

# The codedescribe and codelisting Packages

Version 1.5

Alceu Frigeri\*

February 2025

## Abstract

This documentation 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 about page layout is made (besides ‘having a marginpar’), or underlying macros, so that it can be used with any document class.

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Single versus Multi-column Classes . . . . .	2
1.2	Current Version . . . . .	2
<b>2</b>	<b>codelisting Package</b>	<b>2</b>
2.1	In Memory Code Storage . . . . .	2
2.2	Code Display/Demo . . . . .	2
2.2.1	Code Keys . . . . .	3
<b>3</b>	<b>codedescribe Package</b>	<b>4</b>
3.1	Package Options . . . . .	4
3.2	Object Type keys . . . . .	4
3.2.1	Format Keys . . . . .	4
3.2.2	Format Groups . . . . .	5
3.2.3	Object Types . . . . .	5
3.2.4	Customization . . . . .	5
3.3	Environments . . . . .	6
3.4	Commands . . . . .	7
3.5	Auxiliary Command / Environment . . . . .	8

## 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 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 L<sup>A</sup>T<sub>E</sub>X 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 default 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 1.5 and `codelisting` version 1.5. Those two packages are fairly stable, and given the `<obj-type>` mechanism (see below, 3.2) it can be easily extended without changing it's interface.

# 2 codelisting Package

It requires two packages: `listings` and `scontents`, defines an environment: `codestore` and 3 main commands: `\tscode`, `\tsdemo` and `\tsresult` and 1 auxiliary command: `\setcodekeys`.

## 2.1 In Memory Code Storage

Thanks to `scontents` (`expl3` based) it's possible to store L<sup>A</sup>T<sub>E</sub>X code snippets in a `expl3` key.

`codestore`

This environment is an alias to `scontents` environment (from `scontents` package), 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 'by Default' that the environment body shall be stored in it (for use with `\tscode` and `\tsdemo`).

## 2.2 Code Display/Demo

---

`\setcodekeys`

One has the option to set `<code-keys>` (see 2.2.1) per `\tscode`/`\tsdemo` call, 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.

---

`\tscode*`

`\tsdemo*`

`\tsresult*`

---

update: 2024/01/06

`\tscode` just typesets `<st-name>` (the key-name created with `stcode`), in verbatim mode with syntax highlight. 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 it *executes* said code. The non-start versions place them side-by-side, whilst the star versions places one following the other.

(new 2024/01/06) `\tsresult*` only *executes* said code. The non-start versions centers it and use just half of the base line, whilst the star versions uses the full text width.

For Example:

