The codedescribe and codelisting Packages
Version 1.20

Alceu Frigeri*

December 2025

Abstract

This package is designed to be as class independent as possible, depending only on *expl3*, *scontents*, *listing*, *xpeekahead*, *pifont* and *xcolor*. Unlike other packages of the kind, 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 marginpar"), or underlying macros, so it should be possible to use these with any/most document classes.

Contents

1 Introduction

This package aims to document both `Document` level (i.e. final user) commands, as well `Package/Class` level commands. It's fully implemented using *expl3* syntax and structures, in special *l3coffins*, *l3seq* and *l3keys*. Besides those *scontents* and *listing* packages (see [1] and [2]) are used to typeset code snippets. The package *pifont* is needed just to typeset those (open)stars, in case one wants to mark a command as (restricted) expandable.

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

Besides those, *xcolor* is needed to switch colors, and *xpeekahead* for spacing fine tuning. 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.

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), and *codelstlang* defines a series of *listings* TeX dialects.

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`.

1.2 Current Version

All packages (*codedescribe*, *codelisting*, *codelstlang* and *codedescsets*) share the same version, currently: 1.20. Those packages are fairly stable, and given the ⟨obj-type⟩ mechanism (see 3.2) they can be easily extended without changing their interface.

2 codelisting Package

It loads: *listings*, *scontents* and *xpeekahead*, defines an environment: *codestore*, a few commands for listing/demo code: `\tscode`, `\tsmergedcode`, `\tsdemo`, `\tsresult` and `\tsexec` and 2 auxiliary commands: `\setcodekeys` and `\setnewcodekey`.

2.1 Package Options

The following options are also *codedescribe* options, see 3.1.

`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 [2] 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}
   %%
```

[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

2.2 In Memory Code Storage

Thanks to *scontents* (*expl3* based) it's possible to store LaTeX code snippets in a *expl3* sequence variable.

codestore

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

This environment is an alias to *scontents* environment (from *scontents* package, see [1]), 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.
>
> LaTeX Code:

```
%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**

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

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

\tscode* just typesets ⟨st-name⟩ (created with *codestore*) verbatim with syntax highlight (from *listings* package [2]). 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:

LaTeX Code:

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

This will just typesets \tsobj[key]{stmeta}:
\tscode*[codeprefix={Sample Code:}] {stmeta}

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

LaTeX Result:

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	inner sample result
`1  Some \LaTeX{} coding, for example: \ldots.`	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 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/group.

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

new: 2025/05/01

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.

lststyle *lststyle*

new: 2025/12/11

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
texcs
texcsstyle

settexcs, settexcs2, settexcs3 and *settexcs4*
texcs, texcs2, texcs3 and *texcs4*
texcsstyle, texcs2style, texcs3style and *texcs4style*

updated: 2025/05/01

These define sets of LaTeX commands (csnames, sans the preceding slash bar), the *set* variants initialize/redefine those sets (an empty list, clears the set), while the others extend those sets. The *style* ones redefines the command display style (an empty ⟨value⟩ resets the style to it's default).

setkeywd
keywd
keywdstyle

setkeywd, setkeywd2, setkeywd3 and *setkeywd4*
keywd, keywd2, keywd3 and *keywd4*
keywdstyle, keywd2style, keywd3style and *keywd4style*

updated: 2025/05/01

Same for other *keywords* sets.

setemph
emph
emphstyle

setemph, setemph2, setemph3 and *setemph4*
emph, emph2, emph3 and *emph4*
emphstyle, emph2style, emph3style and *emph4style*

updated: 2025/05/01

for some extra emphasis sets.

letter
other

letter and *other*

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 variants, but just the base name of an *expl3* function.

numbers
numberstyle

numbers and *numberstyle*

numbers possible values are *none* (default) and *left* (to add tinny numbers to the left of the listing). With *numberstyle* one can redefine the numbering style.

stringstyle
commentstyle

stringstyle and *commentstyle*

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

bckgndcolor

bckgndcolor

to change the listing background's color.

codeprefix
resultprefix

codeprefix and *resultprefix*

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

parindent

parindent

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

ruleht

ruleht

When typesetting the 'code demo' (`\tsdemo`) a set of rules are drawn. The Default, 1, equals to `\arrayrulewidth` (usually 0.4pt).

basicstyle

basicstyle

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 macro, or environment, or variable, or key ...) 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 it 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 3.3. 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 3.4) are scaled up from this. It defaults to the font size at load time.

strict Package Warnings will be reported as Package Errors.

silence (new: 2025/11/22, defaults to 18.89999pt) This will suppress some annoying bad boxes warnings. Given the way environments at 3.4 are defined, with *expl* coffins, TeX sometimes thinks they are too wide, when they are not. This just sets *\hfuzz*.

color scheme Possible values: *black*, *default*, *brighter* and *darker*. This will adjust the initial color configuration for the many format groups/objects (see 3.2.1). *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.

load xtra dialects This will be passed over to *codelisting* package (if loaded). See 2.1.

