The codedescribe and codelisting Packages
Version 1.24

Alceu Frigeri*

February 2026

Abstract

This package is designed to be as class independent as possible, depending only on *expl*, *scontents*, *listings*, *xpeekahead*, *pifont* and *xcolor*. A minimal set of macros/commands/environments is defined: most/all defined commands have an "object type" as a *keyval* parameter, allowing for an easy expansion mechanism (instead of the usual "one set of macros/environments" for each object type).

No assumption is made about page layout (besides "having a margin paragraph"), or underlying macros, so it should be possible to use this with any/most document classes.

Contents

*https://github.com/alceu-frigeri/codedescribe

1 Introduction

This package aims at documenting both document level and package/class level commands. It's fully implemented using *expl*, requiring just an up to date kernel. *scontents* and *listings* packages (see [4] and [5]) are used to typeset code snippets. The package *pifont* (see [8]) is needed just to typeset those (open)stars, in case one wants to mark a command as (restricted) expandable. *xcolor* (see [6]) is needed to switch colors, and *xpeekahead* (see [3]) for spacing fine tuning. The package *infograb* (see [2]) is loaded for package's documentation only.

No other package/class is needed, and it should be possible to use these packages with most classes[1], which allows to demonstrate document commands with any desired layout.

Generating an index is supported (since version 1.20, see 3.2 and 3.3.3) but no index package is pre-loaded, leaving it to the end user.

codelisting defines a few macros to display and demonstrate LaTeX code (using *listings* and *scontents*), *codedescribe* defines a series of macros to display/enumerate macros and environments (somewhat resembling the *doc3* style), including some shortcuts (active characters, since version 1.23), and *codelstlang* defines a series of *listings* TeX dialects.

These packages are fairly stable, and given the ⟨obj-type⟩ mechanism (see 3.3) they can be easily extended without changing their interface.

1.1 Single versus Multi-column Classes

This package "can" be used with multi-column classes, given that the `\linewidth` and `\columnsep` are defined appropriately. `\linewidth` shall defaults to text/column real width, whilst `\columnsep`, if needed (2 or more columns), shall be greater than `\marginparwidth` plus `\marginparsep`.

[1] If, by chance, a class with compatibility issues is found, just open an issue at `https://github.com/alceu-frigeri/codedescribe/issues` to see what can be done

2 codelisting Package

It loads: *listings*, *scontents* and *xpeekahead*, defines an environment: *codestore* and a few commands for listing and demonstration of LaTeX code.

2.1 Package Options

The following options can also be set via *codedescribe* package options, see 3.1.

`strict` Package Warnings will be reported as Package Errors.

`suppress deprecated` Suppress the messages associated with deprecated commands/keys.

`colors` Possible values: `black`, `default`, `brighter` and `darker`. This will adjust the initial color configuration for the many listings' elements (used by `\tscode` and `\tsdemo`). `black` will defaults all colors to black. `default`, `brighter` and `darker` are roughly the same color scheme. The `default` scheme is the one used in this document. With `brighter` the colors are brighter than the default, and with `darker` the colors will be darker, but not black.

`load xtra dialects` (defaults to false) If set, it will load the auxiliary package *codelstlang* (see 4), which just defines a series of *listings* TeX dialects.

`TeX dialects` This will set which *listings* TeX dialects will be used when defining the listing style `codestyle`. It defaults to `doctools`, which is derived from the [LaTeX]TeX dialect (this contains the same set of commands used by the package *doctools*). One can use any valid (TeX derived) *listings* dialect, including user defined ones, see [5] for details.

Besides those, one can use (if the `load xtra dialects` is set): *l3kernelsign*, *l3expsign*, *l3amssign*, *l3pgfsign*, *l3bibtexsign*, *l3kernel*, *l3exp*, *l3ams*, *l3pgf*, *l3bibtex*, *kernel*, *xpacks*, *ams*, *pgf*, *pgfplots*, *bibtex*, *babel* and *hyperref*. See 4 for details on those dialects.

> ***Note:*** `TeX dialects` is a comma separated list of the dialect's name, without the base language (internally it will be converted to *[dialect]TeX*).

For example:

```
   %% could be \usepackage[...]{codelisting}
\usepackage[load xtra dialects,
  TeX dialects={doctools,l3kernel,l3ams}]{codedescribe}
   %%
   %% assuming the user has defined a dialect, named: [my-own-set]TeX
   %%
\usepackage[TeX dialects={doctools,my-own-set}]{codedescribe}
   %%
```

2.2 In Memory Code Storage

Thanks to *scontents*, it's possible to store LaTeX code snippets in an *expl* sequence variable.

codestore `\begin{codestore}` [⟨stcontents-keys⟩]
`\end{codestore}`

This environment is an alias to *scontents* environment (from *scontents*, see [4]), all *scontents* keys are valid, with two additional ones: *st* and *store-at* which are aliases to the *store-env* key. If an "isolated" ⟨*st-name*⟩ is given (unknown *key*), it is assumed that the environment body shall be stored in it (for use with `\tscode`, `\tsmergedcode`, `\tsdemo`, `\tsresult` and `\tsexec`).

> ***Note:*** From *scontents*, ⟨st-name⟩ is ⟨index⟩ed (The code is stored in a sequence variable). It is possible to store as many code snippets as needed under the same name. The first one will be ⟨index⟩→ 1, the second 2, and so on.
>
> ***Warning:*** If explicitly using one of the *store-env*, *st* or *store-at* keys, the storage name can be anything. BUT, due to changes (August 2025) in the latex kernel keys processing, if an implicity key is used, then colons (:), besides a comma and equal signs, aren't allowed.

LaTeXCode:

```
%The code will be stored as 'store:A'
\begin{codestore}[store-env = store:A]
...
\end{codestore}

%Same
\begin{codestore}[st = store:A]
...
\end{codestore}

%The code will be stored as 'storeA'
\begin{codestore}[storeA]
...
\end{codestore}

%This might raises an error.
%It will be stored as 'store' (not as 'store:A')
\begin{codestore}[store:A]
...
\end{codestore}
```

2.3 Code Display/Demo

\tscode*
\tsdemo*
\tsresult*

updated: 2024/01/06
updated: 2025/04/29

\tscode* [⟨code-keys⟩] {⟨st-name⟩} [⟨index⟩]
\tsdemo* [⟨code-keys⟩] {⟨st-name⟩} [⟨index⟩]
\tsresult* [⟨code-keys⟩] {⟨st-name⟩} [⟨index⟩]

\tscode* just typesets ⟨st-name⟩ (created with *codestore*) verbatim with syntax highlight (from *listings* package [5]). The non-star version centers it and use just half of the base line. The star version uses the full text width.
\tsdemo* first typesets ⟨st-name⟩, as above, then *executes* it. The non-start version place them side-by-side, whilst the star version places one following the other.
(new 2024/01/06) \tsresult* only *executes* it. The non-start version centers it and use just half of the base line, whilst the star version uses the full text width.

