

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

निरंजन

2 February 2026 (v0.8)

🏠 <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... 38	
2	Planned	4	9	LINGUIS \mathcal{T} \mathbf{I} \mathbf{X} -ipa	11
				Interface... 21; Implementation... 54	
3	Funding	4	10	LINGUIS \mathcal{T} \mathbf{I} \mathbf{X} -LANGUAGES	14
				Interface... 21; Implementation... 64	
4	Acknowledgements	4	11	LINGUIS \mathcal{T} \mathbf{I} \mathbf{X} -LOGOS	16
				Interface... 22; Implementation... 71	
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... 72	
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	85

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 Tsolomitis, the developer of New Computer Modern. In the stylistic set dedicated for linguistics, the correct shape was added to the Italic shape by him. Thus, `\ipatext{a\textit{a}}` (a command from `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. LINGUIS_{CI}X 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 LINGUIS_{CI}X-GL_{OSSING} 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.

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_{CI}X 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 $\text{\textcolor{violet}{T}}\text{\textcolor{violet}{I}}\text{\textcolor{violet}{X}}$ -BASE

$\text{\textcolor{violet}{L}}\text{\textcolor{violet}{A}}\text{\textcolor{violet}{T}}\text{\textcolor{violet}{E}}\text{\textcolor{violet}{X}}_3$ -interface | Implementation

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

$\text{\textbackslash linguistix}$ $\{ \langle \textit{key-value-list} \rangle \}$

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

6 LINGUIS $\text{\textcolor{violet}{T}}\text{\textcolor{violet}{I}}\text{\textcolor{violet}{X}}$ -FIXPEX

$\text{\textcolor{violet}{L}}\text{\textcolor{violet}{A}}\text{\textcolor{violet}{T}}\text{\textcolor{violet}{E}}\text{\textcolor{violet}{X}}_3$ -interface | Implementation

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

$\text{\textbackslash umgla}$ 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 $\text{\textcolor{violet}{T}}\text{\textcolor{violet}{I}}\text{\textcolor{violet}{X}}$ -FONTS

$\text{\textcolor{violet}{L}}\text{\textcolor{violet}{A}}\text{\textcolor{violet}{T}}\text{\textcolor{violet}{E}}\text{\textcolor{violet}{X}}_3$ -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 $\text{\textcolor{violet}{T}}\text{\textcolor{violet}{I}}\text{\textcolor{violet}{X}}$ -IPA separately.

Antonis suggested a typographic enhancement for the logo of $\text{\textcolor{violet}{L}}\text{\textcolor{violet}{A}}\text{\textcolor{violet}{T}}\text{\textcolor{violet}{E}}\text{\textcolor{violet}{X}}$. 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.

$\text{\textbackslash LaTeX}$ $\text{\textcolor{violet}{L}}\text{\textcolor{violet}{A}}\text{\textcolor{violet}{T}}\text{\textcolor{violet}{E}}\text{\textcolor{violet}{X}}$
 $\text{\textbackslash ogLaTeX}$ $\text{\textcolor{violet}{L}}\text{\textcolor{violet}{A}}\text{\textcolor{violet}{T}}\text{\textcolor{violet}{E}}\text{\textcolor{violet}{X}}$

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

1 Text

Most keys of this package are prefixed with the `text` in order to distinguish them from the maths and IPA ones. There aren’t any commands provided by the package. Most of the important features of the `fontspec` 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 Brighurst 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 LINGUIS $\text{\textcolor{violet}{T}}\text{\textcolor{violet}{I}}\text{\textcolor{violet}{X}}$ -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>= {\langle truth value \rangle}</code>	<code>true false</code>
<code>old style one</code>	<code>= {\langle truth value \rangle}</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>
<code>newcm sans</code>
<code>newcm mono</code>
<code>newcm regular</code>
<code>newcm regular sans</code>
<code>newcm regular mono</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 **regular** in their names refer to the ‘regular’ variants of New Computer Modern fonts. These variants match the colour and widths of the Latin Modern fonts. One may use these keys to override the defaults.

2 Maths

LINGUISTEX-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 LINGUISTEX. That will suppress the loading of `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.

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

`bourbaki's empty set` = `{(truth value)}` `true | false`

In (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 `\emptysetset` 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.

$\backslash\text{glx}$	$\{ \langle \text{comma separated list of glosses} \rangle \}$
$\backslash\text{glx}^*$	$\{ \langle \text{comma separated list of glosses} \rangle \}$

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.

$\backslash\text{newgloss}$	$\{ \langle \text{gloss} \rangle \} \{ \langle \text{expansion} \rangle \}$
$\backslash\text{renewgloss}$	$\{ \langle \text{gloss} \rangle \} \{ \langle \text{expansion} \rangle \}$

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

$\backslash\text{listofglosses}$	$[\langle \text{setup keys} \rangle]$
----------------------------------	---

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 $\overline{\text{C}}\text{I}\text{X}$ -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 $\overline{\text{C}}\text{I}\text{X}$ -FONTS and LINGUIS $\overline{\text{C}}\text{I}\text{X}$ -ipa.

Family	LINGUIS $\overline{\text{C}}\text{I}\text{X}$ -FONTS	LINGUIS $\overline{\text{C}}\text{I}\text{X}$ -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	LINGUISCI-X-FONTS	LINGUISCI-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 extra features
- text sans extra features
- text mono extra features
- ipa 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 `LATEX` 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. `LATEX`-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.

<code>\IfEncodingTF</code>	<code>★ {⟨encoding⟩} {⟨true code⟩} {⟨false code⟩}</code>
<code>\IfEncodingT</code>	<code>★ {⟨encoding⟩} {⟨true code⟩}</code>
<code>\IfEncodingF</code>	<code>★ {⟨encoding⟩} {⟨false code⟩}</code>
<code>\CurrentEncoding</code>	<code>★</code> If the current encoding matches with the given <code>⟨encoding⟩</code> , it selects the true branch; false otherwise. The <code>\CurrentEncoding</code> macro expands to the current encoding.

<code>\IfMetaFamilyTF</code>	<code>★ {⟨meta family⟩} {⟨true code⟩} {⟨false code⟩}</code>
<code>\IfMetaFamilyT</code>	<code>★ {⟨meta family⟩} {⟨true code⟩}</code>
<code>\IfMetaFamilyF</code>	<code>★ {⟨meta family⟩} {⟨false code⟩}</code>
<code>\CurrentMetaFamily</code>	<code>★</code> If the current meta family matches with the given <code>⟨meta family⟩</code> , it selects the true branch; false otherwise. The <code>\CurrentMetaFamily</code> macro expands to the current meta family.

<code>\IfSuperFamilyTF</code>	<code>★ {⟨super family⟩} {⟨true code⟩} {⟨false code⟩}</code>
<code>\IfSuperFamilyT</code>	<code>★ {⟨super family⟩} {⟨true code⟩}</code>
<code>\IfSuperFamilyF</code>	<code>★ {⟨super family⟩} {⟨false code⟩}</code>
<code>\CurrentSuperFamily</code>	<code>★</code> If the current super family matches with the given <code>⟨super family⟩</code> , it selects the true branch; false otherwise. The <code>\CurrentSuperFamily</code> macro expands to the current super family.

<code>\IfSeriesTF</code>	<code>★ {⟨series⟩} {⟨true code⟩} {⟨false code⟩}</code>
<code>\IfSeriesT</code>	<code>★ {⟨series⟩} {⟨true code⟩}</code>
<code>\IfSeriesF</code>	<code>★ {⟨series⟩} {⟨false code⟩}</code>
<code>\CurrentSeries</code>	<code>★</code> If the current series matches with the given <code>⟨series⟩</code> , it selects the true branch and false otherwise. The <code>\CurrentSeries</code> macro expands to the current series.

<code>\IfShapeTF</code>	<code>★ {⟨shape⟩} {⟨true code⟩} {⟨false code⟩}</code>
<code>\IfShapeT</code>	<code>★ {⟨shape⟩} {⟨true code⟩}</code>
<code>\IfShapeF</code>	<code>★ {⟨shape⟩} {⟨false code⟩}</code>
<code>\CurrentShape</code>	<code>★</code> If the current series matches with the given <code>⟨shape⟩</code> , it selects the true branch and false otherwise. The <code>\CurrentShape</code> macro expands to the current shape.

<code>\superfontfamily</code>	<code>{⟨family ID⟩} {⟨rm=⟨rm NFSS⟩⟩,sf=⟨sf NFSS⟩⟩,tt=⟨tt NFSS⟩⟩}</code>
-------------------------------	---

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

<code>\softsuperfontfamily</code>	<code>{⟨ID⟩}{⟨<i>encoding, family, series, shape</i>⟩}</code>
<code>\softersuperfontfamily</code>	<code>{⟨ID⟩}</code>
<code>\softtestsuperfontfamily</code>	<code>{⟨ID⟩}</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>{⟨<i>encoding, family, series, shape</i>⟩}</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>	<code>⟨<i>keyval list</i>⟩</code>
-------------------------------	-----------------------------------

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.

<hr/> <code>\lngx_set_main_ipa_font:nn</code> <code>\lngx_set_main_ipa_font:VV</code> <code>\lngx_main_ipa:</code> <code>lngx_ipa_rm_nfss</code> <hr/>	<code>{\features}</code> <code>{\font}</code> These functions set the IPA fonts for the serif variants. The <code>\font</code> is set with <code>\features</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 <code>lngx_ipa_rm_nfss</code> .
<hr/> <code>\lngx_set_sans_ipa_font:nn</code> <code>\lngx_set_sans_ipa_font:VV</code> <code>\lngx_sans_ipa:</code> <code>lngx_ipa_sf_nfss</code> <hr/>	<code>{\features}</code> <code>{\font}</code> These functions set the IPA fonts for the sans variants. The <code>\font</code> is set with <code>\features</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 <code>lngx_ipa_sf_nfss</code> .
<hr/> <code>\lngx_set_mono_ipa_font:nn</code> <code>\lngx_set_mono_ipa_font:VV</code> <code>\lngx_mono_ipa:</code> <code>lngx_ipa_tt_nfss</code> <hr/>	<code>{\features}</code> <code>{\font}</code> These functions set the IPA fonts for the mono variants. The <code>\font</code> is set with <code>\features</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 <code>lngx_ipa_nfss_nfss</code> .
<hr/> <code>\lngx_ipa:</code> <code>lngx_ipa</code> <hr/>	The <code>\lngx_ipa:</code> command loads the super family <code>lngx_ipa</code> (see the documentation of 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.

<hr/> <code>\g_lngx_main_language_tl</code> <hr/>	A <code>tl</code> that globally stores the main language of the document.
<hr/> <code>\g_lngx_languages_clist</code> <hr/>	A <code>clist</code> that globally stores the languages that are used.
<hr/> <code>\lngx_languages:nn</code> <code>\lngx_languages:VV</code> <hr/>	<code>{\language options}</code> <code>{\language name}</code> <code>\language options tl</code> <code>\language tl</code> These functions read the V-type argument provided to them and pass it to the <code>\babelprovide</code> command for loading languages.
<hr/> <code>\lngx_load_languages:n</code> <hr/>	<code>{\list of languages}</code> This function loads the languages in LINGUISṪIX sense.
<hr/> <code>\lngx_counter:n</code> <hr/>	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 <code>native-numbering</code> socket.
<hr/> <code>\lngx_misc_reset:</code> <hr/>	This function resets a lot of custom settings done by some languages. It has to be used inside <code>\addto</code> macro provided by the <code>babel</code> 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 `l3color` 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 `\begindocument/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 \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}

```

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**TI**X

Provide the package with its basic information.

```

1 (*package)
2 \ProvidesExplPackage{linguistix}
3     {2026-02-02}
4     {v0.8}
5     {%
6         The ‘LinguisTiX’ bundle: Enhanced
7         support for linguistics.%
8     }

```

When one loads `LINGUIS-T1X`, all the packages of the bundle are loaded automatically. That's the only content of the umbrella package `LINGUIS-T1X`. All the packages are loaded conditionally (i.e., only if not loaded already).

```

9 \IfPackageLoadedF { linguistix-base } {
10   \RequirePackage { linguistix-base }
11 }
12
13 \IfPackageLoadedF { linguistix-fonts } {
14   \RequirePackage { linguistix-fonts }
15 }
16
17 \IfPackageLoadedF { linguistix-glossing } {
18   \RequirePackage { linguistix-glossing }
19 }
20
21 \IfPackageLoadedF { linguistix-ipa } {
22   \RequirePackage { linguistix-ipa }
23 }
24
25 \IfPackageLoadedF { linguistix-languages } {
26   \RequirePackage { linguistix-languages }
27 }
28
29 \IfPackageLoadedF { linguistix-leipzig } {
30   \RequirePackage { linguistix-leipzig }
31 }
32
33 \IfPackageLoadedF { linguistix-logos } {
34   \RequirePackage { linguistix-logos }
35 }
36
37 \IfPackageLoadedF { linguistix-nfss } {
38   \RequirePackage { linguistix-nfss }
39 }
40 }
41
42 \end{package}

```


Set the essentials of the package.

```

35 <*base>
36 \ProvidesExplPackage{linguistix-base}
37     {2026-02-02}
38     {v0.8}
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     {2026-02-02}
53     {v0.8}
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     {2026-02-02}
101     {v0.8}
102     {%
103         The font-assistant package of the
104         'Linguistix' 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 I use the **.bool_gset:N** key-type of **l3keys** for developing these boolean keys.
\g_lngx_old_style_bool
old style one
\g_lngx_old_style_one_bool
bourbaki's empty set
\g_lngx_bourbaki_bool

```

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
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 here are prefixed with the word `text` in order to distinguish them from the keys provided by the `LINGUISTLX-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.