L<sup>A</sup>T<sub>E</sub>X 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}
```

L<sup>A</sup>T<sub>E</sub>X 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
<pre>1 Some \LaTeX~coding, for example: \ldots.</pre>	Some L <sup>A</sup> T <sub>E</sub> X coding, for example: . . . .

---

### 2.2.1 Code Keys

Using a *key=value* syntax, one can fine tune *listings* syntax highlight.

---

*settexc*  
*texcs*  
*texcsstyle*

Those define sets of L<sup>A</sup>T<sub>E</sub>X 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*

Same for other *keywords* sets.

---

*setemph*  
*emph*  
*emphstyle*

for some extra emphasis sets.

---

*numbers*  
*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*

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

---

*bckgndcolor*

to change the listing background's color.

---

*codeprefix*  
*resultprefix*

those set the *codeprefix* (default: L<sup>A</sup>T<sub>E</sub>X Code:) and *resultprefix* (default: L<sup>A</sup>T<sub>E</sub>X Result:)

---

---

*parindent*

Sets the indentation to be used when 'demonstrating' L<sup>A</sup>T<sub>E</sub>X code (`\tsdemo`). Defaults to whatever value `\parindent` was when the package was first loaded.

---

---

*ruleht*

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

---

---

*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, having the 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

It has a single package option:

`nolisting` it will suppress the `codelisting` package load. In case it's not necessary or one wants to use a differen package for L<sup>A</sup>T<sub>E</sub>X code listing.

### 3.2 Object Type keys

The applied text format is defined in terms of `<obj-types>`, which are defined in terms of `<format-groups>` and each one defines a 'formatting function', 'font shape', bracketing, etc. to be applied.

#### 3.2.1 Format Keys

There is a set of primitive `<format-keys>` used to define `<format-groups>` and `<obj-types>`, which are:

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

#### 3.2.2 Format Groups

Using `\defgroupfmt` one can (re-)define custom `<format-groups>`. There is, though, a set of pre-defined ones as follow:

<i>meta</i>	which sets <i>meta</i> and <i>color</i>
<i>verb</i>	which sets <i>color</i>
<i>oarg</i>	which sets <i>meta</i> and <i>color</i>
<i>code</i>	which sets <i>code</i> and <i>color</i>
<i>syntax</i>	which sets <i>color</i>
<i>keyval</i>	which sets <i>slshape</i> and <i>color</i>
<i>option</i>	which sets <i>color</i>
<i>defaultval</i>	which sets <i>color</i>
<i>env</i>	which sets <i>slshape</i> and <i>color</i>
<i>pkg</i>	which sets <i>slshape</i> and <i>color</i>

**Note:** *color* was used in the list above just as a 'reminder' that a color is defined/associated with the given group.

### 3.2.3 Object Types

Using `\defobjectfmt` one can (re-)define custom `<obj-types>`. Similarly, there is a set of predefined ones, as follow:

<i>arg, meta</i>	based on (group) <i>meta</i>
<i>verb, xverb</i>	based on (group) <i>verb</i> plus <i>verb</i> or <i>xverb</i>
<i>marg</i>	based on (group) <i>meta</i> plus brackets
<i>oarg, parg, xarg</i>	based on (group) <i>oarg</i> plus brackets
<i>code, macro, function</i>	based on (group) <i>code</i>
<i>syntax</i>	based on (group) <i>syntax</i>
<i>keyval, key, keys, values</i>	based on (group) <i>keyval</i>
<i>option</i>	based on (group) <i>option</i>
<i>defaultval</i>	based on (group) <i>defaultval</i>
<i>env</i>	based on (group) <i>env</i>
<i>pkg, pack</i>	based on (group) <i>pkg</i>

### 3.2.4 Customization

One can add user defined groups/objects or change the pre-defined ones with the following commands:

#### `\defgroupfmt`

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 defined in 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

`<obj-type>` is the name of the new `<object>` being defined (or redefined), `<format-group>` is the base group to be used. `<format-keys>` allows 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  
*update:* 2023/05/01  
*update:* 2024/02/16  
*NB:* this is an example

---

This is the main environment to describe *Macros*, *Functions*, *Variable*, *Environments* and *etc.* `<csv-list>` is typeset in the margin. The optional `<obj-type>` (see 3.2 and 3.2.3) defines the object-type format.

**Note 1:** One can change the rule color with the key `rulecolor`, for instance `\begin{codedescribe}[rulecolor=white]` will remove the rules.

**Note 2:** Besides that, one can use the keys `new`, `update` and `note` to further customize it. (2024/02/16 these keys can also be used multiple times).

**Note 3:** Finally, one can use `EXP` and `rEXP` to add a star ★ or a hollow star ☆, as per expl3/doc3 conventions (if expandable, restricted expandable or not).

---

#### `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. Take note, as well, this is not a verbatim environment!

For example, the code for `codedescribe` (entry above) is:

LaTeX Code:

```
\begin{codedescribe}[env,new=2023/05/01,update=2023/05/01,note={this is an 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*`

---

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`

---

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 in the marginpar.

### 3.4 Commands

---

#### `\typesetobj`

#### `\tsobj`

---

This is the main typesetting command (most of the others are based on this). It can be used to typeset a single 'object' or a list thereof. In the case of a list, each term will be separated by commas. The last two by `sep` (defaults to: and).

**Note:** One can change the last 'separator' with the key `sep`, for instance `\tsobj [env,sep=or] {}` (in case one wants to produce an 'or' list of environments). Additionally, one can use the key `comma` to change the last separator to a single comma, like `\tsobj [env,comma] {}`.

`\typesetargs`  
`\tsargs`

Those will typeset `<csv-list>` as a list of parameters, like `[[<arg1>] [<arg2>] [<arg3>],` or `{<arg1>} {<arg2>} {<arg3>}`, etc. `<obj-type>` defines the formatting AND kind of brackets used (see 3.2): `[]` for optional arguments (`oarg`), `{}` for mandatory arguments (`marg`), and so on.

`\typesetmacro`  
`\tsmacro`

This is just a short-cut for  
`\tsobj[<code>]{<macro-list> \tsargs[<oarg>]{<oargs-list> \tsargs[<marg>]{<margs-list>}`.

`\typesetmeta`  
`\tsmeta`

Those will just typeset `<name>` between left/right 'angles' (no other formatting).

`\typesetverb`  
`\tsverb`

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 T<sub>E</sub>X errors may pop out.

`\typesetmarginnote`  
`\tsmarginnote`

Typesets a small note at the margin.

`tsremark`

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

L<sup>A</sup>T<sub>E</sub>X Code:

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

L<sup>A</sup>T<sub>E</sub>X Result:

---

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

---

### 3.5 Auxiliary Command / Environment

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

`typesettitle`  
`tstitle`

This is based on the `\maketitle` from the `article` class. The `<title-keys>` are:

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

`tsabstract`

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

`typesetdate`  
`tsdate`

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

new: 2023/05/16