> ***Note:*** (from *stcontents* package) ⟨index⟩ can be from 1 up to the number of stored codes under the same ⟨st-name⟩. Defaults to 1.
>
> ***Note:*** All are executed in a local group which is discarded at the end. This is to avoid unwanted side effects, but might disrupt code execution that, for instance, depends on local variables being set. That for, see \tsexec below.

For Example:

LaTeXCode:

```
\begin{codestore}[stmeta]
    Some \LaTeX{} coding, for example: \ldots.
\end{codestore}

This will just typesets \tsobj[key,no index]{stmeta}:
\tscode*[codeprefix={Sample Code:}] {stmeta}

and this will demonstrate it, side by side with source code:
\tsdemo[numbers=left,firstnumber=5,ruleht=0.5,
    codeprefix={inner sample code},
    resultprefix={inner sample result}] {stmeta}
```

LaTeXResult:

This will just typesets *stmeta*:

Sample Code:

```
    Some \LaTeX{} coding, for example: \ldots.
```

and this will demonstrate it, side by side with source code:

inner sample code

```
5   Some \LaTeX{} coding, for example: \ldots.
```

inner sample result

Some LaTeX coding, for example: ....

\tsmergedcode* `\tsmergedcode* [⟨code-keys⟩] {⟨st-name-index list⟩}`

new: 2025/04/29

This will typeset (as `\tscode`) the merged contents from ⟨st-name-index list⟩. The list syntax comes from *scontents* (command `\mergesc`), where it is possible to refer to a single index `{⟨st-name A⟩} [⟨index⟩]`, a index range `{⟨st-name B⟩} [⟨indexA-indexB⟩]`, or all indexes from a ⟨st-name⟩, `{⟨st-name C⟩} [⟨1-end⟩]`. The special index ⟨1-end⟩ refers to all indexes stored under a given ⟨st-name⟩.

> ***Note:*** The brackets aren't optional. For instance `\tsmergedcode* [⟨code-keys⟩] { {⟨st-name A⟩} [⟨index⟩] , {⟨st-name B⟩} [⟨indexA-indexB⟩] , {⟨st-name C⟩} [⟨1-end⟩] }`

\tsexec `\tsexec {⟨st-name⟩} [⟨index⟩]`

new: 2025/04/29

Unlike the previous commands which are all executed in a local group (discarded at the end) this will execute the code stored at ⟨st-name⟩ [⟨index⟩] in the current LaTeX group.

2.3.1 Colors Customization

\setlistcolorscheme `\setlistcolorscheme {⟨color-key-list⟩}`

new: 2025/12/14

This allows to customize the default colors used by `\tscode` and `\tsdemo` when typesetting (assuming the default listings's style is being used). Note that the given colors will be mixed with black. The key *brightness* set's the mixing proportion. The changes become effective at the point of use.

⟨color-key-list⟩ can be any combination of:

bckgnd	(default: black) Sets the background base color. Note this is mixed with white, not black as the others.
string	(default: teal) Sets the string base color
comment	(default: green) Sets the comment base color
texcs	(default: blue) Sets the texcs (TeX commands) base color
keywd	(default: cyan) Sets the keywd (keywords) base color
emph	(default: red) Sets the emph (emphasis) base color
rule	(default: gray) Sets the rule (unused by now) base color. Note this is mixed with white, not black as the others.
number	(default: gray) Sets the (small line) numbers base color. Note this is mixed with white, not black as the others.
brightness	(default: 1) Sets the mixing proportion between each base color and black.
scheme	(Defaults to *scheme=default*) Selects a pre-set color scheme, see below, the default scheme sets all of the above to their default value.

\newlistcolorscheme `\newlistcolorscheme {⟨new-scheme⟩} {⟨color-key-list⟩}`

new: 2025/12/14

This creates/defines a ⟨new-scheme⟩ (⟨color-key-list⟩ as above) which can be later used as `\setlistcolorscheme{scheme=new-scheme}`

2.3.2 Code Keys

\setcodekeys `\setcodekeys {⟨code-keys⟩}`

One has the option to set ⟨code-keys⟩ per **\tscode**, **\tsmergedcode**, **\tsdemo** and **\tsresult** call (see 2.3), or *globally*, better said, *in the called context group* .

> ***N.B.:*** All **\tscode** and **\tsdemo** commands create a local group in which the ⟨code-keys⟩ are defined, and discarded once said local group is closed. **\setcodekeys** defines those keys in the *current* context (which might be global or not, the declarations are always local).

\newcodekey `\newcodekey {⟨new-key⟩}{⟨code-keys⟩}`

new: 2025/05/01
updated: 2025/12/23

This will define a new key ⟨new-key⟩, which can be used with **\tscode**, **\tsmergedcode**, **\tsdemo** and **\tsresult**. ⟨code-keys⟩ can be any of the following ones, including other ⟨new-key⟩s. Be careful not to create a definition loop.

> ***Note:*** The old **\setnewcodekey** is (now) an alias to this, and will raise a warning if called (deprecation).

lststyle `lststyle = {⟨listings style⟩}`

new: 2025/11/12

This sets the base style to be used. It defaults to *codestyle*, and the user can use this (*codestyle*) as the base style for his own one (and avoid having to define every single aspect of it). For example:

```
\lstdefinestyle{my-own}{   % see the listings manual for a complete list of keywords
  style=codestyle,
  texcsstyle        = *    {\bfseries\color{red}}
}

\tscode*[lststyle=my-own]{demo-X}
```

settexcs `settexcs = [⟨num⟩]{⟨csv-list⟩}`
texcs `texcs = [⟨num⟩]{⟨csv-list⟩}`
texcsstyle `texcsstyle = [⟨num⟩]{⟨style⟩}`

updated: 2025/05/01
updated: 2025/12/29

These define sets of LaTeX commands (csnames, sans the preceding slash bar), the *settexcs* initialize/redefine those sets (an empty list, clears the set), while *texcs* extend those sets. The *texcsstyle* redefines the display style. ⟨num⟩ can be any number, though, currently, only 1 to 8 have a pre-defined style associated with them.

> ***Note:*** The old keys *settexcs2*, *settexcs3*, *settexcs4*, *texcs2*, *texcs3*, *texcs4*, *texcs2style*, *texcs3style* and *texcs4style* still work, but will raise a warning (deprecation), if used.

setkeywd `setkeywd = [⟨num⟩]{⟨csv-list⟩}`
keywd `keywd = [⟨num⟩]{⟨csv-list⟩}`
keywdstyle `keywdstyle = [⟨num⟩]{⟨style⟩}`

updated: 2025/05/01
updated: 2025/12/29

Same for other *keywords* sets.