```

166   \keys_define:nn { lngx_keys } {
167     text~ #1
168     .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`.

```

169     \group_begin:

```

```

170 \str_clear:N \l_tmpa_str
171 \str_set:Ne \l_tmpa_str {
172   \text_titlecase_all:n { #1 }
173   Font
174 }
175 \str_replace_all:Nnn \l_tmpa_str { ~ } { }

```

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

```

176 \prop_gput:Nne \g__lngx_text_main_fonts_prop
177   { text~ #1 }
178   { \str_use:N \l_tmpa_str = { ##1 } }
179 \group_end:
180 },

```

Same is repeated for features.

```

181 text~ #1~ features
182 .code:n = {
183   \group_begin:
184   \str_clear:N \l_tmpa_str
185   \str_set:Ne \l_tmpa_str {
186     \text_titlecase_all:n { #1 }
187     Features
188   }
189   \str_replace_all:Nnn \l_tmpa_str { ~ } { }
190   \prop_gput:Nne \g__lngx_text_main_fonts_prop
191     { text~ #1~ features }
192     {
193       \str_use:N \l_tmpa_str = { ##1 }
194     }
195   \group_end:
196 }
197 }
198 }

```

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

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

```

199
200 \keys_define:nn { lngx_keys } {
201   text~ extra~ features
202   .prop_gput:N = \g__lngx_text_main_fonts_prop
203 }

```

(End of definition for `text 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 upright
text sans upright features
text sans bold upright
text sans bold upright features
text sans italic
text sans italic features
text sans bold italic
text sans bold italic features
text sans slanted
text sans slanted features
text sans bold slanted
text sans bold slanted features
text sans swash
text sans swash features
text sans bold swash
text sans bold swash features
text sans small caps
text sans small caps features
text mono upright
text mono upright features
text mono bold upright
text mono bold upright features
text mono italic
text mono italic features
text mono bold italic
text mono bold italic features
text mono slanted
text mono slanted features
text mono bold slanted
text mono bold slanted features
text mono swash
text mono swash features
text mono bold swash
text mono bold swash features
text mono small caps
text mono small caps features

```

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

```

204
205 \prop_gclear_new:N \g__lngx_text_sans_fonts_prop
206 \tl_gclear_new:N \g__lngx_text_sans_font_features_tl
207
208 \prop_gclear_new:N \g__lngx_text_mono_fonts_prop
209 \tl_gclear_new:N \g__lngx_text_mono_font_features_tl
210
211 \clist_map_inline:nn {
212   sans,
213   mono
214 } {
215   \clist_map_inline:nn {
216     upright,
217     bold~ upright,
218     italic,
219     bold~ italic,
220     slanted,
221     bold~ slanted,
222     swash,
223     bold~ swash,
224     small~ caps
225   } {
226     \keys_define:nn { lngx_keys } {
227       text~ #1~ ##1
228       .code:n          = {
229         \group_begin:
230         \str_clear:N \l_tmpa_str
231         \str_set:Ne \l_tmpa_str {
232           \text_titlecase_all:n { ##1 }
233           Font
234         }
235         \str_replace_all:Nnn \l_tmpa_str { ~ } { }
236         \prop_gput:cne { g__lngx_text_ #1 _fonts_prop }
237           { text~ #1~ ##1 }
238           { ####1 }
239         \group_end:
240       },
241       text~ #1~ ##1~ features
242       .code:n          = {
243         \group_begin:
244         \str_clear:N \l_tmpa_str
245         \str_set:Ne \l_tmpa_str {
246           \text_titlecase_all:n { #1 }
247           Features
248         }
249         \str_replace_all:Nnn \l_tmpa_str { ~ } { }
250         \prop_gput:cne { g__lngx_text_ #1 _fonts_prop }
251           { text~ #1~ ##1~ features }
252           {
253             \str_use:N \l_tmpa_str = { ####1 }
254           }
255         \group_end:

```

```

256     }
257   }
258 }
259 \keys_define:nn { lngx_keys } {
260   text~ #1~ extra~ features
261   .prop_gput:c          = {
262                         g__lngx_text_ #1 _fonts_prop
263                         }
264 }
265 }

```

(End of definition for `\g__lngx_text_sans_fonts_prop` and others. These functions are documented on page 12.)

`\g__lngx_text_main_font_tl` 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.

```

\g__lngx_text_sans_font_tl
\g__lngx_text_mono_font_tl
266   text main font
267   \clist_map_inline:nn {
268     main,
269     sans,
270     mono
271   } {
272     \keys_define:nn { lngx_keys } {
273       text~ #1~ font
274       .tl_gset:c          = { g__lngx_text_ #1 _font_tl }
275     }
276   }

```

(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
277   \g__lngx_math_bold_fonts_prop
278   \prop_gclear_new:N \g__lngx_math_fonts_prop
279   \tl_gclear_new:N \g__lngx_math_features_tl
280   math
281   \prop_gclear_new:N \g__lngx_math_bold_fonts_prop
282   \tl_gclear_new:N \g__lngx_math_bold_features_tl
283   math features
284   \keys_define:nn { lngx_keys } {
285     math
286     .tl_gset:N          = \g__lngx_math_font_tl,
287     math~ bold
288     .tl_gset:N          = \g__lngx_math_bold_font_tl,
289     math~ features
290     .prop_gput:N        = \g__lngx_math_fonts_prop,
291     math~ bold~ features
292     .prop_gput:N        = \g__lngx_math_bold_fonts_prop
293   }

```

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

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

```

294   \keys_define:nn { lngx_keys } {
295

```



```

296 newcm
297 .meta:n          = {
298   text~ main~ font    = { NewCM10-Book.otf },
299   text~ sans~ font    = { NewCMSans10-Book.otf },
300   text~ mono~ font    = { NewCMMono10-Book.otf },
301   math               = { NewCMMath-Book.otf },
302   math~ bold         = { NewCMMath-Bold.otf }
303 }
304 }

```

(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.

```

305
306 \keys_define:nn { lngx_keys } {
307   newcm~ sans
308   .meta:n          = {
309     main~ font      = { NewCMSans10-Book.otf },
310     sans~ font      = { NewCMSans10-Book.otf },
311     mono~ font      = { NewCMMono10-Book.otf },
312     math            = { NewCMSansMath-Regular.otf },
313     math~ bold      = { NewCMSansMath-Regular.otf }
314   }
315 }

```

(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.

```

316
317 \keys_define:nn { lngx_keys } {
318   newcm~ mono
319   .meta:n          = {
320     main~ font      = { NewCMMono10-Book.otf },
321     sans~ font      = { NewCMSans10-Book.otf },
322     mono~ font      = { NewCMMono10-Book.otf },
323     math            = { NewCMSansMath-Regular.otf },
324     math~ bold      = { NewCMSansMath-Regular.otf }
325   }
326 }

```

(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.

```

327
328 \keys_define:nn { lngx_keys } {
329   newcm~ regular
330   .meta:n          = {
331     main~ font      = { NewCM10-Regular.otf },
332     sans~ font      = { NewCMSans10-Regular.otf },
333     mono~ font      = { NewCMMono10-Regular.otf },
334     math            = { NewCMMath-Regular.otf },
335     math~ bold      = { NewCMMath-Bold.otf }
336   }
337 }

```

(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.

```

338
339 \keys_define:nn { lngx_keys } {
340   newcm~ regular~ sans
341   .meta:n          = {
342     main~ font      = { NewCMSans10-Regular.otf },
343     sans~ font      = { NewCMSans10-Regular.otf },
344     mono~ font      = { NewCMMono10-Regular.otf },
345     math            = { NewCMMath-Regular.otf },
346     math~ bold      = { NewCMMath-Bold.otf }
347   }
348 }
```

(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.

```

349
350 \keys_define:nn { lngx_keys } {
351   newcm~ regular~ mono
352   .meta:n          = {
353     main~ font      = { NewCMMono10-Regular.otf },
354     sans~ font      = { NewCMSans10-Regular.otf },
355     mono~ font      = { NewCMMono10-Regular.otf },
356     math            = { NewCMMath-Regular.otf },
357     math~ bold      = { NewCMMath-Bold.otf },
358   }
359 }
```

(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.

```

360
361 \lngx_set_keys:n {
362   bourbaki's~ empty~ set,
```

Then we load the *old style numbers* boolean.

```

363   old~ style~ numbers,
364   newcm
365 }
```

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

```

366
367 \IfPackageLoadedF { linguistix-languages } {
368   \cs_new_protected:Npn \lngx_set_main_font:nn #1#2 {
369     \setmainfont [ #1 ] { #2 }
370   }
371   \cs_new_protected:Npn \lngx_set_sans_font:nn #1#2 {
372     \setsansfont [ #1 ] { #2 }
```

```

373 }
374 \cs_new_protected:Npn \lngx_set_mono_font:nn #1#2 {
375   \setmonofont [ #1 ] { #2 }
376 }
377 }

```

A wrapper command is provided for loading math fonts.

```

378
379 \cs_new_protected:Npn \lngx_set_math_font:nn #1#2 {
380   \setmathfont [ #1 ] { #2 }
381 }
382
383 \cs_new_protected:Npn \lngx_set_math_bold_font:nn #1#2 {
384   \IfPackageLoadedT { lua-unicode-math } {
385     \DeclareMathVersion { bold }
386   }
387   \setmathfont [
388     #1,
389     version          = { bold }
390   ] { #2 }
391 }

```

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

```

392
393 \cs_generate_variant:Nn \lngx_set_main_font:nn { VV }
394 \cs_generate_variant:Nn \lngx_set_sans_font:nn { VV }
395 \cs_generate_variant:Nn \lngx_set_mono_font:nn { VV }
396 \cs_generate_variant:Nn \lngx_set_math_font:nn { VV }
397 \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.)

```

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

```

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

```

398
399 \clist_map_inline:nn {
400   main,
401   sans,
402   mono
403 } {
404   \cs_new_protected:cpn {
405     __lngx_build_ #1 _font_features:
406   } {
407     \prop_map_inline:cn { g__lngx_text_ #1 _fonts_prop } {
408       \tl_gput_right:cn {
409         g__lngx_text_ #1 _font_features_tl
410       } { ####2 }
411     }
412   }
413 }
414
415 \cs_new_protected:Npn \__lngx_build_math_features: {

```

```

416 \prop_map_inline:Nn \g__lngx_math_fonts_prop {
417   \tl_gput_right:Nn \g__lngx_math_features_tl {
418     { ##2 }
419   }
420 }
421 }
422
423 \cs_new_protected:Npn \__lngx_build_math_bold_features: {
424   \prop_map_inline:Nn \g__lngx_math_bold_fonts_prop {
425     \tl_gput_right:Nn \g__lngx_math_bold_features_tl {
426       { ##2 }
427     }
428   }
429 }

```

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

Now I start the pre-begindocument hook.

```

430
431 \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.

```

432 \lngx_set_keys:n {
433   text~ extra~
434   features = {
435     \bool_if:NT \g_lngx_old_style_bool {
436       Numbers = { OldStyle },
437       \bool_if:NT \g_lngx_old_style_one_bool {
438         CharacterVariant = { 6 }
439       }
440     }
441   },
442   text~ sans~ extra~
443   features = {
444     \bool_if:NT \g_lngx_old_style_bool {
445       Numbers = { OldStyle },
446       \bool_if:NT \g_lngx_old_style_one_bool {
447         CharacterVariant = { 6 }
448       }
449     }
450   }
451 }

```

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

```

452 \__lngx_build_main_font_features:
453 \lngx_set_main_font:VV
454   \g__lngx_text_main_font_features_tl
455   \g__lngx_text_main_font_tl
456 \__lngx_build_sans_font_features:
457 \lngx_set_sans_font:VV
458   \g__lngx_text_sans_font_features_tl
459   \g__lngx_text_sans_font_tl
460 \__lngx_build_mono_font_features:
461 \lngx_set_mono_font:VV
462   \g__lngx_text_mono_font_features_tl

```

```

463     \g__lngx_text_mono_font_tl
464   \__lngx_build_math_features:
465   \lngx_set_math_font:VV \g__lngx_math_features_tl
466                       \g__lngx_math_font_tl
467   \IfPackageLoadedT { unicode-math } {
468     \__lngx_build_math_bold_features:
469     \lngx_set_math_bold_font:VV
470     \g__lngx_math_bold_features_tl
471     \g__lngx_math_bold_font_tl
472   }
473 }
474 </font>

```

```

475 (*glossing)
476 \ProvidesExplPackage{linguistix-glossing}
477     {2026-02-02}
478     {v0.8}
479     {%
480     Accessible glossing with LinguistTiX%
481     }

```

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

```

482
483 \IfPackageLoadedF { multicol } {
484     \RequirePackage { multicol }
485 }

```

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

```

486
487 \bool_new:N          \l_lngx_expansion_bool
488 \tl_clear_new:N      \l_lngx_gloss_separator_tl
489 \tl_clear_new:N      \l_lngx_expansion_separator_tl
490 \tl_clear_new:N      \l_lngx_glossary_separator_tl
491 \dim_zero_new:N      \l_lngx_i_have_dim
492 \dim_zero_new:N      \l_lngx_i_need_dim
493 \dim_zero_new:N      \l_lngx_remain_dim
494 \dim_zero_new:N      \l_lngx_i_hack_dim
495 \int_gzero_new:N     \g__lngx_page_ref_int
496 \str_clear_new:N     \l_lngx_gls_language_str
497 \str_clear_new:N     \l__lngx_gls_sorting_order_str
498 \str_clear_new:N     \l__lngx_gls_expansion_case_str
499 \str_clear_new:N     \l__lngx_glossary_style_str
500 \str_clear_new:N     \l__lngx_separator_str
501 \seq_gclear_new:N    \g__lngx_gls_use_order_seq
502
503 \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.

```

504
505 \IfPDFManagementActiveT {
506     \socket_if_exist:nT { hyp / link / GoTo / Contents } {
507         \socket_new_plug:nnn { hyp / link / GoTo / Contents }
508             { text } {
509                 \pdfstringdef \__lngx_tmp_text: { #2 }
510                 \pdfannot_dict_put:nne { link / GoTo } { Contents } {
511                     ( \__lngx_tmp_text: )
512                 }
513             }
514     }
515 }

```

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

```

516
517 \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.

```

518
519 \socket_new_plug:nnn { lngx / description / gloss } { on } {
520   \cs_set:Npn \_lngx_gloss_description: { Gloss~ }
521 }
522
523 \socket_new_plug:nnn { lngx / description / gloss }
524   { off } {
525   \cs_set_eq:NN \_lngx_gloss_description: \prg_do_nothing:
526 }
527
528 \socket_assign_plug:nn { lngx / description / gloss }
529   { off }
530
531 \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.

```

532
533 \NewTaggingSocket { lngx / gloss } { 2 }
534
535 \NewTaggingSocketPlug { lngx / gloss } { default } {
536   \mode_leave_vertical:
537   \tag_mc_end:
538   \exp_args:Ne
539   \tag_struct_begin:n {
540     tag                = { Span },
541     E                  = {
542       \_lngx_gloss_description: #2
543     }
544   }
545   \tag_mc_begin:n {
546     tag                = { Span }
547   }

```

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).

```

548 \IfPackageLoadedTF { hyperref } {
549   \group_begin:
550   \str_clear:N \l_tmpa_str
551   \str_set:Nn \l_tmpa_str { #1 }
552   \exp_args:Ne \hypersetup {
553     linkcolor      = {

```

```

554     \exp_not:V \g__lngx_gloss_link_color_str
555   }
556 }

```

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

```

557   \IfPDFManagementActiveT {
558     \socket_if_exist:nT { hyp / link / GoTo / Contents } {
559       \socket_assign_plug:nn {
560         hyp / link / GoTo / Contents
561       } { text }
562     }
563   }
564   \lngx_gloss_format:n {
565     \hyperlink { lngx_ #1 _glossary } { #1 }
566   }
567   \group_end:
568 } {

```

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

```

569   \lngx_gloss_format:n { #1 }
570 }
571 \tag_mc_end:
572 \tag_struct_end:
573 \tag_mc_begin:n { }
574 }

```

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

```

575
576 \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.

```

577 \keys_define:nn { lngx_glossing } {
578   format
579   .meta:nn          = { lngx / gloss / format } { #1 },
580

```

(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
581   link~ color
582   .str_gset:N          = \g__lngx_gloss_link_color_str,
583   link~ color
584   .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.

```

585   sort
586   .choices:nn          = { alphabetical, use } {
587     \str_set_eq:NN \l__lngx_gls_sorting_order_str
588                   \l_keys_choice_str
589   },
590   sort
591   .initial:n          = { alphabetical },

```


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

expansion case 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.

`\l__lngx_gls_expansion_case_str`

```
592 expansion~ case
593 .choices:nn          = {
594   lowercase, title~ case~ all, title~ case~ first
595 } {
596   \str_set_eq:NN \l__lngx_gls_expansion_case_str
597                 \l_keys_choice_str
598 },
599 expansion~ case
600 .initial:n           = { lowercase },
```

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

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

`\l__lngx_glossary_style_str`

```
601 style
602 .choices:nn          = { block, inline } {
603   \str_set_eq:NN \l__lngx_glossary_style_str
604                 \l_keys_choice_str
605 },
606 style
607 .initial:n           = { block },
```

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

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

`\l__lngx_glossary_columns_int`

```
608 columns
609 .int_set:N           = \l__lngx_glossary_columns_int,
610 columns
611 .initial:n           = { 2 },
```

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

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

`\l__lngx_glosses_page_number_bool`

```
612 page~ numbers
613 .bool_set:N          =
614   \l__lngx_glosses_page_number_bool,
615 page~ numbers
616 .initial:n           = { true },
```

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

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

`\l__lngx_gls_sectioning_str`

```
617 sectioning
618 .str_set:N           = \l__lngx_gls_sectioning_str,
619 sectioning
620 .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.

