

The codedescribe and codelisting Packages

Version 1.5b

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

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

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^AT_EX 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 1.5b and `codelisting` version 1.5b. Those two packages are fairly stable, and given the `\obj-type` mechanism (see below, 3.2) it can be easily extended without changing its 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^AT_EX code snippets in a `expl3` key.

```
codestore \begin{codestore} [\stcontents-keys]  
\end{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` `\setcodekeys` `{\code-keys}`

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

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^AT_EX Code:

```
\begin{codestore}[stmeta]
  Some \LaTeX~coding, for example: \ldots.
\end{codestore}
This will just typesets \tobj[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^AT_EX 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: ....
```

2.2.1 Code Keys

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

<i>settexcs</i>	<i>settexcs</i> , <i>settexcs2</i> and <i>settexcs3</i>
<i>texcs</i>	<i>texcs</i> , <i>texcs2</i> and <i>texcs3</i>
<i>texcsstyle</i>	<i>texcsstyle</i> , <i>texcs2style</i> and <i>texcs3style</i>

Those define sets of L^AT_EX 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).

<i>setkeywd</i>	<i>setkeywd</i> , <i>setkeywd2</i> and <i>setkeywd3</i>
<i>keywd</i>	<i>keywd</i> , <i>keywd2</i> and <i>keywd3</i>
<i>keywdstyle</i>	<i>keywdstyle</i> , <i>keywd2style</i> and <i>keywd3style</i>

Same for other *keywords* sets.

<i>setemph</i>	<i>setemph</i> , <i>setemph2</i> and <i>setemph3</i>
<i>emph</i>	<i>emph</i> , <i>emph2</i> and <i>emph3</i>
<i>emphstyle</i>	<i>emphstyle</i> , <i>emph2style</i> and <i>emph3style</i>

for some extra emphasis sets.

<i>numbers</i>	<i>numbers</i> and <i>numberstyle</i>
----------------	---------------------------------------

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.

<i>stringstyle</i>	<i>stringstyle</i> and <i>commentstyle</i>
--------------------	--

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

<code>bckgndcolor</code>	<code>bckgndcolor</code> to change the listing background's color.
<code>codeprefix</code> <code>resultprefix</code>	<code>codeprefix</code> and <code>resultprefix</code> those set the <code>codeprefix</code> (default: L ^A T _E X Code:) and <code>resultprefix</code> (default: L ^A T _E X Result:)
<code>parindent</code>	<code>parindent</code> Sets the indentation to be used when 'demonstrating' L ^A T _E X code (<code>\tsdemo</code>). Defaults to whatever value <code>\parindent</code> was when the package was first loaded.
<code>ruleht</code>	<code>ruleht</code> When typesetting the 'code demo' (<code>\tsdemo</code>) a set of rules is drawn. The Default, 1, equals to <code>\arrayrulewidth</code> (usually 0.4pt).
<code>basicstyle</code> <small>new: 2023/11/18</small>	<code>basicstyle</code> Sets the base font style used when typesetting the 'code demo', default being <code>\footnotesize\ttfamily</code>

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

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

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:

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

3.2.4 Customization

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

`\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 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` `\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. `<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 \begin{codedescribe} [(obj-type)] {<csv-list>}
...
\end{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 \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. 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 \begin{describelist} [(item-indent)] {<obj-type>}
describelist* ..\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.

3.4 Commands

`\typesetobj` `\typesetobj [<obj-type>] {<csv-list>}`
`\tsobj` `\tsobj [<obj-type>] {<csv-list>}`

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` `\typesetargs [<obj-type>] {<csv-list>}`
`\tsargs` `\tsargs [<obj-type>] {<csv-list>}`

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` `\typesetmacro {<macro-list>} [<oargs-list>] {<margs-list>}`
`\tsmacro` `\tsmacro {<macro-list>} [<oargs-list>] {<margs-list>}`

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

`\typesetmeta` `\typesetmeta {<name>}`
`\tsmeta` `\tsmeta {<name>}`

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

`\typesetverb` `\typesetverb [<obj-type>] {<verbatim text>}`
`\tsverb` `\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 T_EX errors may pop out.

`\typesetmarginnote` `\typesetmarginnote {<note>}`
`\tsmarginnote` `\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.

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

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` `\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 (Month Year, format).