> ***Note:*** The old keys *setkeywd2*, *setkeywd3*, *setkeywd4*, *keywd2*, *keywd3*, *keywd4*, *keywd2style*, *keywd3style* and *keywd4style* still work, but will raise a warning (deprecation), if used.

setemph `setemph = [⟨num⟩]{⟨csv-list⟩}`
emph `emph = [⟨num⟩]{⟨csv-list⟩}`
emphstyle `emphstyle = [⟨num⟩]{⟨style⟩}`

updated: 2025/05/01
updated: 2025/12/29

for some extra emphasis sets.

> ***Note:*** The old keys *setemph2*, *setemph3*, *setemph4*, *emph2*, *emph3*, *emph4*, *emph2style*, *emph3style* and *emph4style* still work, but will raise a warning (deprecation), if used.

letter `letter = {⟨csv-list⟩}`
other `other = {⟨csv-list⟩}`

new: 2025/05/13

These allow to redefine what a letter or other are (they correspond to the *alsoletter* and *alsoother* keys from *listings*). The default value for the *letter* includes (sans the comma) `@ : _` , whilst *other*'s default value is an empty list.

> ***Note:*** You might want to consider setting *letter* to just *letter={@,_}* so you don't have to list all *expl* variants of a command, but just the base name of them.

numbers `numbers = {⟨none⟩} | {⟨left⟩}`
numberstyle `numberstyle = {⟨style⟩}`
firstnumber `firstnumber = {⟨num⟩}`

updated: 2025/12/16

numbers possible values are *none* (package's default) and *left* (key default value, to add tinny numbers to the left of the listing). With *numberstyle* one can redefine the numbering style. *firstnumber* sets the numbering start, it can be any number, *last* or *auto*. It's initialized with *last* (see [5] for details).

stringstyle `stringstyle = {⟨style⟩}`
commentstyle `commentstyle = {⟨style⟩}`

to redefine *strings* and *comments* formatting style.

bckgndcolor `bckgndcolor = {⟨color⟩}`

to change the listing background's color.

codeprefix `codeprefix = {⟨string⟩}`
resultprefix `resultprefix = {⟨string⟩}`

those set the *codeprefix* (initial value: LaTeX Code:) and *resultprefix* (initial value: LaTeX Result:)

parindent `parindent = {⟨indent⟩}`

Sets the indentation to be used when 'demonstrating' LaTeXcode (`\tsdemo`). It's initialized with whatever value `\parindent` was when the package was first loaded.

ruleht `ruleht = {⟨scale⟩}`

When typesetting the 'code demo' (`\tsdemo`) a set of rules are drawn. It's initialized with 1 (scaling factor, equals to `\arrayrulewidth`, usually 0.4pt).

basicstyle `basicstyle = {⟨style⟩}`

new: 2023/11/18

Sets the base font style used when typesetting the 'code demo', default being `\footnotesize \ttfamily`

3 codedescribe Package

This package aims at minimizing the number of commands, being the object kind (if a command, environment, key etc.) a parameter, allowing for a simple extension mechanism: other object types can be easily introduced without having to change, or add commands.

3.1 Package Options

nolisting Will suppress the *codelisting* package load. In case it isn't needed or another listing package will be used.

label set (new: 2025/11/22) This allows to pre-select a label set, see 3.4. Currently, the possible values are `english`, `german` and `french`, the ones present in the auxiliary package *codedescsets*.

base skip Changes the base skip, all skips (used by the environments at 3.5) are scaled up from this. It defaults to the font size at load time.

strict Package Warnings will be reported as Package Errors. This will be passed over to *codelisting* as well.

suppress deprecated (new: 2025/12/30) Suppress the messages associated with deprecated commands/keys. This will be passed over to *codelisting* as well.

silence (new: 2025/11/22, defaults to 18.89999pt) This will suppress some annoying bad boxes warnings. Given the way environments at 3.5 are defined, with *expl* coffins, TEX sometimes thinks they are too wide, when they are not. This just sets `\hfuzz` to the given value.

describe keys (defaults to *grouped*) This sets the way the keys *new*, *update* and *note* are listed in a *codedescribe* environment, see 3.5. Possible values are *grouped* or *as is*. By default keys are grouped together, with *as is* keys will respect the used sequence.

index (new: 2025/12/15) This will enable the many *index* keys and set the default of some object groups to *index*. This **won't load** any index package, but just change some objects' default behaviour (see 3.2 and 3.3). If not set, all *index* keys will be silently ignored.

colors Possible values: `black`, `default`, `brighter` and `darker`. This will adjust the initial color configuration for the many format groups/objects (see 3.3.1). `black` will defaults all `\tsobj` colors to black. `default`, `brighter` and `darker` are roughly the same color scheme. The `default` scheme is the one used in this document. With `brighter` the colors are brighter than the default, and with `darker` the colors will be darker, but not black.

codelisting The argument of this (it's value) will be passed over to *codelisting* as package options (if loaded). For example: *code listing* = *{colors=brighter, load xtra dialects}*. See 2.1.

infograb This will enable the document level, LATEX2e, aliases from the package *pkginfograb* [2].

> ***Note:*** In case of an unknown `label set`, an error will be risen, and all known sets will be listed in the log file and terminal.

3.2 Indexing

It's up to the user to choose a companion package to format and display index entries, though a very simple setup, using *xindex*'s defaults, could just be:

```
% in the document's preamble
\usepackage{xindex}
\makeindex
...

% at the document's end
\printindex
```

Similarly, given the many index package variants (specially how index entries shall be created), the user is expected to supply an index generating command (key *index fmt*, see 3.3.1), which shall absorb 4 parameters. This *user supplied* command will be used by the command `\tsobj` and environments *codedescribe* and *describelist* to create index entries.

This package offers four such auxiliary commands, for some common cases.

> ***Note:*** The package option `index` won't load any index package, but just set the defaults of some format groups (see 3.3.2) to generate index entries.

\indexfmtraw
\indexfmtrawat
\indexfmtcsraw
\indexfmtcsrawat

```
\indexfmtraw {⟨name⟩}{⟨prefix⟩}{⟨group⟩}{⟨item⟩}
\indexfmtrawat {⟨name⟩}{⟨prefix⟩}{⟨group⟩}{⟨item⟩}
\indexfmtcsraw {⟨name⟩}{⟨prefix⟩}{⟨group⟩}{⟨item⟩}
\indexfmtcsrawat {⟨name⟩}{⟨prefix⟩}{⟨group⟩}{⟨item⟩}
```

new: 2025/12/19

⟨item⟩ is the item (from `\tsobj` or *codedescribe* or *describelist*) to be indexed. ⟨group⟩ corresponds to the key *index group*. ⟨prefix⟩ corresponds to the key *index prefix*. Finally, ⟨name⟩ (which corresponds to the key *index name*) if not empty, will be enclosed in brackets, for instance, `\tsobj[indexname={some},code]{\cmd}` will result in (with everything at its default value) `\indexfmtcsrawat{[some]}{}{}{\cmd}`. This allows to avoid testing the first parameter value, and use it 'as is': If the *index name* isn't set, ⟨name⟩ will be empty.