```

621 section~ number
622 .bool_set:N          = \l__lngx_gls_section_number_bool,
623 section~ number
624 .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).

```

625 no~ bold
626 .bool_set_inverse:N   = \l__lngx_gls_bold_bool,
627 no~ bold
628 .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.

```

629 separator
630 .code:n              = {
631   \tl_set:cn {
632     l__lngx_
633     \str_use:N \l__lngx_separator_str
634     _separator_tl
635   } { #1 }
636 },
637 separator
638 .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.

```

639 entry~ separator
640 .tl_set:N             = \l__lngx_entry_separator_tl,
641 entry~ separator
642 .initial:n            = { \par }
643 }

```

(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.

```

644
645 \IfPDFManagementActiveT {

```

```

646 \str_set:Ne \l_lngx_gls_language_str {
647   \GetDocumentProperties { document / lang }
648 }
649 }

```

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

```

650
651 \keys_define:nn { lngx / gloss / format } {
652   gloss
653   .cs_gset_protected:Np = \lngx_gloss_format:n #1,
654   gloss
655   .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

```

656   expansion
657   .cs_gset_protected:Np = \lngx_expansion_format:n #1,
658   expansion
659   .initial:n           = { #1 }
660 }

```

(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.

```

661
662 \NewDocumentCommand \setupglossing { m } {
663   \keys_set:nn { lngx_glossing } { #1 }
664 }

```

(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

```

665
666 \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.

```

667   \group_begin:
668   \str_clear:N \l_tmpa_str
669   \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).

```

670   \tl_gclear_new:c {
671     g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
672   }
673   \seq_gclear_new:c {
674     g_lngx_ \str_use:N \l_tmpa_str _pages_seq
675   }
676   \tl_gset:cn {
677     g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
678   } { #2 }

```

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

```

679 \cs_new_protected:cpn {
680   __lngx_gloss_ \str_use:N \l_tmpa_str :
681 } {

```

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.

```

682   \IfDocumentMetadataTF {
683     \exp_args:Nee \UseTaggingSocket
684       { lngx / gloss }
685       { \str_use:N \l_tmpa_str }
686       { #2 }
687   } {
688     \IfPackageLoadedTF { hyperref } {
689       \group_begin:
690       \exp_args:Ne \hypersetup {
691         linkcolor = {
692           \exp_not:V \g__lngx_gloss_link_color_str
693         }
694       }
695       \IfPDFManagementActiveT {
696         \socket_if_exist:nT {
697           hyp / link / GoTo / Contents
698         } {
699           \socket_assign_plug:nn {
700             hyp / link / GoTo / Contents
701           } { text }
702         }
703       }
704       \lngx_gloss_format:n {
705         \hyperlink { lngx_ #1 _glossary } { #1 }
706       }
707       \group_end:
708     } {
709       \lngx_gloss_format:n { #1 }
710     }
711   }

```

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.

```

712   \int_gincr:N \g__lngx_page_ref_int
713   \exp_args:Ne
714   \label { lngx_gloss_ \int_use:N \g__lngx_page_ref_int }
715   \cs_if_exist:cT {

```

```

716     r @ lngx_gloss_ \int_use:N \g__lngx_page_ref_int
717 } {
718   \group_begin:
719   \tl_clear:N \l_tmpa_tl
720   \tl_set:N \l_tmpa_tl {
721     \exp_not:N \use_ii:nnnnn
722     \use:c {
723       r @ lngx_gloss_ \int_use:N \g__lngx_page_ref_int
724     }
725   }
726   \seq_if_in:cVF {
727     g_lngx_ \str_use:N \l_tmpa_str _pages_seq
728   } \l_tmpa_tl {
729     \seq_gput_right:ce {
730       g_lngx_ \str_use:N \l_tmpa_str _pages_seq
731     } {
732       \exp_not:N \use_ii:nnnnn
733       \use:c {
734         r @ lngx_gloss_ \int_use:N \g__lngx_page_ref_int
735       }
736     }
737   }
738   \group_end:
739 }
740 \seq_if_in:NeF \g__lngx_gls_use_order_seq {
741   \str_use:N \l_tmpa_str
742 } {
743   \seq_gput_right:Ne \g__lngx_gls_use_order_seq
744     { \str_use:N \l_tmpa_str }
745 }
746 }
747 \group_end:
748 }
749
750 \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.

```

751
752 \NewDocumentCommand \renewgloss { m m } {
753   \cs_undefine:c { __lngx_gloss_ #1 : }
754   \lngx_gloss_new:nn { #1 } { #2 }
755 }

```

(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.

```

756
757 \NewDocumentCommand \glx { s O{ } m } {
758   \group_begin:
759   \IfBooleanT { #1 } {
760     \bool_set_true:N \l_lngx_expansion_bool
761     \str_set:Nn \l__lngx_separator_str { expansion }
762     \keys_set:nn { lngx_glossing } {
763       separator = { , \c_space_tl }
764     }
765   }
766   \keys_set:nn { lngx_glossing } { #2 }
767   \tl_clear:N \l_tmpa_tl
768   \seq_clear:N \l_tmpa_seq
769   \seq_set_from_clist:Nn \l_tmpa_seq { #3 }
770   \seq_pop_left:NN \l_tmpa_seq \l_tmpa_tl
771   \str_set:Nc \l_tmpa_str {
772     \exp_args:Ne \str_lowercase:n {
773       \tl_use:N \l_tmpa_tl
774     }
775   }
776   \bool_if:NTF \l_lngx_expansion_bool {
777     \str_case:Vn \l__lngx_gls_expansion_case_str {
778       { lowercase } {
779         \text_lowercase:n {
780           \tl_use:c {
781             g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
782           }
783         }
784       }
785       { title~ case~ all } {
786         \text_titlecase_all:n {
787           \tl_use:c {
788             g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
789           }
790         }
791       }
792       { title~ case~ first } {
793         \text_titlecase_first:n {
794           \tl_use:c {
795             g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
796           }
797         }
798       }
799     }
800   } {
801     \use:c { __lngx_gloss_ \str_use:N \l_tmpa_str : }
802   }

```

```

803 \seq_if_empty:NF \l_tmpa_seq {
804   \seq_map_inline:Nn \l_tmpa_seq {
805     \group_begin:
806     \str_clear:N \l_tmpa_str
807     \str_set:Ne \l_tmpa_str {
808       \exp_args:Ne \str_lowercase:n { ##1 }
809     }
810     \bool_if:NTF \l_lngx_expansion_bool {
811       \str_case:Vn \l__lngx_gls_expansion_case_str {
812         { lowercase } {
813           \tl_use:N \l_lngx_expansion_separator_tl
814           \text_lowercase:n {
815             \tl_use:c {
816               g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
817             }
818           }
819         }
820         { title~ case~ all } {
821           \tl_use:N \l_lngx_expansion_separator_tl
822           \text_titlecase_all:n {
823             \tl_use:c {
824               g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
825             }
826           }
827         }
828         { title~ case~ first } {
829           \tl_use:N \l_lngx_expansion_separator_tl
830           \text_titlecase_first:n {
831             \tl_use:c {
832               g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
833             }
834           }
835         }
836       }
837     } {
838       \tl_use:N \l_lngx_gloss_separator_tl
839       \use:c { __lngx_gloss_ \str_use:N \l_tmpa_str : }
840     }
841     \group_end:
842   }
843 }
844 \group_end:
845 }

```

(End of definition for `\glx`. 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).

```

846
847 \cs_new_protected:Npn \__lngx_dotfill:nnn #1#2#3 {
848   %% Courtesy: Jonathan P. Spratte

```

```

849 %% topanswers.xyz/tex?q=8155#a7758 (LPPL)
850 \l__lngx_entry_separator_tl
851 \smallskip
852 \group_begin:
853 \rightskip          = 0pt plus -1fil \prg_do_nothing:
854 \parfillskip        = 0pt plus 1fil \prg_do_nothing:
855 \leftskip           = 1em plus 1fil \prg_do_nothing:
856 \finalhyphendemerits = 0           \prg_do_nothing:
857 \parindent          = -1em         \prg_do_nothing:
858 \bool_if:NT \l__lngx_gls_bold_bool { \textbf } {
859   \lngx_gloss_format:n {
860     #1
861   }
862   \tl_use:N \l__lngx_glossary_separator_tl
863 }
864 #2
865 \leavevmode
866 \quad
867 \IfDocumentMetadataT {
868   \tag_mc_end:
869   \tag_struct_begin:n {
870     tag                = { Span },
871     actualtext         = { }
872   }
873   \tag_mc_begin:n {
874     tag                = { Span }
875   }
876 }
877 \cleaders
878   \hbox to 0.44em { \hss . \hss }
879   \hskip 0.5cm plus 1fill \prg_do_nothing:
880 \IfDocumentMetadataT {
881   \tag_mc_end:
882   \tag_struct_end:
883   \tag_mc_begin:n { }
884 }
885 \quad
886 \kern 0pt \prg_do_nothing:
887 \em #3
888 \l__lngx_entry_separator_tl
889 \group_end:
890 }

```

(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.

```

891
892 \NewDocumentEnvironment { lngx_multicols } { m } {
893   \int_compare:nNnTF { 1 } < {
894     \int_use:N \l__lngx_glossary_columns_int
895   } {
896     \begin { multicols } {
897       \int_use:N \l__lngx_glossary_columns_int

```



```

898     } [ #1 ]
899   } { #1 }
900   \noindent
901 } {
902   \int_compare:nNnT { 1 } < {
903     \int_use:N \l__lngx_glossary_columns_int
904   } {
905     \end { multicol }
906   }
907 }

```

(End of definition for `lngx_multicol`. 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.

```

908
909 \cs_new_protected:Npn \lngx_gloss_list: {
910   \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.

```

911   \group_begin:
912   \seq_clear:N \l_tmpa_seq
913   \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_EX 3's mechanism for sorting strings.

```

914   \str_case:Vn \l__lngx_gls_sorting_order_str {
915     { alphabetical } {
916       \seq_sort:Nn \l_tmpa_seq {
917         \str_compare:nNnTF { ##1 } > { ##2 } {
918           \sort_return_swapped:
919         } {
920           \sort_return_same:
921         }
922       }
923     }
924   }

```

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.

```

925   \str_if_eq:VnTF \l__lngx_glossary_style_str { inline } {
926     \group_begin:
927     \keys_set:nn { lngx_glossing } {
928       separator      = { \c_colon_str \c_space_tl },
929       entry~ separator = { ,~ }
930     }

```

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.

```

931     \tl_clear:N \l_tmpa_tl
932     \str_clear:N \l_tmpa_str

```

```

933 \seq_pop_left:NN \l_tmpa_seq \l_tmpa_tl
934 \str_set:NV \l_tmpa_str \l_tmpa_tl
935 \tag_mc_end:
936 \tag_struct_begin:n {
937     tag = { Span },
938 }
939 \tag_mc_begin:n {
940     tag = { Span }
941 }
942 \MakeLinkTarget * {
943     lngx_ \str_use:N \l_tmpa_str _glossary
944 }
945 \bool_if:NT \l__lngx_gls_bold_bool { \textbf } {
946     \lngx_gloss_format:n {
947         \tl_use:N \l_tmpa_tl
948         \tl_use:N \l_lngx_glossary_separator_tl
949     }
950 }
951 \tag_mc_end:
952 \tag_struct_end:

```

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

```

953 \str_case:Vn \l__lngx_gls_expansion_case_str {
954     { lowercase } {
955         \lngx_expansion_format:n {
956             \text_lowercase:n {
957                 \tl_use:c {
958                     g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
959                 }
960             }
961         }
962     }
963     { title~ case~ all } {
964         \lngx_expansion_format:n {
965             \text_titlecase_all:n {
966                 \tl_use:c {
967                     g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
968                 }
969             }
970         }
971     }
972     { title~ case~ first } {
973         \lngx_expansion_format:n {
974             \text_titlecase_first:n {
975                 \tl_use:v {
976                     g_lngx_ \str_use:N \l_tmpa_str _expansion_tl
977                 }
978             }
979         }
980     }
981 }

```

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.

```

982 \seq_if_empty:NF \l_tmpa_seq {
983   \seq_map_inline:Nn \l_tmpa_seq {
984     \group_begin:
985     \tl_use:N \l__lngx_entry_separator_tl
986     \MakeLinkTarget * { lngx_ ##1 _glossary }
987     \textbf {
988       \lngx_gloss_format:n {
989         ##1
990         \tl_use:N \l_lngx_glossary_separator_tl
991       }
992     }
993     \str_case:Vn \l__lngx_gls_expansion_case_str {
994       { lowercase } {
995         \lngx_expansion_format:n {
996           \text_lowercase:n {
997             \exp_not:v { g_lngx_ ##1 _expansion_tl }
998           }
999         }
1000       }
1001       { title~ case~ all } {
1002         \lngx_expansion_format:n {
1003           \text_titlecase_all:n {
1004             \exp_not:v { g_lngx_ ##1 _expansion_tl }
1005           }
1006         }
1007       }
1008       { title~ case~ first } {
1009         \lngx_expansion_format:n {
1010           \text_titlecase_first:n {
1011             \exp_not:v { g_lngx_ ##1 _expansion_tl }
1012           }
1013         }
1014       }
1015     }
1016     \group_end:
1017   }
1018 }
1019 \group_end:
1020 } {

```

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`.

```

1021 \ProvideDocumentCommand \glossaryname { } { Glossary }

```

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

```

1022 \begin { lngx_multicols } {
1023   \str_if_eq:VnF \l__lngx_gls_sectioning_str { null } {
1024     \use:e {
1025       \exp_not:N \use:c
1026       { \str_use:N \l__lngx_gls_sectioning_str }
1027       \bool_if:NF \l__lngx_gls_section_number_bool { * }
1028       { \exp_not:N \glossaryname }

```

```

1029     }
1030   }
1031 }
1032 \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.

```

1033   \group_begin:
1034   \seq_clear:N \l_tmpb_seq
1035   \seq_map_inline:cn { g_lngx_ ##1 _pages_seq } {

```

The pages are hyperlinked with the internal PDF names.

```

1036     \seq_put_right:Ne \l_tmpb_seq { ####1 }
1037   }

```

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

```

1038   \__lngx_dotfill:nnn {
1039     \MakeLinkTarget * { lngx_ ##1 _glossary }
1040     ##1
1041   } {

```

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

```

1042     \str_case:Vn \l__lngx_gls_expansion_case_str {
1043       { lowercase } {
1044         \lngx_expansion_format:n {
1045           \text_lowercase:n {
1046             \exp_not:v { g_lngx_ ##1 _expansion_tl }
1047           }
1048         }
1049       }
1050       { title~ case~ all } {
1051         \lngx_expansion_format:n {
1052           \text_titlecase_all:n {
1053             \exp_not:v { g_lngx_ ##1 _expansion_tl }
1054           }
1055         }
1056       }
1057       { title~ case~ first } {
1058         \lngx_expansion_format:n {
1059           \text_titlecase_first:n {
1060             \exp_not:v { g_lngx_ ##1 _expansion_tl }
1061           }
1062         }
1063       }
1064     }
1065   } {

```

The list of page numbers is printed.

```

1066     \seq_use:Nn \l_tmpb_seq { ,~ }
1067   }
1068   \group_end:
1069 }
1070 \end { lngx_multicols }
1071 }

```

```

1072 \group_end:
1073 }

```

(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 L^AT_EX₃ function for printing the glosses.

