The codedescribe and codelisting Packages
Version 1.11

Alceu Frigeri*

September 2025

Abstract

This package is designed to be as class independent as possible, depending only on *expl3*, *scontents*, *listing* and *pifont*. 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 that it can be used with any document class.

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.

No other package/class is needed, any class can be used as the base one, which allows to demonstrate the documented commands with any desired layout.

codelisting defines a few macros to display and demonstrate LaTeX code (using *listings* and *scontents*), whilst *codedescribe* defines a series of macros to display/enumerate macros and environments (somewhat resembling the *doc3* style).

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

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

This doc regards to *codedescribe* version 1.11 and *codelisting* version 1.11. Those two 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 requires two packages: *listings* and *scontents*, defines an environment: *codestore*, commands for listing/demo code: `\tscode`, `\tsmergedcode`, `\tsdemo`, `\tsresult` and `\tsexec` and 2 auxiliary commands: `\setcodekeys` and `\setnewcodekey`.

2.1 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⟩ ⟨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 are not 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}
```

2.2 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*), in verbatim mode and 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
`Some \LaTeX~coding, for example: \ldots.`	Some LaTeX coding, for example: . . . .

`\tsmergedcode*`

new: 2025/04/29

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

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`

new: 2025/04/29

`\tsexec` {⟨st-name⟩} [⟨index⟩]

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.

2.2.1 Code Keys

`\setcodekeys`

`\setcodekeys {⟨code-keys⟩}`

One has the option to set ⟨code-keys⟩ per `\tscode`, `\tsmergedcode`, `\tsdemo` and `\tsresult` call (see 2.2), 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`
new: 2025-05-01

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

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.

settexcs
texcs
texcsstyle
updated: 2025-05-01

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

Those define sets of LaTeX commands (csnames), 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
updated: 2025-05-01

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

Same for other *keywords* sets.

setemph
emph
emphstyle
updated: 2025-05-01

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

for some extra emphasis sets.

letter
other
new: 2025-05-13

letter and *other*

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* 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
codestyle

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' LaTeX code (`\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, with object kind (if a macro, or a function, or environment, or variable, or key ...) as 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.

`base skip` Changes the base skip, all skips (used by the environments at 3.3) are scaled from this. It defaults to the font size at load time.

`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.

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⟩ wich defines a formatting function, font shape, bracketing, etc. to be 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	to typeset between angles,
xmeta	to typeset *verbatim* between angles,
verb	to typeset *verbatim*,
xverb	to typeset *verbatim*, suppressing all spaces,
code	to typeset *verbatim*, suppressing all spaces and replacing a TF by <u>TF</u>,
nofmt	in case of a redefinition, to remove the 'base' formatting,
slshape	to use a slanted font shape,
itshape	to use an italic font shape,
noshape	in case of a redefinition, to remove the 'base' shape,
lbracket	defines the left bracket (when using `\tsargs`). **Note:** this key must have an associated value,
rbracket	defines the right bracket (when using `\tsargs`). **Note:** this key must have an associated value,

color	defines the text color. **Note:** this key must have an associated value (a color, as understood by *xcolor* package).

3.2.2 Format Groups

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

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 3.4) defining the specific formatting 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, 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`
new: 2023/05/16

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

⟨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).

`\defobjectfmt`
new: 2023/05/16

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

⟨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 Environments

codedescribe

new: 2023/05/01
updated: 2023/05/01
updated: 2024/02/16
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)

codesyntax

```
\begin{codesyntax}
...
\end{codesyntax}
```

The *codesyntax* environment sets the fontsize and activates `\obeylines`, `\obeyspaces` , so one can list macros/cmds/keys use, one per line.

Note: *codesyntax* environment shall appear only once, inside of a *codedescribe* environment. An error will be raised if called outside. Take note, as well, this is not a verbatim environment!

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]{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 if called outside of a *describelist* or *describelist** environment.

3.4 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”.
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).

\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 other 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 in `\tsobj`.

Note: This is meant for short expressions, and not multi-line, complex code (one is better of, then, using 2.2). ⟨verbatim text⟩ must be balanced ! otherwise, some low level TEX errors may pop out.

3.5 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:	LaTeX Result:

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

3.6 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.

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

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

`\typesetdate`
`\tsdate`
new: 2023/05/16

`\typesetdate`
`\tsdate`

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

References

[1] Pablo González. *SCONTENTS - Stores LaTeX Contents*. 2024, p. 48. URL: `http://mirrors.ctan.org/macros/latex/contrib/scontents/scontents.pdf` (visited on 03/10/2025).

[2] Jobst Hoffmann. *The Listings Package*. 2024, p. 65. URL: `http://mirrors.ctan.org/macros/latex/contrib/listings/listings.pdf` (visited on 03/10/2025).