features</code>	I2 , 151
<code>text_upright</code>	II , 151
<code>text_upright_features</code>	II , 151
<code>\textbf</code>	932 , 1023 , 1067
<code>\textit</code>	1784
<code>\textsc</code>	9 , 131 , 729
<code>\thechapter</code>	1703
<code>\theenumii</code>	1775 , 1776
<code>\theenumiii</code>	1777 , 1778
<code>\theenumiv</code>	1779 , 1780

<code>\theequation</code>	1736
<code>\thefigure</code>	1724
<code>\thefootnote</code>	1730
<code>\thempfootnote</code>	1733
<code>\thepage</code>	1721
<code>\theparagraph</code>	1715
<code>\thesection</code>	1706
<code>\thesubparagraph</code>	1718
<code>\thesubsection</code>	1709
<code>\thesubsubsection</code>	1712
<code>\thetable</code>	1727
tl commands:	
<code>\c_space_tl</code>	837, 1004, 1159
<code>\tl_clear:N</code>	793, 841, 1007, 1814, 1815, 1831, 1832, 1962, 1974, 1986, 1998, 2197
<code>\tl_clear_new:N</code>	562, 563, 564
<code>\tl_const:Nn</code>	1941, 1942, 1943
<code>\tl_gclear_new:N</code>	153, 170, 222, 241, 244, 250, 268, 353, 356, 744, 1215, 1232, 1284, 1303, 1306, 1312, 1330, 2184
<code>\tl_gput_right:Nn</code>	201, 227, 302, 325, 482, 491, 499, 1263, 1289, 1364, 1387, 1606
<code>\tl_gset:Nn</code>	750, 2187
<code>\tl_if_empty:NTF</code>	204, 228, 305, 328, 1266, 1290, 1367, 1390, 2210
<code>\tl_if_eq:NnTF</code>	2043, 2061, 2078, 2095, 2112, 2133, 2148, 2168
<code>\tl_new:N</code>	1646, 1929, 1930, 1931, 1932, 1934, 1935, 1936, 1937, 1938
<code>\tl_set:Nn</code>	705, 794, 1801, 1819, 1944, 1945, 1948, 1951, 1952, 1956, 1966, 1967, 1971, 1978, 1979, 1983, 1990, 1991, 1995, 2002, 2006, 2010, 2014, 2018, 2022, 2026, 2030, 2034, 2195, 2204, 2272
<code>\tl_set_eq:NN</code>	1800, 1960, 1972, 1984, 1996
<code>\tl_use:N</code>	847, 854, 861, 868, 887, 889, 895, 897, 903, 905, 912, 936, 1025, 1026, 1037, 1046, 1065, 1070, 1745, 2212, 2279, 2334, 2337, 2340, 2343, 2346
<code>\tl_use:n</code>	1055
<code>\l_tmpa_tl</code>	793, 794, 802, 841, 844, 847, 1007, 1009, 1010, 1025, 1799, 1800, 1805, 1806, 1808, 1812, 1814, 1818, 1820, 1822, 1825, 1829, 1831
<code>\l_tmpb_tl</code>	1801, 1805, 1815, 1819, 1820, 1832
<code>\ttdefault</code>	1943

U

<code>\umgla</code>	5, 83
use commands:	
<code>\use:N</code>	796, 807, 875, 913, 1105, 2224, 2232, 2237, 2290, 2298, 2303
<code>\use:n</code>	1104
<code>\use_ii:nnnnn</code>	795, 806
<code>\UseTaggingSocket</code>	757