```

1074
1075 \NewDocumentCommand \listofglosses { 0 { } } {
1076   \group_begin:
1077   \str_set:Nn \l__lngx_separator_str { glossary }
1078   \keys_set:nn { lngx_glossing } {
1079     separator      = { \c_colon_str \c_space_tl }
1080   }
1081   \keys_set:nn { lngx_glossing } { #1 }
1082   \lngx_gloss_list:
1083   \group_end:
1084 }
1085 </glossing>

```

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

```

1086 <*ipa>
1087 \ProvidesExplPackage{linguistix-ipa}
1088     {2026-02-02}
1089     {v0.8}
1090     {%
1091         A package for typesetting the IPA
1092         (International Phonetic Alphabet) from
1093         the ‘LinguistX’ bundle.%
1094     }

```

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).

```

1095
1096 \sys_if_engine_luatex:TF {
1097     \IfPackageLoadedF { unicode-math } {
1098         \IfPackageLoadedF { lua-unicode-math } {
1099             \RequirePackage { fontspec, lua-unicode-math }
1100         }
1101     }
1102 } { {
1103     \IfPackageLoadedF { unicode-math } {
1104         \RequirePackage { unicode-math }
1105     }
1106 }
1107
1108 \IfPackageLoadedF { linguistix-base } {
1109     \RequirePackage { linguistix-base }
1110 }
1111
1112 \IfPackageLoadedF { linguistix-nfss } {
1113     \RequirePackage { linguistix-nfss }
1114 }
1115
1116 \IfPackageLoadedF { linguistix-fixpex } {
1117     \RequirePackage { linguistix-fixpex }
1118 }

```

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

```

1119
1120 \NewDocumentCommand \ipatext { s m } {
1121     \IfBooleanTF { #1 } {
1122         {
1123             \lngxipa
1124             / #2 /
1125         }
1126     } {
1127         {
1128             \lngxipa
1129             [ #2 ]
1130         }
1131     }
1132 }

```

(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.

```

II33 \prop_gclear_new:N \g__lngx_ipa_main_fonts_prop
II34 \tl_gclear_new:N \g__lngx_ipa_main_font_features_tl
II35
II36 \clist_map_inline:nn {
II37     upright,
II38     bold~ upright,
II39     italic,
II40     bold~ italic,
II41     slanted,
II42     bold~ slanted,
II43     swash,
II44     bold~ swash,
II45     small~ caps
II46 } {
II47
All the keys here are prefixed with the word ipa in order to distinguish them from the
keys provided by the LINGUISTICX-FONTS package. These keys have identical method as
their text counterparts, though.
II48 \keys_define:nn { lngx_keys } {
II49     ipa~ #1
II50     .code:n = {
II51         \group_begin:
II52         \str_clear:N \l_tmpa_str
II53         \str_set:Ne \l_tmpa_str {
II54             \text_titlecase_all:n { #1 }
II55             Font
II56         }
II57         \str_replace_all:Nnn \l_tmpa_str { ~ } { }
II58         \prop_gput:Nne \g__lngx_ipa_main_fonts_prop
II59             { ipa~ #1 }
II60             { \str_use:N \l_tmpa_str = { ##1 } }
II61         \group_end:
II62     },
II63     ipa~ #1~ features
II64     .code:n = {
II65         \group_begin:
II66         \str_clear:N \l_tmpa_str
II67         \str_set:Ne \l_tmpa_str {
II68             \text_titlecase_all:n { #1 }
II69             Features
II70         }
II71         \str_replace_all:Nnn \l_tmpa_str { ~ } { }
II72         \prop_gput:Nne \g__lngx_ipa_main_fonts_prop
II73             { ipa~ #1~ features }
II74             {
II75                 \str_use:N \l_tmpa_str = { ##1 }
II76             }
II77         \group_end:
II78     }
II79 }
II80 }

```

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

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

```
1181
1182 \keys_define:nn { lngx_keys } {
1183   ipa~ extra~ features
1184   .prop_gput:N          = \g__lngx_ipa_main_fonts_prop
1185 }
```

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


```

\g__lngx_ipa_sans_fonts_prop
\g__lngx_ipa_sans_font_features_tl
\g__lngx_ipa_mono_fonts_prop
\g__lngx_ipa_mono_font_features_tl
ipa sans upright
ipa sans upright features
ipa sans bold upright
ipa sans bold upright features
ipa sans italic
ipa sans italic features
ipa sans bold italic
ipa sans bold italic features
ipa sans slanted
ipa sans slanted features
ipa sans bold slanted
ipa sans bold slanted features
ipa sans swash
ipa sans swash features
ipa sans bold swash
ipa sans bold swash features
ipa sans small caps
ipa sans small caps features
ipa mono upright
ipa mono upright features
ipa mono bold upright
ipa mono bold upright features
ipa mono italic
ipa mono italic features
ipa mono bold italic
ipa mono bold italic features
ipa mono slanted
ipa mono slanted features
ipa mono bold slanted
ipa mono bold slanted features
ipa mono swash
ipa mono swash features
ipa mono bold swash
ipa mono bold swash features
ipa mono small caps
ipa mono small caps features

```

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

```

1186 \prop_gclear_new:N \g__lngx_ipa_sans_fonts_prop
1187 \tl_gclear_new:N \g__lngx_ipa_sans_font_features_tl
1188 \prop_gclear_new:N \g__lngx_ipa_mono_fonts_prop
1189 \tl_gclear_new:N \g__lngx_ipa_mono_font_features_tl
1190
1191 \clist_map_inline:nn {
1192   sans,
1193   mono
1194 } {
1195   \clist_map_inline:nn {
1196     upright,
1197     bold~ upright,
1198     italic,
1199     bold~ italic,
1200     slanted,
1201     bold~ slanted,
1202     swash,
1203     bold~ swash,
1204     small~ caps
1205   } {
1206     \keys_define:nn { lngx_keys } {
1207       ipa~ #1~ ##1
1208         .code:n = {
1209           \group_begin:
1210           \str_clear:N \l_tmpa_str
1211           \str_set:Ne \l_tmpa_str {
1212             \text_titlecase_all:n { ##1 }
1213             Font
1214           }
1215           \str_replace_all:Nnn \l_tmpa_str { ~ } { }
1216           \prop_gput:cne { g__lngx_ipa_ #1 _fonts_prop }
1217             { ipa~ #1~ ##1 }
1218             { #####1 }
1219           \group_end:
1220         },
1221       ipa~ #1~ ##1~ features
1222         .code:n = {
1223           \group_begin:
1224           \str_clear:N \l_tmpa_str
1225           \str_set:Ne \l_tmpa_str {
1226             \text_titlecase_all:n { #1 }
1227             Features
1228           }
1229           \str_replace_all:Nnn \l_tmpa_str { ~ } { }
1230           \prop_gput:cne { g__lngx_ipa_ #1 _fonts_prop }
1231             { ipa~ #1~ ##1~ features }
1232             {
1233               \str_use:N \l_tmpa_str = { #####1 }
1234             }
1235           \group_end:
1236         }
1237     }

```

```

1238     }
1239   }
1240   \keys_define:nn { lngx_keys } {
1241     ipa~ #1~ extra~ features
1242     .prop_gput:c          = {
1243                           g__lngx_ipa_ #1 _fonts_prop
1244                           }
1245   }
1246 }

```

(End of definition for `\g__lngx_ipa_sans_fonts_prop` and others. These functions are documented on page 12.)

`\g__lngx_ipa_main_font_tl` These keys provide keys to set fonts for IPA.

```

\g__lngx_ipa_sans_font_tl
\g__lngx_ipa_mono_font_tl
ipa main font
ipa sans font
ipa mono font
1247
1248 \clist_map_inline:nn {
1249   main,
1250   sans,
1251   mono
1252 } {
1253   \keys_define:nn { lngx_keys } {
1254     ipa~ #1~ font
1255     .tl_gset:c          = { g__lngx_ipa_ #1 _font_tl }
1256   }
1257 }

```

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

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.

```

1258
1259 \keys_define:nn { lngx_keys } {
1260   ipa~ newcm
1261   .meta:n          = {
1262     ipa~ extra~
1263     features        = {
1264       Renderer      = {HarfBuzz},
1265       StylisticSet  = {05}
1266     },
1267     ipa~ sans~ extra~
1268     features        = {
1269       Renderer      = {HarfBuzz},
1270       StylisticSet  = {05}
1271     },
1272     ipa~ mono~ extra~
1273     features        = {
1274       Renderer      = {HarfBuzz},
1275       StylisticSet  = {05}
1276     },
1277     ipa~ main~ font = { NewCM10-Book.otf },
1278     ipa~ sans~ font = { NewCMSans10-Book.otf },
1279     ipa~ mono~ font = { NewCMMono10-Book.otf }

```

```

1280 }
1281 }

```

(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.

```

1282
1283 \keys_define:nn { lngx_keys } {
1284   ipa~ newcm~ sans
1285   .meta:n          = {
1286     ipa~ extra~
1287     features        = {
1288       Renderer      = {HarfBuzz},
1289       StylisticSet  = {05}
1290     },
1291     ipa~ sans~ extra~
1292     features        = {
1293       Renderer      = {HarfBuzz},
1294       StylisticSet  = {05}
1295     },
1296     ipa~ mono~ extra~
1297     features        = {
1298       Renderer      = {HarfBuzz},
1299       StylisticSet  = {05}
1300     },
1301     ipa~ main~ font = { NewCMSans10-Book.otf },
1302     ipa~ sans~ font = { NewCMSans10-Book.otf },
1303     ipa~ mono~ font = { NewCMMono10-Book.otf }
1304   }
1305 }

```

(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.

```

1306
1307 \keys_define:nn { lngx_keys } {
1308   ipa~ newcm~ mono
1309   .meta:n          = {
1310     ipa~ extra~
1311     features        = {
1312       Renderer      = {HarfBuzz},
1313       StylisticSet  = {05}
1314     },
1315     ipa~ sans~ extra~
1316     features        = {
1317       Renderer      = {HarfBuzz},
1318       StylisticSet  = {05}
1319     },
1320     ipa~ mono~ extra~
1321     features        = {
1322       Renderer      = {HarfBuzz},
1323       StylisticSet  = {05}
1324     },
1325     ipa~ main~ font = { NewCMMono10-Book.otf },
1326     ipa~ sans~ font = { NewCMSans10-Book.otf },

```

```

1327     ipa~ mono~ font      = { NewCMMono10-Book.otf }
1328   }
1329 }

```

(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.

```

1330
1331 \keys_define:nn { lngx_keys } {
1332   ipa~ newcm~ regular
1333   .meta:n      = {
1334     ipa~ extra~
1335     features    = {
1336       Renderer  = {HarfBuzz},
1337       StylisticSet = {05}
1338     },
1339     ipa~ sans~ extra~
1340     features    = {
1341       Renderer  = {HarfBuzz},
1342       StylisticSet = {05}
1343     },
1344     ipa~ mono~ extra~
1345     features    = {
1346       Renderer  = {HarfBuzz},
1347       StylisticSet = {05}
1348     },
1349     ipa~ main~ font      = { NewCM10-Regular.otf },
1350     ipa~ sans~ font      = { NewCMSans10-Regular.otf },
1351     ipa~ mono~ font      = { NewCMMono10-Regular.otf }
1352   }
1353 }

```

(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.

```

1354
1355 \keys_define:nn { lngx_keys } {
1356   ipa~ newcm~ regular~ sans
1357   .meta:n      = {
1358     ipa~ extra~
1359     features    = {
1360       Renderer  = {HarfBuzz},
1361       StylisticSet = {05}
1362     },
1363     ipa~ sans~ extra~
1364     features    = {
1365       Renderer  = {HarfBuzz},
1366       StylisticSet = {05}
1367     },
1368     ipa~ mono~ extra~
1369     features    = {
1370       Renderer  = {HarfBuzz},

```

```

1371     StylisticSet           = {05}
1372   },
1373   ipa~ main~ font         = { NewCMSans10-Regular.otf },
1374   ipa~ sans~ font         = { NewCMSans10-Regular.otf },
1375   ipa~ mono~ font         = { NewCMMono10-Regular.otf }
1376 }
1377 }

```

(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.

```

1378
1379 \keys_define:nn { lngx_keys } {
1380   ipa~ newcm~ regular~ mono
1381   .meta:n           = {
1382     ipa~ extra~
1383     features         = {
1384       Renderer       = {HarfBuzz},
1385       StylisticSet   = {05}
1386     },
1387     ipa~ sans~ extra~
1388     features         = {
1389       Renderer       = {HarfBuzz},
1390       StylisticSet   = {05}
1391     },
1392     ipa~ mono~ extra~
1393     features         = {
1394       Renderer       = {HarfBuzz},
1395       StylisticSet   = {05}
1396     },
1397     ipa~ main~ font  = { NewCMMono10-Regular.otf },
1398     ipa~ sans~ font  = { NewCMSans10-Regular.otf },
1399     ipa~ mono~ font  = { NewCMMono10-Regular.otf }
1400   }
1401 }

```

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

We set the *ipa newcm* key by default.

```

1402
1403 \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) at several other places, so use them with caution.

lngx_ipa_rm_nfss

\lngx_set_sans_ipa_font:nn

\lngx_sans_ipa:

lngx_ipa_sf_nfss

```

1404
1405 \cs_new_protected:Npn \lngx_set_main_ipa_font:nn #1#2 {
1406   \setfontfamily \lngx_main_ipa: [
1407     #1,
1408     NFSSFamily           = { lngx_ipa_rm_nfss }
1409   ] { #2 }
1410 }

```

```

I411
I412 \cs_new_protected:Npn \lngx_set_sans_ipa_font:nn #1#2 {
I413   \setfontfamily \lngx_sans_ipa: [
I414     #1,
I415     NFSSFamily           = { lngx_ipa_sf_nfss }
I416   ] { #2 }
I417 }
I418
I419 \cs_new_protected:Npn \lngx_set_mono_ipa_font:nn #1#2 {
I420   \setfontfamily \lngx_mono_ipa: [
I421     #1,
I422     NFSSFamily           = { lngx_ipa_tt_nfss }
I423   ] { #2 }
I424 }
I425
I426 \cs_generate_variant:Nn \lngx_set_main_ipa_font:nn { VV }
I427 \cs_generate_variant:Nn \lngx_set_sans_ipa_font:nn { VV }
I428 \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_{CI}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.

```

I429
I430 \lngx_super_font_family:nn { lngx_ipa } {
I431   rm           = { lngx_ipa_rm_nfss },
I432   sf           = { lngx_ipa_sf_nfss },
I433   tt           = { lngx_ipa_tt_nfss }
I434 }

```

(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_{CI}X-NFSS for defining this switch to the IPA.

```

I435
I436 \cs_new_protected:Npn \lngx_ipa: {
I437   \lngx softer_super_font_family:n { lngx_ipa }
I438 }
I439
I440 \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_{CI}X-FONTS for setting the size variants. This is done with lazy evaluation, just before `\begin{document}`.

```

I441
I442 \clist_map_inline:nn {
I443   main,
I444   sans,
I445   mono
I446 } {
I447   \cs_new_protected:cpn {

```

```

1448     lngx_build_ #1 _ipa_font_features:
1449 } {
1450     \prop_map_inline:cn { g__lngx_ipa_ #1 _fonts_prop } {
1451         \tl_gput_right:cn {
1452             g__lngx_ipa_ #1 _font_features_tl
1453         } { ####2 }
1454     }
1455 }
1456 }
1457
1458 \hook_gput_code:nnn { begindocument / before } { . } {
1459     \lngx_build_main_ipa_font_features:
1460     \lngx_set_main_ipa_font:VV
1461     \g__lngx_ipa_main_font_features_tl
1462     \g__lngx_ipa_main_font_tl
1463     \lngx_build_sans_ipa_font_features:
1464     \lngx_set_sans_ipa_font:VV
1465     \g__lngx_ipa_sans_font_features_tl
1466     \g__lngx_ipa_sans_font_tl
1467     \lngx_build_mono_ipa_font_features:
1468     \lngx_set_mono_ipa_font:VV
1469     \g__lngx_ipa_mono_font_features_tl
1470     \g__lngx_ipa_mono_font_tl
1471 }
1472 </ipa>

```

```

1473 (*lang)
1474 \ProvidesExplPackage{linguistix-languages}
1475     {2026-02-02}
1476     {v0.8}
1477     {%
1478     An assistant package for automatically
1479     loading fonts and more settings for
1480     languages.%
1481     }

```

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

```

1482
1483 \IfPackageLoadedF { linguistix-base } {
1484   \RequirePackage { linguistix-base }
1485 }

```

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

```

1486
1487 \IfPackageLoadedF { babel } {
1488   \RequirePackage [ provide * = * ] { babel }
1489 }

```

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

```

1490
1491 \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.