`\indexfmtraw{name}{prefix}{group}{item}` will ignore the 2nd and 3rd parameters, being equivalent to `\index{item}` (or `\index[name]{item}`).

`\indexfmtcsraw{name}{prefix}{group}{\cmd}` will also ignore the 2nd and 3rd parameters, being equivalent to `\index{\string\cmd}` (or `\index[name]{\string\cmd}`). The primitive `\string` will precede the ⟨item⟩ (if it is a command).

The other two commands, `\indexfmtrawat` and `\indexfmtcsrawat`, will create index entries as ⟨**prefix**⟩⟨**item**⟩**@**⟨**group**⟩**!**⟨**item**⟩. The backslash, if any, is removed from the first ⟨item⟩ (preceding the @) in `\indexfmtcsrawat`.

> ***Note:*** The actual characters specifiers can be changed with the command `\indexcodesetup`.
>
> ***Note:*** Of course, ⟨name⟩ is useful only in case of packages like *imakeidx* or *splitindex*, which allows multiple indexes. Don't use/set *index name*, if you aren't using a multi-index aware package.

\indexgenkey

```
\indexgenkey {⟨tl-var⟩}{⟨terms-list⟩}
```

new: 2025/12/23

This auxiliary command will construct part of an index key from ⟨terms-list⟩. All terms from ⟨terms-list⟩ will be concatenate using a *level* separator (normally a ***!***) and the result assigned to ⟨tl-var⟩. The actual *level* character being used can be changed with `\indexcodesetup`. For example, `\indexgenkey{\partkey}{a,b,c}` will result in `\partkey` having *a!b!c*, or `\indexgenkey{\partkey}{a,b,c,{}}` will result in `\partkey` having *a!b!c!*.

> ***Note:*** ⟨terms-list⟩ will be fully expanded before being processed.

\indexcodesetup

```
\indexcodesetup {⟨index-specs⟩}
```

new: 2025/12/21

This customize some aspects of the index code. ⟨index-specs⟩ can be any combination of

index cmd — In case the index package being used defines a distinct index command. This set's the actual index command, defaults to `\index`, used by the provided auxiliary index commands (see above). The given command must adhere to the same syntax of the original `\index` command (see [1]).

index specs — This allows to change *makeindex* [1] character specifiers. This expects a set of 4 parameters (from *makeindex*: *level*, *actual*, *encap* and *quote*). Its default is *index specs* = *{{!}{@}{|}{"}}*

index specs oc — This allows to change *makeindex* [1] open/close character specifiers. It expects a set of 4 parameters (from *makeindex*: *arg open*, *arg close*, *range open* and *range close*). Its default is *index specs oc* = *{{\{}{\}}{(}{)}}* . This isn't used, and is just a place holder in case further customization (indexes) is needed.

index specs others — This allows to change *makeindex* [1] others specifiers. It expects a set of 3 parameters (from *makeindex*: *escape*, *page compositor* and *index command*). Its default is *index specs others* = *{{\\}{-}{\indexentry }}*. This isn't used, and is just a place holder in case further customization (indexes) is needed.

> ***Note:*** This can only be used in the document preamble. It will raise an error, if used after `\begin{document}` .

3.2.1 Index Keys

When defining object types (see 3.3.3) or typesetting (see 3.5 and 3.6) the following keys can be used:

no index	To NOT include the items in the default index.
index	To include the items in the default index.
index name	Sets ⟨name⟩ for those items.
index group	Sets ⟨group⟩ for those items.
index prefix	Sets ⟨prefix⟩ for those items.
index gen group	Sets ⟨group⟩, using **\indexgenkey**, for those items.
index gen prefix	Sets ⟨prefix⟩, using **\indexgenkey**, for those items.

3.3 Object Type keys

⟨obj-types⟩ defines the applied format: font shape, bracketing, etc. to be applied. When using an ⟨obj-type⟩, first the associated ⟨format-group⟩ is applied, then the particular (if any) object format is applied.

3.3.1 Format Keys

Those are the primitive ⟨format-keys⟩ used when (re)defining ⟨format-groups⟩ and ⟨obj-types⟩ (see 3.3.5):

meta	Sets base format to typeset between angles.
xmeta	Sets base format to typeset *verbatim* between angles.
verb	Sets base format to typeset *verbatim*.
xverb	Sets base format to typeset *verbatim*, no spaces.
code	Sets base format to typeset *verbatim*, no spaces, replacing a TF by *TF*.
nofmt	In case of a redefinition, removes the base formatting. Note that, it only makes sense if applied at the same level, meaning, if the format was originally defined at group formatting level, it only can be removed at this level.
format	Sets the base format. Possible values: *meta*, *xmeta*, *verb*, *xverb*, *code*, *nofmt* or *none*, as above.
	Note: The *format* Key is just an alternative way of setting the base formatting. *none* is just an alias to *nofmt*.
slshape	To use a slanted font shape.
itshape	To use an italic font shape.
noshape	In case of a redefinition, removes the base shape. Note that, it only makes sense if applied at the same level, meaning, if shape was originally defined at group formatting level, it only can be removed at this level.
shape	Sets the font shape. Possible values: *itshape*, *italic*, *slshape*, *slanted*, *noshape* or *none*, as above.
	Note: The *shape* Key is just an alternative way of setting the font shape. *none* is just an alias to *noshape*.
shape preadj	Adds a (thin) space before each term in **\tsobj**, see 3.6. Possible values: *none*, *very thin*, *thin* or *mid*.
shape posadj	Adds a (thin) space after each term in **\tsobj**, see 3.6. Possible values: *none*, *very thin*, *thin* or *mid*.
	Note: These are meant for the case in which the italic or slanted shapes of the used font renders a character too close to an upright character.
lbracket	Sets the left bracket (when using **\tsargs**), see 3.6.
rbracket	Sets the right bracket (when using **\tsargs**), see 3.6.

color Sets the text color. **NB:** color's name as understood by *xcolor* package.

font Defaults to `\ttfamily`. Sets font family.

fsize Defaults to `\small`. Sets font size.

> ***Note:*** *font* and *fsize* shall receive a single command that absorbs no tokens.

index fmt Sets the index generating command (see 3.2).