TeX dialects This will be passed over to *codelisting* package (if loaded). See 2.1.

> ***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.
>
> ***Note:*** *color scheme* doesn't affect *codelisting* / *listings* colors.

3.2 Object Type keys

⟨obj-types⟩ defines the applied format, which is defined in terms of ⟨format-groups⟩. Both defines the formatting function, font shape, bracketing, etc. to be applied. When using a ⟨obj-type⟩, first the associated ⟨format-group⟩ is applied, then the particular (if any) object format is applied.

3.2.1 Format Keys

Those are the primitive ⟨format-keys⟩ used when (re)defining ⟨format-groups⟩ and ⟨obj-types⟩ (see 3.2.4):

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 3.5. Possible values: *none*, *very thin*, *thin* or *mid*,
shape posadj	adds a (thin) space after each term in \tsobj, see 3.5. 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 a upright character.

lbracket	sets the left bracket (when using \tsargs), see 3.5,
rbracket	sets the right bracket (when using \tsargs), see 3.5,
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.

3.2.2 Format Groups

Using \defgroupfmt (see 3.2.4) one can (re-)define custom ⟨format-groups⟩. The following ones are predefined:

meta	which sets *meta* and *color*
verb	which sets *color*
oarg	which sets *meta* and *color*
code	which sets *code* and *color*
syntax	which sets *color*
keyval	which sets *slshape* and *color*
option	which sets *color*
defaultval	which sets *color*
env	which sets *slshape* and *color*
pkg	which sets *slshape* and *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.

3.2.3 Object Types

Object types are the ⟨keys⟩ used with \tsobj (and friends, see 3.5) defining the specific format to be used. With \defobjectfmt (see 3.2.4) one can (re-)define custom ⟨obj-types⟩. The predefined ones are:

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, value, values	based on (group) *keyval*

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

3.2.4 Customization

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

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

new: 2023/05/16

⟨format-group⟩ is the name of the new group (or one being redefined, which can be one of the standard ones). ⟨format-keys⟩ is any combination of the keys from 3.2.1

For example, one can redefine the *code group* standard color with `\defgroupfmt{code}{color=red}` and all *obj-types* based on it will be typeset in red (in the standard case: *code*, *macro* and *function* objects).

`\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 3.2.2). ⟨format-keys⟩ (see 3.2.1) allow for 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={>} }
```

3.3 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⟩}`
`\newlabelset` `\newlabelset {⟨lang⟩}{⟨labels-list⟩}`
`\selectlabelset` `\selectlabelset {⟨lang⟩}`

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.
`\newlabelset` will create a label's set (named as ⟨lang⟩) for later use, while `\selectlabelset` will select (activate) the given set. All those commands can be used at any time.

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 3.7. NB.: it expects a list of names starting at 'January' and ending at 'December'.

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

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 when loading this package, 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
      }
  }
```

3.4 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 optional ⟨obj-keys⟩ defaults to just *code*, it can be any object type as defined at 3.2.3 (and 3.2.4), besides the following keys:

new	To add a *new* line.
update	To add an *updated* line.
note	To add a *NB* line.
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.

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

Note: If using one of the keys *new*, *update*, *note*, *rulecolor*, *EXP* or *rEXP* the user must also provide an object type. *code* is the solely default IF nothing else is provided.

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 3.2.3 (or 3.2.4)

Note: *codesyntax* environment shall appear only once, inside of a *codedescribe* environment. Take note, as well, 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:

LaTeX Code:

```
\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}
```

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 marginpar. 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⟩.

\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-ident⟩ wide, so that ⟨item-description⟩ will be fully indented, otherwise ⟨item-name⟩ will be typed in the marginpar.

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

3.5 Typeset Commands

Note that, in the following commands, ⟨obj-type⟩ refers to any object type defined in 3.2.3 and 3.2.4.

\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 3.2.3 (or 3.2.4) and the following keys:

mid sep	To change the item separator. Defaults to a comma, can be anything.
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".
comma	To set the separator between the last two items to a comma.
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* or *shape posadj* are set (see 3.2.1, 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 3.2): `[]` 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 (verbatim...). ⟨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 ! otherwise, some low level TEX errors might pop out.

3.6 Note/Remark Commands

`\typesetmarginnote` `\tsmarginnote`

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

Typesets a small note at the margin.

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:

LATEX Code:

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

LATEX Result:

Sample text. Sample test.
N.B. This is an example.

3.7 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` `\tstitle`

```
\typesettitle {⟨title-keys⟩}
\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*[3] 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 2.3.1). Take note that those 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 2.3.1). Those 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: Those are usual document level, LATEX 2_ε, commands. In particular none of them includes any '@' symbol.

References

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

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

[3] The LATEX Project. *The LATEX3 Interfaces*. 2025. URL: `https://mirrors.ctan.org/macros/latex/required/l3kernel/interface3.pdf` (visited on 11/20/2025).