```

1492
1493 \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`

```

1494
1495 \cs_new_protected:Npn \lngx_languages:nn #1#2 {
1496   \babelprovide [ #1 ] { #2 }
1497 }
1498
1499 \cs_generate_variant:Nn \lngx_languages:nn { VV }
1500 \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.


```

1501
1502 \IfPackageLoadedTF { linguistix-fonts } {
1503   \cs_gset_protected:Npn \lngx_set_main_font:nn #1#2 {
1504     \bafont { rm } [ #1 ] { #2 }
1505   }
1506   \cs_gset_protected:Npn \lngx_set_sans_font:nn #1#2 {
1507     \bafont { sf } [ #1 ] { #2 }
1508   }
1509   \cs_gset_protected:Npn \lngx_set_mono_font:nn #1#2 {
1510     \bafont { tt } [ #1 ] { #2 }
1511   }
1512 } {
1513   \cs_new_protected:Npn \lngx_set_main_font:nn #1#2 {
1514     \bafont { rm } [ #1 ] { #2 }
1515   }
1516   \cs_new_protected:Npn \lngx_set_sans_font:nn #1#2 {
1517     \bafont { sf } [ #1 ] { #2 }
1518   }
1519   \cs_new_protected:Npn \lngx_set_mono_font:nn #1#2 {
1520     \bafont { tt } [ #1 ] { #2 }
1521   }
1522 }

```

`\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`

```

1523
1524 \cs_gset_protected:Npn \lngx_other_main_font:nnn #1#2#3 {
1525   \bafont [ #1 ] { rm } [ #2 ] { #3 }
1526 }
1527
1528 \cs_gset_protected:Npn \lngx_other_sans_font:nnn #1#2#3 {
1529   \bafont [ #1 ] { sf } [ #2 ] { #3 }
1530 }
1531
1532 \cs_gset_protected:Npn \lngx_other_mono_font:nnn #1#2#3 {
1533   \bafont [ #1 ] { tt } [ #2 ] { #3 }
1534 }
1535
1536 \cs_generate_variant:Nn \lngx_other_main_font:nnn { nee }
1537 \cs_generate_variant:Nn \lngx_other_sans_font:nnn { nee }
1538 \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.

```

1539
1540 \cs_new_protected:Npn \lngx_load_languages:n #1 {
1541   \lngx_set_keys:n { languages = { #1 } }
1542 }
1543
1544 \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`

```

1545
1546 \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`.

```

1547
1548 \cs_set:Npn \thechapter {
1549   \lngx_counter:n { chapter }
1550 }
1551 \cs_set:Npn \thesection {
1552   \lngx_counter:n { section }
1553 }
1554 \cs_set:Npn \thesubsection {
1555   \lngx_counter:n { subsection }
1556 }
1557 \cs_set:Npn \thesubsubsection {
1558   \lngx_counter:n { subsubsection }
1559 }
1560 \cs_set:Npn \theparagraph {
1561   \lngx_counter:n { section }
1562 }
1563 \cs_set:Npn \thesubparagraph {
1564   \lngx_counter:n { section }
1565 }
1566 \cs_set:Npn \thepage {
1567   \lngx_counter:n { page }
1568 }
1569 \cs_set:Npn \thefigure {
1570   \lngx_counter:n { figure }
1571 }
1572 \cs_set:Npn \thetable {
1573   \lngx_counter:n { table }
1574 }
1575 \cs_set:Npn \thefootnote {
1576   \lngx_counter:n { footnote }
1577 }
1578 \cs_set:Npn \thempfootnote {
1579   \lngx_counter:n { mpfootnote }
1580 }
1581 \cs_set:Npn \theequation {
1582   \lngx_counter:n { equation }
1583 }

```

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

```

1584
1585 \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).

```

1586
1587 \socket_new_plug:nnn { lngx / native-numbering }
1588         { strict } {
1589     \cs_gset_eq:Nc \lngx_counter:n {
1590         \tl_use:N \g_lngx_main_language_tl counter
1591     }
1592 }

```

(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).

```

1593
1594 \socket_new_plug:nnn { lngx / native-numbering }
1595         { logical } {
1596     \cs_gset_protected:Npn \lngx_counter:n ##1 {
1597         \localecounter { digits } { ##1 }
1598     }
1599 }

```

(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.

```

1600
1601 \socket_new_plug:nnn { lngx / native-numbering } { off } {
1602     \cs_gset_eq:NN \lngx_counter:n \arabic
1603 }

```

(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.

```

1604
1605 \cs_generate_variant:Nn \socket_assign_plug:nn { ne }
1606
1607 \keys_define:nn { lngx_keys } {
1608     native~ numbering
1609     .choices:nn      = { strict,logical,off } {
1610         \socket_assign_plug:ne { lngx / native-numbering } {
1611             \str_use:N \l_keys_choice_str
1612         }
1613         \socket_use:n { lngx / native-numbering }
1614     },

```

Similarly, we set the default value to on.

```

1615     native~ numbering
1616     .default:n       = { strict }
1617 }

```

(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.

```

r618
r619 \cs_new_protected:Npn \lngx_misc_reset: {
r620   \cs_set:Npn \theenumii { \alph { enumii } }
r621   \cs_set:Npn \labelenumii { ( \theenumii ) }
r622   \cs_set:Npn \theenumiii { \roman { enumiii } }
r623   \cs_set:Npn \labelenumiii { \theenumiii . }
r624   \cs_set:Npn \theenumiv { \Alph { enumiv } }
r625   \cs_set:Npn \labelenumiv { \theenumiv . }
r626   \IfPackageLoadedT { expex } {
r627     \lingset { labeltype = alpha }
r628   }
r629   \cs_gset_eq:NN \emph \textit
r630 }

```

(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.

```

r631
r632 \msg_new:nnn { linguistix-languages } { no_support } {
r633   '#1'~ is~ not~ supported.\
r634   If~ you~ want~ it~ to~ be~ supported,~ please~ report~
r635   to~ the~ maintainers.
r636 }

```

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

```

r637
r638 \keys_define:nn { lngx_keys } {
r639   languages
r640   .code:n          = {

```

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

```

r641   \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.

```

r642   \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.

```

r643   \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.

```

r644   \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.

```

r645   \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.

```

1646      \tl_set:Ne \l_tmpb_tl {
1647          main,

```

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

```

1648      import
1649  }

```

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

```

1650      \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.

```

1651      \file_if_exist:nTF { linguistix - \l_tmpa_tl . sty } {
1652          \exp_args:Ne \RequirePackage
1653              { linguistix - \l_tmpa_tl }
1654      } {

```

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.

```

1655          \msg_warning:nnV { linguistix-languages }
1656                          { no_support }
1657                          \l_tmpa_tl
1658      }

```

The temporary `tls` are cleared.

```

1659      \tl_clear:N \l_tmpa_tl
1660      \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.

```

1661      \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`.

```

1662      \clist_map_inline:Nn \l_tmpa_clist {
1663          \clist_pop:NN \l_tmpa_clist \l_tmpa_tl
1664          \tl_set:Ne \l_tmpb_tl { import }
1665          \lngx_languages:VV \l_tmpb_tl \l_tmpa_tl
1666          \file_if_exist:nTF {
1667              linguistix - \l_tmpa_tl . sty
1668          } {
1669              \exp_args:Ne \RequirePackage
1670                  { linguistix - \l_tmpa_tl }
1671          } {
1672              \msg_warning:nnV { linguistix-languages }
1673                          { no_ldf }
1674                          \l_tmpa_tl
1675          }
1676          \tl_clear:N \l_tmpa_tl
1677          \tl_clear:N \l_tmpb_tl
1678      }
1679  }

```

```

I680     }
I681   }
I682 }
I683 </lang>

```

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

```

1684 (*logos)
1685 \ProvidesExplPackage{linguistix-logos}
1686     {2026-02-02}
1687     {v0.8}
1688     {%
1689     Logos of the ‘Linguistix’ bundle.%
1690     }

```

The fontspec package (if not already loaded).

```

1691
1692 \IfPackageLoadedF { fontspec } {
1693     \RequirePackage { fontspec }
1694 }

```

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

```

1695
1696 \newfontfamily \lngx_logo_font: [
1697     UprightFont          = { NewCMUncial10-Book.otf },
1698     UprightFeatures      = {
1699         SizeFeatures      = {
1700             {
1701                 Size          = {-8},
1702                 Font          = {NewCMUncial08-Book.otf}
1703             },
1704             {
1705                 Size          = {8-},
1706                 Font          = {NewCMUncial10-Book.otf}
1707             },
1708         }
1709     },
1710     BoldFont             = { NewCMUncial10-Bold.otf },
1711     BoldFeatures         = {
1712         SizeFeatures      = {
1713             {
1714                 Size          = {-8},
1715                 Font          = {NewCMUncial08-Bold.otf}
1716             },
1717             {
1718                 Size          = {8-},
1719                 Font          = {NewCMUncial10-Bold.otf}
1720             },
1721         }
1722     }
1723 ]{ 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`.

```

1724
1725 \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.

```

1726
1727 \NewDocumentCommand \lngxlogo { 0{} } {%
1728   \group_begin:
1729   \lngx_logo_font:
1730   LinguisTi
1731   \color_group_begin:
1732   \color_select:n { lngx_purple_color }
1733   X
1734   \color_group_end:
1735   \IfBlankF { #1 } { - #1 }
1736   \group_end:
1737 }

```

(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.

```

1738
1739 \cs_new:Npn \lngxpkg {
1740   \IfPackageLoadedTF { hyperref } {
1741     \texorpdfstring {
1742       \lngxlogo
1743     } {
1744       LinguisTiX
1745     }
1746   } {
1747     \lngxlogo
1748   }
1749 }

```

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

```

1750
1751 \clist_map_inline:nn {
1752   base,examples,fixpex,fonts,ipa,languages,logos,nfss,
1753   marathi,british,american,english,greek,malayalam,glossing,
1754   leipzig
1755 } {

```

`#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.

```

1756   \cs_new:cpn { lngx #1 logo } {
1757     \texorpdfstring {
1758       \lngxlogo [ #1 ]
1759     } {
1760       LinguisTiX - #1
1761     }
1762   }
1763 }
1764 </logos>

```

LINGUIS*Ti*X-NFSS

Documentation | L^AT_EX₃-interface

```

1765 < *nfss >

```



```

1766 \ProvidesExplPackage{linguistix-nfss}
1767         {2026-02-02}
1768         {v0.8}
1769         {%
1770         An extension to the core NFSS commands
1771         from the ‘LinguisTiX’ bundle.%
1772     }

```

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.

```

1773
1774 \tl_new:N \l__lngx_normalfont_tmp_tl
1775 \tl_new:N \l__lngx_selectfont_tmp_tl
1776 \tl_new:N \l__lngx_family_tmp_tl
1777 \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.

```

1778
1779 \tl_new:N \l_lngx_current_encoding_tl
1780 \tl_new:N \l_lngx_current_meta_family_tl
1781 \tl_new:N \l_lngx_current_super_family_tl
1782 \tl_new:N \l_lngx_current_series_tl
1783 \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`

```

1784
1785 \hook_gput_code:nnn { begindocument / end } { . } {
1786     \tl_const:Nc \c_lngx_default_rmdefault_tl { \rmdefault }
1787     \tl_const:Nc \c_lngx_default_sfdefault_tl { \sfdefault }
1788     \tl_const:Nc \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`
`\l_lngx_current_super_family_tl` `\tl_set:Nn \l_lngx_current_super_family_tl { default }`
`\l_lngx_current_series_tl` The current encoding is saved in the relevant t_l.
`\l_lngx_current_shape_tl` `\tl_set:Nc \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.

```

1793 \tl_set:Nc \l_lngx_current_meta_family_tl {
1794     \@currentmetafamily % new from ltnews42, thanks Frank!
1795 }

```

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

```

1796 \tl_set:Nn \l_lngx_current_series_tl { md }
1797 \tl_set:Nn \l_lngx_current_shape_tl { up }
1798 }

```

(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`.

```

1799
1800 \hook_gput_code:nnn { cmd / selectfont / before } { . } {
1801   \tl_set:Nc \l__lngx_selectfont_tmp_tl { \f@encoding }
1802 }

```

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`.

```

1803
1804 \hook_gput_code:nnn { cmd / selectfont / after } { . } {
1805   \tl_set_eq:NN \l_lngx_current_encoding_tl
1806               \l__lngx_selectfont_tmp_tl
1807   \tl_clear:N   \l__lngx_selectfont_tmp_tl
1808 }

```

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.

```

1809
1810 \hook_gput_code:nnn { cmd / rmfamily / before } { . } {
1811   \tl_set:Nn \l_lngx_current_meta_family_tl { rm }
1812   \tl_set:Nc \l__lngx_family_tmp_tl { \f@encoding }
1813 }
1814
1815 \hook_gput_code:nnn { cmd / rmfamily / after } { . } {
1816   \tl_set:Nn \l_lngx_current_meta_family_tl { rm }
1817   \tl_set_eq:NN \l_lngx_current_encoding_tl
1818               \l__lngx_family_tmp_tl
1819   \tl_clear:N   \l__lngx_family_tmp_tl
1820 }
1821
1822 \hook_gput_code:nnn { cmd / sffamily / before } { . } {
1823   \tl_set:Nn \l_lngx_current_meta_family_tl { sf }
1824   \tl_set:Nc \l__lngx_family_tmp_tl { \f@encoding }
1825 }
1826
1827 \hook_gput_code:nnn { cmd / sffamily / after } { . } {
1828   \tl_set:Nn \l_lngx_current_meta_family_tl { sf }
1829   \tl_set_eq:NN \l_lngx_current_encoding_tl
1830               \l__lngx_family_tmp_tl
1831   \tl_clear:N   \l__lngx_family_tmp_tl
1832 }
1833
1834 \hook_gput_code:nnn { cmd / ttfamily / before } { . } {
1835   \tl_set:Nn \l_lngx_current_meta_family_tl { tt }
1836   \tl_set:Nc \l__lngx_family_tmp_tl { \f@encoding }
1837 }
1838
1839 \hook_gput_code:nnn { cmd / ttfamily / after } { . } {

```

```

1840 \tl_set:Nn \l_lngx_current_meta_family_tl { tt }
1841 \tl_set_eq:NN \l_lngx_current_encoding_tl
1842         \l__lngx_family_tmp_tl
1843 \tl_clear:N \l__lngx_family_tmp_tl
1844 }

```

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.

```

1845
1846 \hook_gput_code:nnn { cmd / mdseries / after } { . } {
1847   \tl_set:Nn \l_lngx_current_series_tl { md }
1848 }
1849
1850 \hook_gput_code:nnn { cmd / bfseries / after } { . } {
1851   \tl_set:Nn \l_lngx_current_series_tl { bf }
1852 }