> ***Note:*** Besides *index fmt* the other index keys (see 3.2.1) can also be used.
>
> ***Important:*** Except for *font*, *fsize* and *index fmt* all other keys will be expanded at definition time, including the others *index* keys.
>
> ***Note:*** The *index fmt* command shall absorb 4 parameters, like `\usercmd{name}{⟨prefix⟩}{⟨group⟩}{⟨item⟩}`. ⟨prefix⟩ will come from the key *index prefix*, ⟨group⟩ from the key *index group* and ⟨item⟩ will be the item to be indexed. ⟨name⟩ will come from the key *index name* (if not empty, ⟨name⟩ will be between brackets).
> For instance, having *index fmt* = `\usercmd`, `\tsobj ⟨[index name=iname, index prefix=pre, index group=grp]⟩ ⟨{\some }⟩` will result in the execution of `\usercmd{[iname]}{pre}{grp}{\some}`.

3.3.2 Format Groups

Using `\defgroupfmt` (see 3.3.5) one can (re-)define custom ⟨format-groups⟩. Predefined ones:

meta which sets *meta* and *color*

verb which sets *color*

code which sets *code*, *color* and *index* (*index fmt* = `\indexfmtcsraw`)

oarg which sets *meta* and *color*

syntax which sets *color*

env which sets *slshape*, *color* and *index* (*index fmt* = `\indexfmtraw`)

pkg which sets *slshape* and *color*

option which sets *color* and *index* (*index fmt* = `\indexfmtraw`)

keys which sets *slshape*, *color* and *index* (*index fmt* = `\indexfmtraw`)

values which sets *slshape* and *color*

defaultval which sets *color*

> ***Note:*** *color* was used in the list above just as a 'reminder' that a color is defined/associated with the given group, it can be changed with `\defgroupfmt`.
>
> ***Note:*** *index* and *index fmt* will only be set if the option `index` was used when loading this package, see 3.1.

3.3.3 Object Types

Object types are the ⟨keys⟩ used with `\tsobj` (and friends, see 3.6) defining the specific format to be used. With `\defobjectfmt` (see 3.3.5) one can (re-)define custom ⟨obj-types⟩. Predefined ones:

arg, meta based on (group) *meta*

verb, xverb based on (group) *verb* plus *verb* or *xverb*

marg based on (group) *meta* plus brackets

oarg, parg, xarg based on (group) *oarg* plus brackets

code, macro, function based on (group) *code*

syntax based on (group) *syntax*

keyval, key, keys based on (group) *keys*

value, values based on (group) *values*

option	based on (group) *option*
defaultval	based on (group) *defaultval*
env	based on (group) *env*
pkg, pack	based on (group) *pkg*

3.3.4 Command List Keys

The following keys are just some "sugar syntax" (to reduce a few keyboard strokes), and only make sense when typesetting (see 3.6) or describing (see 3.5) *expl* commands. They are only applied in case the base format (object type, see 3.3.1) is *code*, in which case a command (an item from ⟨csv-list⟩) can be any of the following:

1. \⟨cmd⟩
2. \⟨cmd⟩:⟨signature⟩
3. \⟨cmd⟩:⟨base-signature⟩ {⟨variants-list⟩}

In the first two cases, they will always be handle "as is". The third case depends on how the key *variants* is set (see below). Besides that, the keys *TF*, *noTF*, *pTF* and *nopTF* "helps" defining conditional variants of a base command.

> ***Attention:*** The keys bellow won't check for any *expl* convention. It's up to the user to use them correctly.

variants (defaults to *none*) This will set how the 3rd case is processed. Possible values are:

- *none* — The items (in a ⟨csv-list⟩) will be handled "as is" (no further special treatment). That's the default behaviour.
- *list* — A first entry will be generate with \⟨cmd⟩:⟨base-signature⟩ then for each ⟨sig-item⟩ in ⟨variants-list⟩ an entry \⟨cmd⟩:⟨sig-item⟩ will be generated. For example, `\tsobj[code,variants=list]{\exp_cmd:Nn{cn,cV}}` is equivalent to `\tsobj{\exp_cmd:Nn,\exp_cmd:cn,\exp_cmd:Cv}`
- *compact* — A first entry will be generate with \⟨cmd⟩:⟨base-signature⟩ then a second (or more, see remark below) entry will generated as \⟨cmd⟩:(⟨bnf-or⟩). For example, `\tsobj[code,variants=compact]{\exp_cmd:Nn{cn,cV}}` is equivalent to `\tsobj{\exp_cmd:Nn,\exp_cmd:(cn|cV)}`

  > ***Note:*** In the *compact* case, a dash "-" item will break down the "bnf or" in two entries at the dash entry. This helps avoiding extra-long entries. In the *list* case, a dash "-" item will be ignored.
  >
  > ***Note:*** In case of *list* or *compact* the generated entries will use a slightly fainted color (the color defined by the object type mixed with white)

TF — This will add a trailing *TF* to all items. The base name won't be listed as an item.

noTF — This will preserve the base(s) name and add the *TF* variant to all items.

pTF — This will add a trailing *TF* and a predicate `_p:` variant, to all items, and mark them as *EXP*. The base name won't be listed as an item.

nopTF — This will preserve the base(s) name and add the *TF* and predicate `_p:` variants to all items. Marking them as *EXP*.

> ***Note:*** The *pTF* and *nopTF* also implies *EXP* since the predicate variants must be expandable (see 3.5).

3.3.5 Customization

To create user defined groups/objects or change the predefined ones:

`\defgroupfmt` `\defgroupfmt {⟨format-group⟩}{⟨format-keys⟩}`

new: 2023/05/16

⟨format-group⟩ is the name of the new group (or the one being redefined, which can be one of the standard ones). ⟨format-keys⟩ is any combination of the keys from 3.3.1

For example, to change the color of all *obj-types* based on the *code* group (*code*, *macro* and *function* objects) to red, it's enought to `\defgroupfmt{code}{color=red}`.

`\dupgroupfmt` `\dupgroupfmt {⟨new-group⟩}{⟨org-group⟩}`

new: 2025/12/11

⟨new-group⟩ will be a copy of ⟨org-group⟩ definition at time of use. Both can be later chaged/re-defined independently of each other.

`\defobjectfmt` `\defobjectfmt {⟨obj-type⟩}{⟨format-group⟩}{⟨format-keys⟩}`

new: 2023/05/16

⟨obj-type⟩ is the name of the new ⟨object⟩ being defined (or redefined), ⟨format-group⟩ is the base group to be used (see 3.3.2). ⟨format-keys⟩ (see 3.3.1) allows further differentiation.