```

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

```

1853
1854 \hook_gput_code:nnn { cmd / upshape / after } { . } {
1855   \tl_set:Nn \l_lngx_current_shape_tl { up }
1856 }
1857
1858 \hook_gput_code:nnn { cmd / itshape / after } { . } {
1859   \tl_set:Nn \l_lngx_current_shape_tl { it }
1860 }
1861
1862 \hook_gput_code:nnn { cmd / scshape / after } { . } {
1863   \tl_set:Nn \l_lngx_current_shape_tl { sc }
1864 }
1865
1866 \hook_gput_code:nnn { cmd / sscshape / after } { . } {
1867   \tl_set:Nn \l_lngx_current_shape_tl { ssc }
1868 }
1869
1870 \hook_gput_code:nnn { cmd / slshape / after } { . } {
1871   \tl_set:Nn \l_lngx_current_shape_tl { sl }
1872 }
1873
1874 \hook_gput_code:nnn { cmd / swshape / after } { . } {
1875   \tl_set:Nn \l_lngx_current_shape_tl { sw }
1876 }
1877
1878 \hook_gput_code:nnn { cmd / ulcshape / after } { . } {
1879   \tl_set:Nn \l_lngx_current_shape_tl { ulc }
1880 }

```

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

```

1881
1882 \prg_new_conditional:Nnn \lngx_if_encoding:n {
1883   p,
1884   T,

```

```

1885   F,
1886   TF
1887 } {
1888   \tl_if_eq:NnTF \l_lngx_current_encoding_tl { #1 } {
1889     \prg_return_true:
1890   } {
1891     \prg_return_false:
1892   }
1893 }
1894

```

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

`\IfEncodingTF` For non- \LaTeX 3 contexts, these simpler alternatives are provided.
`\IfEncodingT`
`\IfEncodingF`

```

1895
1896 \cs_new_eq:NN \IfEncodingTF \lngx_if_encoding:nTF
1897 \cs_new_eq:NN \IfEncodingT  \lngx_if_encoding:nT
1898 \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`

```

1899
1900 \prg_new_conditional:Nnn \lngx_if_meta_family:n {
1901   P,
1902   T,
1903   F,
1904   TF
1905 } {
1906   \tl_if_eq:NnTF \l_lngx_current_meta_family_tl { #1 } {
1907     \prg_return_true:
1908   } {
1909     \prg_return_false:
1910   }
1911 }

```

(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`

```

1912
1913 \cs_new_eq:NN \IfMetaFamilyTF \lngx_if_meta_family:nTF
1914 \cs_new_eq:NN \IfMetaFamilyT  \lngx_if_meta_family:nT
1915 \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`

```

1916
1917 \prg_new_conditional:Nnn \lngx_if_super_family:n {
1918   P,
1919   T,
1920   F,
1921   TF

```

```

1922 } {
1923   \tl_if_eq:NnTF \l_lngx_current_super_family_tl { #1 } {
1924     \prg_return_true:
1925   } {
1926     \prg_return_false:
1927   }
1928 }

```

(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`

```

1929
1930 \cs_new_eq:NN \IfSuperFamilyTF \lngx_if_super_family:nTF
1931 \cs_new_eq:NN \IfSuperFamilyT \lngx_if_super_family:nT
1932 \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`

```

1933
1934 \prg_new_conditional:Nnn \lngx_if_series:n {
1935   P,
1936   T,
1937   F,
1938   TF
1939 } {
1940   \tl_if_eq:NnTF \l_lngx_current_series_tl { #1 } {
1941     \prg_return_true:
1942   } {
1943     \prg_return_false:
1944   }
1945 }

```

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

`\IfSeriesTF` Its user-side macros.

`\IfSeriesT`

`\IfSeriesF`

```

1946
1947 \cs_new_eq:NN \IfSeriesTF \lngx_if_series:nTF
1948 \cs_new_eq:NN \IfSeriesT \lngx_if_series:nT
1949 \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`

```

1950
1951 \prg_new_conditional:Nnn \lngx_if_shape:n {
1952   P,
1953   T,
1954   F,
1955   TF
1956 } {
1957   \tl_if_eq:NnTF \l_lngx_current_shape_tl { #1 } {
1958     \prg_return_true:

```

```

1959 } {
1960   \prg_return_false:
1961 }
1962 }

```

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

\IfShapeTF User-side macros for the same.

```

1963 \IfShapeT
1964 \IfShapeF
1965 \cs_new_eq:NN \IfShapeTF \lngx_if_shape:nTF
1966 \cs_new_eq:NN \IfShapeT \lngx_if_shape:nT
1967 \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.

```

1967
1968 \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.

\lngx_if_meta_family_sf_p:

\lngx_if_meta_family_sf:TF

\lngx_if_meta_family_tt_p:

\lngx_if_meta_family_tt:TF

```

1969
1970 \clist_map_inline:nn {
1971   rm,
1972   sf,
1973   tt
1974 } {
1975   \prg_new_conditional:cnn { lngx_if_meta_family_ #1 : } {
1976     p, T, F, TF
1977   } {
1978     \tl_if_eq:NnTF \l_lngx_current_meta_family_tl { #1 } {
1979       \prg_return_true:
1980     } {
1981       \prg_return_false:
1982     }
1983   }
1984 }

```

(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.

\lngx_if_series_md:TF

\lngx_if_series_bf_p:

\lngx_if_series_bf:TF

```

1985
1986 \clist_map_inline:nn {
1987   md,
1988   bf
1989 } {
1990   \prg_new_conditional:cnn { lngx_if_series_ #1 : } {
1991     p, T, F, TF
1992   } {
1993     \tl_if_eq:NnTF \l_lngx_current_series_tl { #1 } {
1994       \prg_return_true:
1995     } {

```

```

1996     \prg_return_false:
1997   }
1998 }
1999 }

```

(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:TF
\lngx_if_shape_it_p:
\lngx_if_shape_it:TF
\lngx_if_shape_sc_p:
\lngx_if_shape_sc:TF
\lngx_if_shape_ssc_p:
\lngx_if_shape_ssc:TF
\lngx_if_shape_sl_p:
\lngx_if_shape_sl:TF
\lngx_if_shape_sw_p:
\lngx_if_shape_sw:TF
\lngx_if_shape_ulc_p:
\lngx_if_shape_ulc:TF
2000
2001 \clist_map_inline:nn {
2002   up,
2003   it,
2004   sc,
2005   ssc,
2006   sl,
2007   sw,
2008   ulc
2009 } {
2010   \prg_new_conditional:cnn { lngx_if_shape_ #1 : } {
2011     p, T, F, TF
2012   } {
2013     \tl_if_eq:NnTF \l_lngx_current_shape_tl { #1 } {
2014       \prg_return_true:
2015     } {
2016       \prg_return_false:
2017     }
2018   }
2019 }

```

(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.

```

2020
2021 \clist_map_inline:nn {
2022   rm,
2023   sf,
2024   tt
2025 } {
2026   \keys_define:nn { lngx_nfss } {
2027     #1
2028     .code:n = {
2029       \tl_gclear_new:c {
2030         g_lngx_ \l__lngx_nfss_tmp_tl _ #1 default _tl
2031       }
2032       \tl_gset:cn {
2033         g_lngx_ \l__lngx_nfss_tmp_tl _ #1 default _tl
2034       } { ##1 }
2035     }
2036   }
2037 }

```

`\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`

```
2038
2039 \cs_new_protected:Npn \lngx_super_font_family:nn #1#2 {
2040   \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.

```
2041   \keys_set:nn { lngx_nfss } { #2 }
2042   \tl_clear:N \l__lngx_nfss_tmp_tl
2043 }
2044
2045 \cs_gset_eq:NN \superfontfamily
2046               \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`

```
2047
2048 \cs_new_protected:Npn \lngx_soft_super_font_family:nn #1#2 {
2049   \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.

```
2050   \clist_map_inline:nn {
2051     rm,
2052     sf,
2053     tt
2054   } {
2055     \tl_if_empty:cF { g_lngx_ #1 _ ##1 default_tl } {
2056       \cs_set:cpe { ##1 default } {
2057         \tl_use:c { g_lngx_ #1 _ ##1 default _tl }
2058       }
2059     }
2060   }
```

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

```
2061   \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.

```
2062   \clist_map_inline:nn { #2 } {
2063     \str_case:nn { ##1 } {
2064       { encoding } {
2065         \exp_args:NV \fontencoding
2066           \l__lngx_current_encoding_tl
2067       }
2068       { family } {
2069         \use:c {
2070           \l__lngx_current_meta_family_tl family
```



```

2071     }
2072     \exp_args:NV \fontencoding
2073                 \l_lngx_current_encoding_tl
2074     \selectfont
2075 }
2076 { series } {
2077     \use:c {
2078         \l_lngx_current_series_tl series
2079     }
2080 }
2081 { shape } {
2082     \use:c {
2083         \l_lngx_current_shape_tl shape
2084     }
2085 }
2086 }
2087 }
2088 }
2089
2090 \cs_gset_eq:NN \softsuperfontfamily
2091                 \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_softest_super_font_family:n` This function excludes the encoding and resets all the other attributes. It comes with a user-side macro.

`\softersuperfontfamily`

```

2092
2093 \cs_new_protected:Npn \lngx_softest_super_font_family:n #1 {
2094     \lngx_soft_super_font_family:nn { #1 } {
2095         family,
2096         series,
2097         shape
2098     }
2099 }
2100
2101 \cs_gset_eq:NN \softersuperfontfamily
2102                 \lngx_softest_super_font_family:n

```

(End of definition for `\lngx_softest_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.

`\softestsuperfontfamily`

```

2103
2104 \cs_new_protected:Npn \lngx_softest_super_font_family:n #1 {
2105     \lngx_soft_super_font_family:nn { #1 } {
2106         encoding,
2107         family,
2108         series,
2109         shape
2110     }
2111 }
2112
2113 \cs_gset_eq:NN \softestsuperfontfamily
2114                 \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`

```

2115
2116 \cs_new_protected:Npn \lngx_soft_normal_font:n #1 {
2117   \tl_set:Nc \l_lngx_current_super_family_tl { default }
2118   \clist_map_inline:nn {
2119     rm,
2120     sf,
2121     tt
2122   } {
2123     \cs_set:cpe { ##1 default } {
2124       \tl_use:c { c_lngx_default_ ##1 default _tl }
2125     }
2126   }
2127   \normalfont
2128   \clist_map_inline:nn { #1 } {
2129     \str_case:nn { ##1 } {
2130       { encoding } {
2131         \exp_args:NV \fontencoding
2132           \l_lngx_current_encoding_tl
2133       }
2134       { family } {
2135         \use:c {
2136           \l_lngx_current_meta_family_tl family
2137         }
2138         \exp_args:NV \fontencoding
2139           \l_lngx_current_encoding_tl
2140         \selectfont
2141       }
2142       { series } {
2143         \use:c {
2144           \l_lngx_current_series_tl series
2145         }
2146       }
2147       { shape } {
2148         \use:c {
2149           \l_lngx_current_shape_tl shape
2150         }
2151       }
2152     }
2153   }
2154 }
2155
2156 \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`

```

2157
2158 \cs_new_protected:Npn \lngx_softer_normal_font: {

```

```

2159 \ltx_soft_normal_font:n {
2160     family,
2161     series,
2162     shape
2163 }
2164 }
2165
2166 \cs_gset_eq:NN \softernormalfont \ltx softer_normal_font:

```

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

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

```

2167
2168 \cs_new_protected:Npn \ltx_softest_normal_font: {
2169     \ltx_soft_normal_font:n {
2170         encoding,
2171         family,
2172         series,
2173         shape
2174     }
2175 }
2176
2177 \cs_gset_eq:NN \softestnormalfont \ltx softest_normal_font:

```

(End of definition for `\ltx 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.

```

2178 \cs_new:Npn \CurrentEncoding {
2179     \tl_use:N \l_ltx_current_encoding_tl
2180 }
2181 \cs_new:Npn \CurrentMetaFamily {
2182     \tl_use:N \l_ltx_current_meta_family_tl
2183 }
2184 \cs_new:Npn \CurrentSuperFamily {
2185     \tl_use:N \l_ltx_current_super_family_tl
2186 }
2187 \cs_new:Npn \CurrentSeries {
2188     \tl_use:N \l_ltx_current_series_tl
2189 }
2190 \cs_new:Npn \CurrentShape {
2191     \tl_use:N \l_ltx_current_shape_tl
2192 }
2193 </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, 1633

A

`\addto` 21
`\Alph` 1624
`\alph` 1620
`\arabic` 66, 67, 1546, 1602
`\AssignTaggingSocketPlug` 576

B

`\babelfont` 1504, 1507, 1510, 1514, 1517, 1520, 1525, 1529, 1533
`\babelprovide` 1496
`\begin` 896, 1022
bool commands:
 `\bool_gset_false:N` 910
 `\bool_if:NTF` 435, 437, 444, 446, 776, 810, 858, 945, 1027
 `\bool_new:N` 487
 `\bool_set_true:N` 760
bourbaki's `\empty_set` 7, 136

C

`\cleaders` 877
clist commands:
 `\clist_gset:Nn` 1641
 `\clist_if_empty:NTF` 1643, 1661
 `\clist_map_inline:Nn` 1662
 `\clist_map_inline:nn` . 155, 211, 215, 267, 399, 1137, 1192, 1196, 1248, 1442, 1751, 1970, 1986, 2001, 2021, 2050, 2062, 2118, 2128
 `\clist_new:N` 1493
 `\clist_pop:NN` 1644, 1663
 `\clist_set_eq:NN` 1642
 `\l_tmpa_clist` 1642, 1643, 1644, 1661, 1662, 1663
color commands:
 `\color_group_begin:` 1731
 `\color_group_end:` 1734
 `\color_select:n` 1732
 `\color_set:nn` 1725
columns 10, 608
cs commands:
 `\cs_generate_variant:Nn` 393, 394, 395, 396, 397, 1426, 1427, 1428, 1499, 1536, 1537, 1538, 1605, 1968
 `\cs_gset_eq:NN` . 48, 83, 84, 750, 1440, 1500, 1544, 1546, 1589, 1602, 1629, 2045, 2090, 2101, 2113, 2156, 2166, 2177
 `\cs_gset_protected:Npn` 1503, 1506, 1509, 1524, 1528, 1532, 1596

\cs_if_exist:NTF	715
\cs_new:Npn	1739, 1756, 2178, 2181, 2184, 2187, 2190
\cs_new_eq:NN	1896, 1897, 1898, 1913, 1914, 1915, 1930, 1931, 1932, 1947, 1948, 1949, 1964, 1965, 1966
\cs_new_protected:Npn	45, 44, 368, 371, 374, 379, 383, 404, 415, 423, 666, 679, 847, 909, 1405, 1412, 1419, 1436, 1447, 1495, 1513, 1516, 1519, 1540, 1619, 2039, 2048, 2093, 2104, 2116, 2158, 2168
\cs_set:Npe	2056, 2123
\cs_set:Npn	520, 1548, 1551, 1554, 1557, 1560, 1563, 1566, 1569, 1572, 1575, 1578, 1581, 1620, 1621, 1622, 1623, 1624, 1625
\cs_set_eq:NN	525
\cs_undefine:N	753
\CurrentEncoding	18, 2178
\CurrentMetaFamily	18, 2178
\CurrentSeries	18, 2178
\CurrentShape	18, 2178
\CurrentSuperFamily	18, 2184

D

\DeclareMathVersion	385
dim commands:	
\dim_zero_new:N	491, 492, 493, 494
\DocumentMetadata	44

E

\em	887
\emph	1629
\encodingdefault	1791
\end	905, 1070
entry_separator	10, 639
exp commands:	
\exp_args:Ne	70, 538, 552, 690, 713, 772, 808, 1652, 1669
\exp_args:Nee	44, 683
\exp_args:NV	2065, 2072, 2131, 2138
\exp_not:N	721, 732, 1025, 1028
\exp_not:n	554, 692, 997, 1004, 1011, 1046, 1053, 1060
expansion	9, 656
expansion_case	9, 592

F

file commands:	
\file_if_exist:nTF	1651, 1666
\finalhyphendemerits	856
\fontencoding	2065, 2072, 2131, 2138
format	9, 577

G

\GetDocumentProperties	647
\gla	83, 84
gloss	9, 650

`\glossaryname` 51, 1021, 1028
`\glx` 8, 10, 756
`\glx*` 8, 10
group commands:
`\group_begin:` . . 169, 183, 229, 243, 549, 667, 689, 718, 758, 805, 852, 911,
926, 984, 1033, 1076, 1151, 1165, 1210, 1224, 1728
`\group_end:` 179, 195, 239, 255, 567, 707, 738, 747, 841, 844, 889, 1016, 1019,
1068, 1072, 1083, 1161, 1177, 1220, 1236, 1736

H

`\hbox` 132, 878
hook commands:
`\hook_gput_code:nnn` . 68, 82, 431, 1458, 1785, 1800, 1804, 1810, 1815, 1822,
1827, 1834, 1839, 1846, 1850, 1854, 1858, 1862, 1866, 1870, 1874, 1878
`\hskip` 879
`\hss` 878
`\hyperlink` 565, 705
`\hypersetup` 552, 690

I

`\IfBlankF` 1735
`\IfBooleanT` 759
`\IfBooleanTF` 1121
`\IfDocumentMetadataT` 867, 880
`\IfDocumentMetadataTF` 682
`\IfEncodingF` 18, 1895
`\IfEncodingT` 18, 1895
`\IfEncodingTF` 18, 1895
`\IfMetaFamilyF` 18, 1912
`\IfMetaFamilyT` 18, 1912
`\IfMetaFamilyTF` 18, 1912
`\IfPackageLoadedF` . 10, 13, 16, 19, 22, 25, 28, 31, 73, 107, 112, 113, 118, 123, 367,
483, 1097, 1098, 1103, 1108, 1112, 1116, 1483, 1487, 1692
`\IfPackageLoadedT` 384, 467, 1626
`\IfPackageLoadedTF` 69, 71, 548, 688, 1502, 1740
`\IfPDFManagementActiveT` 505, 557, 645, 695
`\IfSeriesF` 18, 1946
`\IfSeriesT` 18, 1946
`\IfSeriesTF` 18, 1946
`\IfShapeF` 18, 1963
`\IfShapeT` 18, 1963
`\IfShapeTF` 18, 1963
`\IfSuperFamilyF` 18, 1929
`\IfSuperFamilyT` 18, 1929
`\IfSuperFamilyTF` 18, 1929
int commands:
`\int_compare:nNnTF` 893, 902
`\int_gincr:N` 712

\int_gzero_new:N	495
\int_use:N	714, 716, 723, 734, 894, 897, 903
ipa_bold_italic	I2, II33
ipa_bold_italic_features	I2, II33
ipa_bold_slanted	I2, II33
ipa_bold_slanted_features	I2, II33
ipa_bold_swash	I2, II33
ipa_bold_swash_features	I2, II33
ipa_bold_upright	II, II33
ipa_bold_upright_features	II, II33
ipa_extra_features	I3, II81
ipa_italic	I2, II33
ipa_italic_features	I2, II33
ipa_main_font	II, I247
ipa_mono_bold_italic	I2, II86
ipa_mono_bold_italic_features	I2, II86
ipa_mono_bold_slanted	I2, II86
ipa_mono_bold_slanted_features	I3, II86
ipa_mono_bold_swash	I3, II86
ipa_mono_bold_swash_features	I3, II86
ipa_mono_bold_upright	I2, II86
ipa_mono_bold_upright_features	I2, II86
ipa_mono_extra_features	I3
ipa_mono_font	I2, I247
ipa_mono_italic	I2, II86
ipa_mono_italic_features	I2, II86
ipa_mono_slanted	I2, II86
ipa_mono_slanted_features	I2, II86
ipa_mono_small_caps	I3, II86
ipa_mono_small_caps_features	I3, II86
ipa_mono_swash	I3, II86
ipa_mono_swash_features	I3, II86
ipa_mono_upright	I2, II86
ipa_mono_upright_features	I2, II86
ipa_newcm	II, I258
ipa_newcm_mono	II, I306
ipa_newcm_regular	II, I330
ipa_newcm_regular_mono	II, I378
ipa_newcm_regular_sans	II, I354
ipa_newcm_sans	II, I282
ipa_sans_bold_italic	I2, II86
ipa_sans_bold_italic_features	I2, II86
ipa_sans_bold_slanted	I2, II86
ipa_sans_bold_slanted_features	I2, II86
ipa_sans_bold_swash	I2, II86
ipa_sans_bold_swash_features	I2, II86
ipa_sans_bold_upright	I2, II86

<code>ipa_sans_bold_upright_features</code>	I2, <u>II86</u>
<code>ipa_sans_extra_features</code>	I3
<code>ipa_sans_font</code>	I2, <u>I247</u>
<code>ipa_sans_italic</code>	I2, <u>II86</u>
<code>ipa_sans_italic_features</code>	I2, <u>II86</u>
<code>ipa_sans_slanted</code>	I2, <u>II86</u>
<code>ipa_sans_slanted_features</code>	I2, <u>II86</u>
<code>ipa_sans_small_caps</code>	I2, <u>II86</u>
<code>ipa_sans_small_caps_features</code>	I2, <u>II86</u>
<code>ipa_sans_swash</code>	I2, <u>II86</u>
<code>ipa_sans_swash_features</code>	I2, <u>II86</u>
<code>ipa_sans_upright</code>	I2, <u>II86</u>
<code>ipa_sans_upright_features</code>	I2, <u>II86</u>
<code>ipa_slanted</code>	I2, <u>II33</u>
<code>ipa_slanted_features</code>	I2, <u>II33</u>
<code>ipa_small_caps</code>	I2, <u>II33</u>
<code>ipa_small_caps_features</code>	I2, <u>II33</u>
<code>ipa_swash</code>	I2, <u>II33</u>
<code>ipa_swash_features</code>	I2, <u>II33</u>
<code>ipa_upright</code>	II, <u>II33</u>
<code>ipa_upright_features</code>	II, <u>II33</u>
<code>\ipatext</code>	II, <u>III9</u>
<code>\ipatext*</code>	II, <u>III9</u>

K

<code>\kern</code>	I30, I31, I32, I33, 886
keys commands:	
<code>\l_keys_choice_str</code>	588, 597, 604, I6II
<code>\keys_define:nn</code>	I37, I66, 200, 226, 259, 272, 284, 295, 306, 317, 328, 339, 350, 578, 651, II48, II82, I207, I240, I253, I259, I283, I307, I331, I355, I379, I607, I638, 2026
<code>\keys_set:nn</code>	45, 663, 762, 766, 927, I078, I081, 2041

L

<code>\label</code>	44, 714
<code>\labelenumii</code>	I621
<code>\labelenumiii</code>	I623
<code>\labelenumiv</code>	I625
<code>languages</code>	I5, I637
<code>\LaTeX</code>	5, <u>I26</u>
<code>\leavevmode</code>	865
<code>\leftskip</code>	855
<code>\lingset</code>	I627
<code>\linguistix</code>	5, I9, 47
<code>link_color</code>	9, <u>581</u>
<code>\listofglosses</code>	8, I0, 20, <u>I074</u>
lngx commands:	
<code>\g_lngx_bourbaki_bool</code>	20, <u>I36</u>

\lngx_build_main_ipa_font_features:	1459
\lngx_build_mono_ipa_font_features:	1467
\lngx_build_sans_ipa_font_features:	1463
\lngx_counter:n 15, 21, 66, 67, 1545, 1546, 1549, 1552, 1555, 1558, 1561, 1564, 1567, 1570, 1573, 1576, 1579, 1582, 1589, 1596, 1602	
\l_lngx_current_encoding_tl 22, 1779, 1789, 1805, 1817, 1829, 1841, 1888, 2066, 2073, 2132, 2139, 2179	
\l_lngx_current_meta_family_tl .. 22, 1780, 1789, 1811, 1816, 1823, 1828, 1835, 1840, 1906, 1978, 2070, 2136, 2182	
\l_lngx_current_series_tl . 22, 1782, 1789, 1847, 1851, 1940, 1993, 2078, 2144, 2188	
\l_lngx_current_shape_tl .. 22, 1783, 1789, 1855, 1859, 1863, 1867, 1871, 1875, 1879, 1957, 2013, 2083, 2149, 2191	
\l_lngx_current_super_family_tl 22, 1781, 1789, 1923, 2049, 2117, 2185	
\c_lngx_default_rmdefault_tl	22, 1784
\c_lngx_default_sfdefault_tl	22, 1784
\c_lngx_default_ttdefault_tl	22, 1784
\l_lngx_expansion_bool	487, 760, 776, 810
\lngx_expansion_format:n . 20, 656, 657, 955, 964, 973, 995, 1002, 1009, 1044, 1051, 1058	
\l_lngx_expansion_separator_tl	489, 813, 821, 829
\lngx_gloss_format:n .. 20, 564, 569, 650, 653, 704, 709, 859, 946, 988	
\g_lngx_gloss_link_color_str	39
\lngx_gloss_list:	20, 908, 909, 1082
\lngx_gloss_new:nn	20, 45, 665, 666, 750, 754
\l_lngx_gloss_separator_tl	488, 838
\l_lngx_glossary_separator_tl	490, 862, 948, 990
\l_lngx_gls_language_str	496, 646
\l_lngx_i_hack_dim	494
\l_lngx_i_have_dim	491
\l_lngx_i_need_dim	492
\lngx_if_encoding:n	1882
\lngx_if_encoding:nTF	22, 1881, 1896, 1897, 1898
\lngx_if_encoding_p:n	22, 1881
\lngx_if_meta_family:n	1900
\lngx_if_meta_family:nTF	22, 1899, 1913, 1914, 1915
\lngx_if_meta_family_p:n	22, 1899
\lngx_if_meta_family_rm:TF	22, 1969
\lngx_if_meta_family_rm_p:	22, 1969
\lngx_if_meta_family_sf:TF	22, 1969
\lngx_if_meta_family_sf_p:	22, 1969
\lngx_if_meta_family_tt:TF	22, 1969
\lngx_if_meta_family_tt_p:	22, 1969
\lngx_if_series:n	1934
\lngx_if_series:nTF	22, 1933, 1947, 1948, 1949
\lngx_if_series_bf:TF	23, 1985
\lngx_if_series_bf_p:	23, 1985

\lngx_if_series_md:TF	23, 1985
\lngx_if_series_md_p:	23, 1985
\lngx_if_series_p:n	22, 1933
\lngx_if_shape:n	1951
\lngx_if_shape:nTF	22, 1950, 1964, 1965, 1966
\lngx_if_shape_it:TF	23, 2000
\lngx_if_shape_it_p:	23, 2000
\lngx_if_shape_p:n	22, 1950
\lngx_if_shape_sc:TF	23, 2000
\lngx_if_shape_sc_p:	23, 2000
\lngx_if_shape_sl:TF	23, 2000
\lngx_if_shape_sl_p:	23, 2000
\lngx_if_shape_ssc:TF	23, 2000
\lngx_if_shape_ssc_p:	23, 2000
\lngx_if_shape_sw:TF	23, 2000
\lngx_if_shape_sw_p:	23, 2000
\lngx_if_shape_ulc:TF	23, 2000
\lngx_if_shape_ulc_p:	23, 2000
\lngx_if_shape_up:TF	23, 2000
\lngx_if_shape_up_p:	23, 2000
\lngx_if_super_family:n	1917
\lngx_if_super_family:nTF	22, 1916, 1930, 1931, 1932
\lngx_if_super_family_p:n	22, 1916
lngx_ipa	21, 1429
\lngx_ipa:	21, 1435, 1436, 1440
lngx_ipa_rm_nfss	21, 1404
lngx_ipa_sf_nfss	21, 1404
lngx_ipa_tt_nfss	21, 1404
\lngx_languages:nn	21, 1494, 1495, 1499, 1500, 1650, 1665
\g_lngx_languages_clist	21, 1492, 1641, 1642
\lngx_load_languages:n	21, 1539, 1540, 1544
\lngx_logo_font:	22, 1695, 1696, 1729
\lngx_main_ipa:	21, 1404, 1406
\g_lngx_main_language_tl	21, 1490, 1590, 1645
\lngx_misc_reset:	21, 1618, 1619
\lngx_mono_ipa:	21, 1404, 1420
lngx_multicols	20, 891
\g_lngx_old_style_bool	20, 136, 435, 444
\g_lngx_old_style_one_bool	20, 136, 437, 446
\lngx_other_main_font:nnn	20, 1523, 1524, 1536
\lngx_other_mono_font:nnn	20, 1523, 1532, 1538
\lngx_other_sans_font:nnn	20, 1523, 1528, 1537
lngx_purple_color	22, 1724
\l_lngx_remain_dim	493
\lngx_sans_ipa:	21, 1404, 1413
\l_lngx_separator_tl	53

\lngx_set_keys:n	19, <u>43</u> , 44, 48, 361, 432, 1403, 1541
\lngx_set_main_font:nn	20, <u>366</u> , 368, 393, 453, 1503, 1513
\lngx_set_main_ipa_font:nn	21, <u>1404</u> , 1405, 1426, 1460
\lngx_set_math_bold_font:nn	383, 397, 469
\lngx_set_math_font:nn	20, <u>366</u> , 379, 396, 465
\lngx_set_mono_font:nn	20, <u>366</u> , 374, 395, 461, 1509, 1519
\lngx_set_mono_ipa_font:nn	21, <u>1404</u> , 1419, 1428, 1468
\lngx_set_sans_font:nn	20, <u>366</u> , 371, 394, 457, 1506, 1516
\lngx_set_sans_ipa_font:nn	21, <u>1404</u> , 1412, 1427, 1464