For instance, the many optional ⟨*arg⟩ are defined as follow:

```
\colorlet {c__codedesc_oarg_color} { gray!90!black }

\defgroupfmt  {oarg} { meta , color=c__codedesc_oarg_color }

\defobjectfmt {oarg} {oarg} { lbracket={[} , rbracket={]} }
\defobjectfmt {parg} {oarg} { lbracket={(} , rbracket={)} }
\defobjectfmt {xarg} {oarg} { lbracket={<} , rbracket={>} }
```

`\setcolorscheme` `\setcolorscheme {⟨color-key-list⟩}`

new: 2025/12/14

This allows to customize the default colors used by the many object types and format groups. Note that the given colors will be mixed with black. The key *brightness* set's the mixing proportion. The changes become effective at the point of use.

⟨color-key-list⟩ can be any combination of:

error	(default: red) Sets the error base color
verb	(default: black) Sets the verb base color
args	(default: white) Sets the args base color
code	(default: blue) Sets the code base color
keys	(default: teal) Sets the keys base color
values	(default: green) Sets the values base color
env	(default: green) Sets the env base color
pack	(default: green) Sets the pack base color
brightness	(default: 1) Sets the mixing proportion between each base color and black.
scheme	(Defaults to *scheme=default*) Selects a pre-set color scheme, see below, the default scheme sets all of the above to their default value.

`\newcolorscheme` `\newcolorscheme {⟨new-scheme⟩}{⟨color-key-list⟩}`

new: 2025/12/14

This creates/defines a ⟨new-scheme⟩ (⟨color-key-list⟩ as above) which can be later used as `\setcolorscheme{scheme=new-scheme}`

3.4 Locale

The following commands allows to customize the many 'labels' in use, in particular the auxiliary package *codedescsets* holds a few locale sets, the user is invited to submit translations for a specific case/language via a PR (Push Request) at `https://github.com/alceu-frigeri/codedescribe`

\setcodelabels `\setcodelabels {⟨labels-list⟩}`

new: 2025/11/22

`\setcodelabels` allows to change the many 'labels' used (like 'updated' in the *codedescribe* environment). See below for a complete list of possible labels.

The ⟨labels-list⟩ can be any combination of:

new	It set's the 'new' label used in the *codedescribe* environment.
update	It set's the 'update' label used in the *codedescribe* environment.
note	It set's the 'note' label used in the *codedescribe* environment.
and	It set's the 'and' label used by `\tsobj` (hint: last item separator).
or	It set's the 'or' label used by `\tsobj` (hint: last item separator).
months	It set's the month list used by `\tsdate`, see 3.9. NB.: it expects a list of names starting at 'January' and ending at 'December'.
label set	Selects a given set. No default. see below.

Note: The given ⟨labels-list⟩ doesn't need to be complete, though, only the given labels will be changed.

Note: The old `\selectlabelset{lang}` is (now) an alias to `\setcodelabels {label set=lang}`, and will raise a warning if called (deprecation).

\newlabelset `\newlabelset {⟨lang⟩} {⟨labels-list⟩}`

new: 2025/11/22

This creates/defines a new label's set (named as ⟨lang⟩), ⟨labels-list⟩ as above, which can be later used as `\setcodelabels{labelset=lang}`

Note: `\newlabelset` is used in the auxiliary package *codedescsets* to predefine some sets, which can then be used as a package option, see 3.1.

Note: `\newlabelset` can be used to redefine a given set, though, if doing so, one has to provide all labels. The old (if any) definitions will be erased. No warnings given.

For example, this sets a new label set for German. In fact, since this is defined in the package *codedescsets* this label set can be used at load time, see 3.1.

```
\newlabelset {german}
  {
    new     = neu           ,
    update  = aktualisiert  ,
    note    = NB            ,
    remark  = Hinweis       ,
    and     = und           ,
    or      = oder          ,
    months  =
      {
        Januar, Februar, März, April,
        Mai, Juni, Juli, August,
        September, Oktober, November, Dezember
      }
  }
```

3.5 Environments

codedescribe

new: 2023/05/01
updated: 2023/05/01
updated: 2024/02/16
updated: 2025/09/25
NB: a note example

`\begin{codedescribe}` [⟨obj-keys⟩] {⟨csv-list⟩}
...
`\end{codedescribe}`

This is the main environment to describe *Commands*, *Variables*, *Environments*, *etc.* ⟨csv-list⟩ items will be listed in the left margin. The *codesyntax* will be attached to it's right, and the rest of the text will be below them, with the usual text width. The optional ⟨obj-keys⟩ defaults to just *code*, it can be any object type as defined at 3.3.3 (and 3.3.5), index keys (see 3.2.1), command list keys (see 3.3.4) or the following:

new	To add a *new* line.
update	To add an *updated* line.
note	To add a *NB* line.
keys seq	Possible values are *grouped* or *as is*. By default the keys *new*, *update* and *note* are grouped together, first all *new* keys, then all *update* keys and lastly all *note* keys. With *as is* keys will respect the used sequence. The default can be changed with the package option `describe keys`, see 3.1.
rulecolor	For instance `\begin{codedescribe}[rulecolor=white]` will suppress the rules.
EXP	A star ★ will be added to all items, signaling the commands are fully expandable.
rEXP	A hollow star ☆ will be added to all items, signaling the commands are restricted expandable.
force margin	If set, ⟨csv-list⟩ items will be listed in the margin, regardless of their width.

Note: The keys *new*, *update* and *note* can be used multiple times. (2024/02/16)

Note: If using one of these keys the user must also provide an object type. *code* is the solely default IF nothing else is provided.

Attention: The *codedescribe* environment 'acts' as a single block! That assures the margin block, the *codesyntax* environment (block) and the following text (inside the *codedescribe* environment) will always stay in the same page.

Attention: If the items don't fit in the margin, the ⟨csv-list⟩ will advance towards the text window (up to approximately half of text width plus margin width), reducing the horizontal space of the *codesyntax* block. This can be changed with the *force margin*, in which case the ⟨csv-list⟩ will always be at the margin, growing leftwards (might end outside the page).

Note: With the `strict` package option, an error will be raised if used inside another *codedescribe* environment. Otherwise a warning will be raised. (it's safe to do so, but it doesn't make much sense).

codesyntax

updated: 2025/09/25
updated: 2025/11/25

`\begin{codesyntax}` [⟨obj-type⟩]
...
`\end{codesyntax}`

The *codesyntax* environment sets the fontsize and activates `\obeylines`, `\obeyspaces` , so one can list macros/cmds/keys use, one per line. The content will be formatted as defined by ⟨obj-type⟩ (defaults to `syntax`). ⟨obj-type⟩ can be any object from 3.3.3 (or 3.3.5). For a *verbatim* alternative, see *codesyntax** below.

Note: *codesyntax* and/or *codesyntax** environments shall appear only once, inside of a *codedescribe* environment. Remember, this is not a verbatim environment!

Note: With the `strict` package option an error will be raised if used outside a *codedescribe* environment, or more than once inside. Otherwise a warning will be raised.

For example, the code for *codedescribe* (previous entry) is:

```
\begin{codedescribe}[ env , new=2023/05/01, update=2023/05/01, note={a note example}, update
      =2024/02/16, update=2025/09/25]{codedescribe}
  \begin{codesyntax}
    \tsmacro{\begin{codedescribe}}[obj-type]{csv-list}
    \ldots
    \tsmacro{\end{codedescribe}}{}
  \end{codesyntax}
  This is the main ...
\end{codedescribe}
```

*codesyntax**

new: 2025/12/18

```
\begin{codesyntax*} [⟨code-keys⟩]
...
\end{codesyntax*}
```

The *codesyntax** is a true *verbatim* environment (derived from *listings* package, see [5]). ⟨code-keys⟩ can be any valid code key from 2.3.2, and syntax highlight will be applied (see 2.3). The background color will always be white, whilst line numbering will be suppressed. For a non *verbatim* alternative, see *codesyntax* above.

> ***Note:*** If `nolisting` package option is set, this environment won't be defined.
>
> ***Note:*** *codesyntax* and/or *codesyntax** environments shall appear only once, inside of a *codedescribe* environment.
>
> ***Note:*** With the `strict` package option an error will be raised if used outside a *codedescribe* environment, or more than once inside. Otherwise a warning will be raised.

describelist
*describelist**

```
\begin{describelist} [⟨item-indent⟩] {⟨obj-type⟩}
  \describe {⟨item-name⟩} {⟨item-description⟩}
  \describe {⟨item-name⟩} {⟨item-description⟩}
\end{describelist}
```

This sets a *description* like 'list'. In the non-star version the ⟨items-name⟩ will be typeset on the margin. In the star version, ⟨item-description⟩ will be indented by ⟨item-indent⟩ (defaults to: 20mm). ⟨obj-type⟩ defines the object-type format used to typeset ⟨item-name⟩, it can be any object from 3.3.3 (or 3.3.5) and index keys (see 3.2.1).

\describe

```
\describe {⟨item-name⟩} {⟨item-description⟩}
```

This is the *describelist* companion macro. In case of the *describe**, ⟨item-name⟩ will be typeset in a box ⟨item-indent⟩ wide, so that ⟨item-description⟩ will be fully indented, otherwise ⟨item-name⟩ will be typed at the margin.

> ***Note:*** An error will be raised (undefined control sequence) if called outside of a *describelist* or *describelist** environment.

3.6 Typeset Commands

\typesetobj
\tsobj

updated: 2025/05/29

```
\typesetobj [⟨obj-type⟩] {⟨csv-list⟩}
\tsobj [⟨obj-type⟩] {⟨csv-list⟩}
```

This is the main typesetting command, each term of ⟨csv-list⟩ will be separated by a comma and formatted as defined by ⟨obj-type⟩ (defaults to *code*). ⟨obj-type⟩ can be any object from 3.3.3 (or 3.3.5), index keys (see 3.2.1) and the following keys:

mid sep	To change the item separator. Defaults to a comma, can be anything.
comma	To set the separator between the last two items to a comma.
sep	To change the separator between the last two items. Defaults to "and".
or	To set the separator between the last two items to "or".
bnf or	To produce a bnf style or list, like [`abc`|`xdh`|`htf`|`hrf`].
meta or	To produce a bnf style or list between angles, like ⟨`abc`|`xdh`|`htf`|`hrf`⟩.
par or	To produce a bnf style or list between parentheses, like (`abc`|`xdh`|`htf`|`hrf`).

Note: If *shape preadj* and/or *shape posadj* are set (see 3.3.1, a (thin) space will be added before and/or after each term of ⟨csv-list⟩.

\typesetargs
\tsargs

```
\typesetargs [⟨obj-type⟩] {⟨csv-list⟩}
\tsargs [⟨obj-type⟩] {⟨csv-list⟩}
```

These will typeset ⟨csv-list⟩ as a list of parameters, like [⟨arg1⟩] [⟨arg2⟩] [⟨arg3⟩], or {⟨arg1⟩} {⟨arg2⟩} {⟨arg3⟩}, etc. ⟨obj-type⟩ defines the formating AND kind of brackets used (see 3.3): [] for optional arguments (oarg), {} for mandatory arguments (marg), and so on.

\typesetmacro
\tsmacro

```
\typesetmacro {⟨macro-list⟩} [⟨oargs-list⟩] {⟨margs-list⟩}
\tsmacro {⟨macro-list⟩} [⟨oargs-list⟩] {⟨margs-list⟩}
```

These are just short-cuts for
\tsobj[code]{macro-list} \tsargs[oarg]{oargs-list} \tsargs[marg]{margs-list}.

\typesetmeta
\tsmeta

```
\typesetmeta {⟨name⟩}
\tsmeta {⟨name⟩}
```

These will just typeset ⟨name⟩ between left/right 'angles' (no further formatting).

\typesetverb
\tsverb

```
\typesetverb [⟨obj-type⟩] {⟨verbatim text⟩}
\tsverb [⟨obj-type⟩] {⟨verbatim text⟩}
```

Typesets ⟨verbatim text⟩ as is. ⟨obj-type⟩ defines the used format. The difference with \tsobj[verb]{something} is that ⟨verbatim text⟩ can contain commas (which, otherwise, would be interpreted as a list separator by \tsobj.

Note: This is meant for short expressions, and not multi-line, complex code (one is better of, then, using 2.3). ⟨verbatim text⟩ must be balanced !

3.7 Note/Remark Commands

\typesetmarginnote
\tsmarginnote

```
\typesetmarginnote {⟨note⟩}
\tsmarginnote {⟨note⟩}
```

Typesets a small note at the margin.

Note: Don't try to use these inside one of this packages environments, like *tsremark* or *codedescribe*, given the way they are constructed (*expl* coffins) it will result in a *Float(s) lost* error.

tsremark

```
\begin{tsremark} [⟨NB⟩]
\end{tsremark}
```

The environment body will be typeset as a text note. ⟨NB⟩ (defaults to Note:) is the note begin (in boldface). For instance:

```
Sample text. Sample test.
 \begin{tsremark}[N.B.]
   This is an example.
 \end{tsremark}
```

Sample text. Sample test.

N.B. This is an example.

3.8 Shortcuts (experimental)

This is marked as experimental because the actual chosen short cuts might change, for instance, if it crashes badly with some other package. As of now, the current implementation tries to minimize any side effect, only kicking in if, and only if, one of the given patterns is found. Moreover, once deactivated, `\tsOff`, any previous code is fully restored.

`\tsOn` `\tsOn`
`\tsOff` `\tsOff`

This will switch the 'shortcuts' on and off. Currently, only the character '!' is affected. `\tsOn` preserves its status (if active or not, and related code, if any), so that `\tsOff` can restore its full definition and status.