\lngx_soft_normal_font:n	23, <u>2115</u> , <u>2116</u> , 2156, 2159, 2169
\lngx_soft_super_font_family:nn	23, <u>2047</u> , 2048, 2091, 2094, 2105
\lngx_softer_normal_font:	23, <u>2157</u> , 2158, 2166
\lngx_softer_super_font_family:n	23, <u>1437</u> , <u>2092</u> , 2093, 2102
\lngx_softest_normal_font:	23, <u>2167</u> , 2168, 2177
\lngx_softest_super_font_family:n	23, <u>2103</u> , 2104, 2114
\lngx_super_font_family:nn	23, <u>1430</u> , <u>2038</u> , 2039, 2046
\g_lngx_trigger_aux_file_bool	910
lngx internal commands:	
\g__lngx_bold_math_font_features_tl	<u>398</u>
__lngx_build_bold_math_font_features:	<u>398</u>
__lngx_build_main_font_features:	<u>398</u> , 452
__lngx_build_math_bold_features:	423, 468
__lngx_build_math_features:	415, 464
__lngx_build_math_font_features:	<u>398</u>
__lngx_build_mono_font_features:	<u>398</u> , 460
__lngx_build_sans_font_features:	<u>398</u> , 456
__lngx_dotfill:nnn	<u>846</u> , 847, 1038
\l__lngx_entry_separator_tl	639, 850, 888, 985
\l__lngx_family_tmp_tl	1776, 1812, 1818, 1819, 1824, 1830, 1831, 1836, 1842, 1843
__lngx_gloss_description:	39, <u>518</u> , 520, 525, 542
\g__lngx_gloss_link_color_str	554, <u>581</u> , 692
\l__lngx_glossary_columns_int	20, <u>608</u> , 894, 897, 903
\l__lngx_glossary_style_str	499, <u>601</u> , 925
\l__lngx_glosses_page_number_bool	<u>612</u>
\l__lngx_gls_bold_bool	<u>625</u> , 858, 945
\l__lngx_gls_expansion_case_str	498, <u>592</u> , 777, 811, 953, 993, 1042
\l__lngx_gls_section_number_bool	<u>621</u> , 1027
\l__lngx_gls_sectioning_str	<u>617</u> , 1023, 1026
\l__lngx_gls_sorting_order_str	497, <u>585</u> , 914
\g__lngx_gls_use_order_seq	501, 740, 743, 913
\g__lngx_ipa_main_font_features_tl	<u>1133</u> , 1461
\g__lngx_ipa_main_font_tl	<u>1247</u> , 1462
\g__lngx_ipa_main_fonts_prop	<u>1133</u> , 1184
\g__lngx_ipa_mono_font_features_tl	<u>1186</u> , 1469

<code>\g__lngx_ipa_mono_font_tl</code>	1247 , 1470
<code>\g__lngx_ipa_mono_fonts_prop</code>	1186
<code>\g__lngx_ipa_sans_font_features_tl</code>	1186 , 1465
<code>\g__lngx_ipa_sans_font_tl</code>	1247 , 1466
<code>\g__lngx_ipa_sans_fonts_prop</code>	1186
<code>\g__lngx_math_bold_features_tl</code>	277 , 425 , 470
<code>\g__lngx_math_bold_font_tl</code>	288 , 471
<code>\g__lngx_math_bold_fonts_prop</code>	277 , 424
<code>\g__lngx_math_features_tl</code>	277 , 417 , 465
<code>\g__lngx_math_font_features_tl</code>	398
<code>\g__lngx_math_font_tl</code>	286 , 466
<code>\g__lngx_math_fonts_prop</code>	277 , 416
<code>\l__lngx_nfss_tmp_tl</code>	1777 , 2030 , 2033 , 2040 , 2042
<code>\l__lngx_normalfont_tmp_tl</code>	1774
<code>\g__lngx_page_ref_int</code>	44 , 495 , 712 , 714 , 716 , 723 , 734
<code>\l__lngx_selectfont_tmp_tl</code>	1775 , 1801 , 1806 , 1807
<code>\l__lngx_separator_str</code>	500 , 503 , 633 , 761 , 1077
<code>\l__lngx_separator_tl</code>	629
<code>\g__lngx_text_main_font_features_tl</code>	151 , 398 , 454
<code>\g__lngx_text_main_font_tl</code>	266 , 455
<code>\g__lngx_text_main_fonts_prop</code>	151 , 202
<code>\g__lngx_text_mono_font_features_tl</code>	204 , 398 , 462
<code>\g__lngx_text_mono_font_tl</code>	266 , 463
<code>\g__lngx_text_mono_fonts_prop</code>	204
<code>\g__lngx_text_sans_font_features_tl</code>	204 , 398 , 458
<code>\g__lngx_text_sans_font_tl</code>	266 , 459
<code>\g__lngx_text_sans_fonts_prop</code>	204
<code>__lngx_tmp_text:</code>	509 , 511
<code>\lngxipa</code>	11 , 17 , 21 , 1123 , 1128 , 1435
<code>\lngxlogo</code>	16 , 1726 , 1742 , 1747 , 1758
<code>\lngxpkg</code>	1739
<code>\loadlanguages</code>	15 , 1539
<code>\localecounter</code>	1597
<code>logical</code>	15 , 1593

M

<code>\MakeLinkTarget</code>	942 , 986 , 1039
<code>\MakeLinkTarget*</code>	49 , 52
<code>math</code>	6 , 277
<code>math_{bold}</code>	6 , 277
<code>math_{bold}_{features}</code>	6 , 277
<code>math_{features}</code>	6 , 277
mode commands:	
<code>\mode_leave_vertical:</code>	536
msg commands:	
<code>\msg_info:nnn</code>	87 , 92
<code>\msg_new:nnn</code>	59 , 1632

\msg_warning:nnn	1655, 1672
\multicol	38

N

native_numbering	15, 1604
newcm	6, 294
newcm_mono	6, 316
newcm_regular	6, 327
newcm_regular_mono	6, 349
newcm_regular_sans	6, 338
newcm_sans	6, 305
\NewCommandCopy	127
\NewDocumentCommand	662, 752, 757, 1075, 1120, 1727
\NewDocumentEnvironment	892
\newfontfamily	1696
\newgloss	8, 20, 665
\NewTaggingSocket	533
\NewTaggingSocketPlug	535
no_bold	10, 625
\noindent	900
\normalfont	2061, 2127

O

off	15, 1600
\ogLaTeX	5, 126
old_style_numbers	6, 136
old_style_one	6, 136

P

page_numbers	10, 612
\par	642
\parfillskip	854
\parindent	857
pdfannot commands:	
\pdfannot_dict_put:nnn	510
\pdfstringdef	509
prg commands:	
\prg_do_nothing:	525, 853, 854, 855, 856, 857, 879, 886
\prg_new_conditional:Nnn	1882, 1900, 1917, 1934, 1951, 1968, 1975, 1990, 2010
\prg_return_false:	1891, 1909, 1926, 1943, 1960, 1981, 1996, 2016
\prg_return_true:	1889, 1907, 1924, 1941, 1958, 1979, 1994, 2014
prop commands:	
\prop_gclear_new:N	152, 205, 208, 278, 281, 1134, 1187, 1189
\prop_gput:Nnn	176, 190, 236, 250, 1158, 1172, 1217, 1231
\prop_map_inline:Nn	407, 416, 424, 1450
\ProvideDocumentCommand	1021
\providelanguage	15, 1494
\ProvidesExplPackage	2, 36, 51, 99, 476, 1087, 1474, 1685, 1766

Q

\quad 866, 885

R

\raisebox 131, 133
\ref 44
\relax 130, 131, 132, 133
\RenewDocumentCommand 129
\renewgloss 8, 45, 751
\RequirePackage II, 14, 17, 20, 23, 26, 29, 32, 108, 114, 119, 124, 484, 1099, 1104,
1109, 1113, 1117, 1484, 1488, 1652, 1669, 1693
\rightskip 853
\rmdefault 1786
\roman 1622

S

\section 41
section_number 10, 621
sectioning 10, 617
\selectfont 2074, 2140
separator 10, 629
seq commands:
 \seq_clear:N 768, 912, 1034
 \seq_gclear_new:N 501, 673
 \seq_gput_right:Nn 729, 743
 \seq_if_empty:NTF 803, 982
 \seq_if_in:NnTF 726, 740
 \seq_map_inline:Nn 804, 983, 1032, 1035
 \seq_pop_left:NN 770, 933
 \seq_put_right:Nn 1036
 \seq_remove_duplicates:N 44
 \seq_set_eq:NN 913
 \seq_set_from_clist:Nn 769
 \seq_sort:Nn 916
 \seq_use:Nn 1066
 \l_tmpa_seq .. 768, 769, 770, 803, 804, 912, 913, 916, 933, 982, 983, 1032
 \l_tmpb_seq 1034, 1036, 1066
\setfontfamily 1406, 1413, 1420
\setmainfont 369
\setmathfont 380, 387
\setmonofont 375
\setsansfont 372
\setupglossing 9, 661
\sfddefault 1787
\smallskip 851
socket commands:
 \socket_assign_plug:nn 528, 559, 699, 1605, 1610
 \socket_if_exist:nTF 506, 558, 696
 \socket_new:nn 517, 1585
 \socket_new_plug:nnn 507, 519, 523, 1587, 1594, 1601

`\socket_use:n` 531, 1613
`\softernormalfont` 19, 2157
`\softersuperfontfamily` 19, 2092
`\softestnormalfont` 19, 2167
`\softestsuperfontfamily` 19, 2103
`\softnormalfont` 19, 2115
`\softsuperfontfamily` 19, 2047
`sort` 9, 585
sort commands:
`\sort_return_same:` 920
`\sort_return_swapped:` 918
str commands:
`\c_colon_str` 928, 1079
`\str_case:nn` 777, 811, 914, 953, 993, 1042, 2063, 2129
`\str_clear:N` 170, 184, 230, 244, 550, 668, 806, 932, 1152, 1166, 1211, 1225
`\str_clear_new:N` 496, 497, 498, 499, 500
`\str_compare:nNnTF` 917
`\str_if_eq:nnTF` 925, 1023
`\str_lowercase:n` 43, 669, 772, 808
`\str_replace_all:Nnn` 175, 189, 235, 249, 1157, 1171, 1216, 1230
`\str_set:Nn` 171, 185, 231, 245, 503, 551, 646, 669, 761, 771, 807, 934, 1077, 1153, 1167, 1212, 1226
`\str_set_eq:NN` 587, 596, 603
`\str_use:N` . 178, 193, 253, 633, 671, 674, 677, 680, 685, 727, 730, 741, 744, 781, 788, 795, 801, 816, 824, 832, 839, 943, 958, 967, 976, 1026, 1160, 1175, 1234, 1611
`\l_tmpa_str` . 170, 171, 175, 178, 184, 185, 189, 193, 230, 231, 235, 244, 245, 249, 253, 550, 551, 668, 669, 671, 674, 677, 680, 685, 727, 730, 741, 744, 771, 781, 788, 795, 801, 806, 807, 816, 824, 832, 839, 932, 934, 943, 958, 967, 976, 1152, 1153, 1157, 1160, 1166, 1167, 1171, 1175, 1211, 1212, 1216, 1225, 1226, 1230, 1234
`strict` 15, 1586
`style` 9, 601
`\superfontfamily` 18, 2038
sys commands:
`\sys_if_engine luatex:TF` 72, 111, 1096

T

tag commands:
`\tag_mc_begin:n` 545, 573, 873, 883, 939
`\tag_mc_end:` 537, 571, 868, 881, 935, 951
`\tag_struct_begin:n` 539, 869, 936
`\tag_struct_end:` 572, 882, 952
T_EX and L^AT_EX 2_ε commands:
`\@currentmetafamily` 1794
`\f@encoding` 1801, 1812, 1824, 1836
`\glw@gla` 84
`\texorpdfstring` 1741, 1757
`text_{}bold_{}italic` 12, 151

<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>	779, 814, 956, 996, 1045
<code>\text_titlecase_all:n</code>	I72, I86, 232, 246, 786, 822, 965, 1003, 1052, II54, II68, I2I3, I227
<code>\text_titlecase_first:n</code>	793, 830, 974, 1010, 1059
<code>text_extra_features</code>	I3, I99
<code>text_italic</code>	I2, I5I
<code>text_italic_features</code>	I2, I5I
<code>text_main_font</code>	II, 266
<code>text_mono_bold_italic</code>	I2, 204
<code>text_mono_bold_italic_features</code>	I2, 204
<code>text_mono_bold_slanted</code>	I2, 204
<code>text_mono_bold_slanted_features</code>	I3, 204
<code>text_mono_bold_swash</code>	I3, 204
<code>text_mono_bold_swash_features</code>	I3, 204
<code>text_mono_bold_upright</code>	I2, 204
<code>text_mono_bold_upright_features</code>	I2, 204
<code>text_mono_extra_features</code>	I3
<code>text_mono_font</code>	I2, 266
<code>text_mono_italic</code>	I2, 204
<code>text_mono_italic_features</code>	I2, 204
<code>text_mono_slanted</code>	I2, 204
<code>text_mono_slanted_features</code>	I2, 204
<code>text_mono_small_caps</code>	I3, 204
<code>text_mono_small_caps_features</code>	I3, 204
<code>text_mono_swash</code>	I3, 204
<code>text_mono_swash_features</code>	I3, 204
<code>text_mono_upright</code>	I2, 204
<code>text_mono_upright_features</code>	I2, 204
<code>text_sans_bold_italic</code>	I2, 204
<code>text_sans_bold_italic_features</code>	I2, 204
<code>text_sans_bold_slanted</code>	I2, 204
<code>text_sans_bold_slanted_features</code>	I2, 204
<code>text_sans_bold_swash</code>	I2, 204
<code>text_sans_bold_swash_features</code>	I2, 204
<code>text_sans_bold_upright</code>	I2, 204
<code>text_sans_bold_upright_features</code>	I2, 204
<code>text_sans_extra_features</code>	I3

text_sans_font	I2, 266
text_sans_italic	I2, 204
text_sans_italic_features	I2, 204
text_sans_slanted	I2, 204
text_sans_slanted_features	I2, 204
text_sans_small_caps	I2, 204
text_sans_small_caps_features	I2, 204
text_sans_swash	I2, 204
text_sans_swash_features	I2, 204
text_sans_upright	I2, 204
text_sans_upright_features	I2, 204
text_slanted	I2, 151
text_slanted_features	I2, 151
text_small_caps	I2, 151
text_small_caps_features	I2, 151
text_swash	I2, 151
text_swash_features	I2, 151
text_upright	II, 151
text_upright_features	II, 151
\textbf	858, 945, 987
\textit	1629
\textsc	9, 131, 655
\thechapter	1548
\theenumii	1620, 1621
\theenumiii	1622, 1623
\theenumiv	1624, 1625
\theequation	1581
\thefigure	1569
\thefootnote	1575
\thempfootnote	1578
\thepage	1566
\theparagraph	1560
\thesection	1551
\thesubparagraph	1563
\thesubsection	1554
\thesubsubsection	1557
\thetable	1572
tl commands:	
\c_space_tl	763, 928, 1079
\tl_clear:N	719, 767, 931, 1659, 1660, 1676, 1677, 1807, 1819, 1831, 1843, 2042
\tl_clear_new:N	488, 489, 490
\tl_const:Nn	1786, 1787, 1788
\tl_gclear_new:N	153, 206, 209, 279, 282, 670, 1135, 1188, 1190, 2029
\tl_gput_right:Nn	408, 417, 425, 1451
\tl_gset:Nn	676, 2032
\tl_if_empty:NTF	2055

`\tl_if_eq:NnTF` 1888, 1906, 1923, 1940, 1957, 1978, 1993, 2013
`\tl_new:N` 1491, 1774, 1775, 1776, 1777, 1779, 1780, 1781, 1782, 1783
`\tl_set:Nn` 631, 720, 1646, 1664, 1789, 1790, 1793, 1796, 1797,
1801, 1811, 1812, 1816, 1823, 1824, 1828, 1835, 1836, 1840, 1847, 1851, 1855, 1859,
1863, 1867, 1871, 1875, 1879, 2040, 2049, 2117
`\tl_set_eq:NN` 1645, 1805, 1817, 1829, 1841
`\tl_use:N` 773, 780, 787, 794, 813, 815, 821, 823, 829, 831, 838, 862, 947,
948, 957, 966, 985, 990, 1590, 2057, 2124, 2179, 2182, 2185, 2188, 2191
`\tl_use:n` 975
`\l_tmpa_tl` 719, 720, 728, 767, 770, 773, 931, 933, 934, 947, 1644, 1645, 1650,
1651, 1653, 1657, 1659, 1663, 1665, 1667, 1670, 1674, 1676
`\l_tmpb_tl` 1646, 1650, 1660, 1664, 1665, 1677
`\ttdefault` 1788

U

`\umgla` 5, 83
use commands:
`\use:N` 722, 733, 801, 839, 1025, 2069, 2077, 2082, 2135, 2143, 2148
`\use:n` 1024
`\use_ii:nnnnn` 721, 732
`\UseTaggingSocket` 683