> ***Note:*** `\tsOn` won't try to patch the current (if any) active definition of '!', but just save it (to be restored by `\tsOff`), before setting its own code. Moreover, whilst active, if the use don't fit any of the given patterns (below), the previous active code (if any, and active) will be executed, or just '!' if it wasn't active.
>
> ***Note:*** If '!' already had an associated code, a warning will be raised, showing the previous code.

`!:`	`!:⟨[obj-type]{csv-list}⟩` ↔ `\tsobj`
`!::`	`!::⟨[obj-type]{csv-list}⟩` ↔ `\tsargs`
`!!`	`!!⟨[obj-type]{verbatim text}⟩` ↔ `\tsverb`
`!!:`	`!!:⟨{name}⟩` ↔ `\tsmeta`
`!!::`	`!!::⟨{note}⟩` ↔ `\tsmarginnote`
`!!?`	`!!?⟨{macro-list}[oargs-list]{margs-list}⟩` ↔ `\tsmacro`

new: 2025/12/29
NB: active char

Once active (`\tsOn`), `!:` is a shortcut for `\tsobj`, including it's optional parameter. Same for `!::` (`\tsargs`), and the others.

> ***Note:*** To reduce undesirable side effects, no space (or any other character besides the ones shown) is allowed between the first '!' and either the '[' or '{', for instance `!!:{some}` will typeset ⟨some⟩, but not `!!:{some}` or `!!:{some}` or `!!:some`.
>
> ***Note:*** If none of these patterns are recognized it will either leave the '!' character or execute its previous code (if it was active). Same for the companions colon, ':', and question mark, '?', peeked in the process.

3.9 Auxiliary Commands and Environment

In case the Document Class being used redefines the `\maketitle` command and/or *abstract* environment, alternatives are provided (based on the article class).

`\typesettitle` `\typesettitle {⟨title-keys⟩}`
`\tstitle` `\tstitle {⟨title-keys⟩}`

This is based on the `\maketitle` from the *article* class. The ⟨title-keys⟩ are:

title	The title.
author	Author's name. It's possible to use the `\footnote` command in it.
date	Title's date.

> ***Note:*** The `\footnote` (inside this) will use an uniquely assigned counter, starting at one, each time this is used (to avoid *hyperref* warnings).

tsabstract

```
\begin{tsabstract}
...
\end{tsabstract}
```

This is the *abstract* environment from the *article* class.

`\typesetdate` `\typesetdate`
`\tsdate` `\tsdate`

new: 2023/05/16

This provides the current date (in Month Year format).

4 codelstlang Package

This is an auxiliary package (which can be used on its own). It assumes the package *listings* was already loaded, and just defines the following TEX dialects, all of them derived from *listings [LaTeX]TeX*:

[l3kernelsign]TeX Most/all *expl* keys found in the *l3kernel*[7] packages, including signatures.

[l3expsign]TeX Most/all *expl* keys found in the *l3kernel* experimental packages, including signatures.

[l3amssign]TeX Most/all *expl* keys found in the *ams*, *siunitx* and related packages, including signatures.

[l3pgfsign]TeX Most/all *expl* keys found in the *pgf* and related packages, including signatures.

[l3bibtexsign]TeX Most/all *expl* keys found in the *bibtex*, *biblatex* and related packages, including signatures.

Note: The underscore '_' and colon ':' have to be defined as letters (*letter = { _ , : }*, see 2.3.2). Take note that these dialects are quite large, due the many signatures variants.

[l3kernel]TeX Most/all *expl* keys found in the *l3kernel* packages, without signatures.

[l3exp]TeX Most/all *expl* keys found in the *l3kernel* experimental packages, without signatures.

[l3ams]TeX Most/all *expl* keys found in the *ams*, *siunitx* and related packages, without signatures.

[l3pgf]TeX Most/all *expl* keys found in the *pgf* and related packages, without signatures.

[l3bibtex]TeX Most/all *expl* keys found in the *bibtex*, *biblatex* and related packages, without signatures.

Note: The underscore '_' has to be defined as letter (*letter = { _ }*, but not the colon ':', see 2.3.2). These are more compact versions of the previous ones.

[kernel]TeX Most/all document level keys found in the *kernel* packages.

[xpacks]TeX Most/all document level keys found in the *x** packages, like *xkeyval*, *xpatch*, *xcolor* etc.

[ams]TeX Most/all document level keys found in the *ams*, *siunitx* and related packages.

[pgf]TeX Most/all document level keys found in the *pgf* and related packages.

[pgfplots]TeX Most/all document level keys found in the *pgfplots* and related packages.

[bibtex]TeX Most/all document level keys found in the *bibtex*, *biblatex* and related packages.

[babel]TeX Most/all document level keys found in the *babel* and related packages.

[hyperref]TeX Most/all document level keys found in the *hyperref* and related packages.

Note: These are usual document level, LATEX 2_ε, commands. In particular none of them includes any '@' symbol.

References

[1] Pehong Chen and Michael A. Harrison. "Index Preparation and Processing". In: *Software: Practice and Experience* vol.19 (Sept. 1988). Can be found at `https://ctan.org/tex-archive/indexing/makeindex/paper/ind.pdf`. (Visited on 12/16/2025).

[2] Alceu Frigeri. *The pkginfograb Package*. 2025. URL: `https://mirrors.ctan.org/macros/latex/contrib/pkginfograb/doc/pkginfograb.pdf` (visited on 12/16/2025).

[3] Alceu Frigeri. *The xpeedahead Package*. 2025. URL: `https://mirrors.ctan.org/macros/latex/contrib/xpeekahead/doc/xpeekahead.pdf` (visited on 12/16/2025).

[4] Pablo González. *SCONTENTS - Stores LaTeX Contents*. 2024. URL: `https://mirrors.ctan.org/macros/latex/contrib/scontents/scontents.pdf` (visited on 03/10/2025).

[5] Jobst Hoffmann. *The Listings Package*. 2024. URL: `https://mirrors.ctan.org/macros/latex/contrib/listings/listings.pdf` (visited on 03/10/2025).

[6] Uwe Kern. *Extending LaTeX's color facilities: the xcolor package*. 2024. URL: `https://mirrors.ctan.org/macros/latex/contrib/xcolor/xcolor.pdf` (visited on 11/20/2025).

[7] The LaTeX Project. *The LaTeX3 Interfaces*. 2025. URL: https://mirrors.ctan.org/macros/latex/required/l3kernel/interface3.pdf (visited on 11/20/2025).

[8] Walter Schmidt. *Using common PostScript fonts with LaTeX*. 2020. URL: https://mirrors.ctan.org/macros/latex/required/psnfss/psnfss2e.pdf (visited on 11/20/2025).