The codelist and codelisting Packages
Version 1.4

Alceu Frigeri*
February 2024

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 under-
lying 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 codelist Package ............................................... 4
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 \texttt{l3coffins}, \texttt{l3seq} and \texttt{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 \LaTeX{} code (using listings
and scontents), whilst codelist defines a series of macros to display/enumerate macros
and environments (somewhat resembling the doc3 style).

*https://github.com/alceu-frigeri/codelist
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 code\-describe version 1.4 and code\-listing version 1.4. Those two packages are fairly stable, and given the \langle\text{obj-type}\rangle mechanism (see below, 3.2) it can be easily extended without changing it’s interface.

2 code\-listing Package

It requires two packages: listings and scontents, defines an environment: codestore and 3 main commands: \texttt{tcode}, \texttt{tdemo} and \texttt{tsresult} and 1 auxiliary command: \texttt{set\text{codekeys}}.

2.1 In Memory Code Storage

Thanks to scontents (expl3 based) it’s possible to store \LaTeX\ code snippets in a expl3 key.

\begin{codestore} \[
\langle \text{stcontents-keys} \rangle
\end{codestore}

This environment is an alias to scontents environment (from scontents package), all scontents keys are valid, with two additional ones: \texttt{st} and \texttt{store\text{-}at} which are aliases to the store-env key. If an ‘isolated’ \texttt{st-name} is given (unknown \texttt{key}), it is assumed ‘by Default’ that the environment body shall be stored in it (for use with \texttt{tcode} and \texttt{tdemo}).

2.2 Code Display/Demo

\texttt{set\text{codekeys}} \langle \text{code-keys} \rangle

One has the option to set \langle\text{code-keys}\rangle per \texttt{tcode/tdemo} call, or \textit{globally}, better said, \textit{in the called context group}.

\textbf{N.B.:} All \texttt{tcode} and \texttt{tdemo} commands create a local group in which the \langle\text{code-keys}\rangle are defined, and discarded once said local group is closed. \texttt{set\text{codekeys}} defines those keys in the \textit{current context/group}.

\texttt{tcode*} \langle \text{code-keys} \rangle \langle \text{st-name} \rangle
\texttt{tdemo*} \langle \text{code-keys} \rangle \langle \text{st-name} \rangle
\texttt{tsresult*} \langle \text{code-keys} \rangle \langle \text{st-name} \rangle

update: 2024/01/06

\texttt{tcode} just typesets \langle\text{st-name}\rangle (the key-name created with \texttt{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.
\texttt{tdemo*} first typesets \langle\text{st-name}\rangle, as above, then it \textit{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) \texttt{tsresult*} only \textit{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:

\begin{codestore}[stmeta]
\begin{Verbatim}[codeprefix={Sample Code:}]
Some \LaTeX\ coding, for example: \ldots.
\end{Verbatim}
\end{codestore}

This will just typeset \texttt{stmeta}:

\begin{sdemo}[numbers=left,ruleht=0.5,\
codeprefix={inner sample code},\
resultprefix={inner sample result}]
\begin{Verbatim}[codeprefix={Sample Code:}]
Some \LaTeX\ coding, for example: \ldots.
\end{Verbatim}
\end{sdemo}

LATEX Result:

This will just typesets \texttt{stmeta}:

Sample Code:

Some \LaTeX\ coding, for example: \ldots.

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

\begin{sdemo}[numbers=left,ruleht=0.5,\
codeprefix={inner sample code},\
resultprefix={inner sample result}]
\begin{Verbatim}[codeprefix={Sample Code:}]
Some \LaTeX\ coding, for example: \ldots.
\end{Verbatim}
\end{sdemo}

2.2.1 Code Keys

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

\begin{tabular}{ll}
\texttt{settexcs} & \texttt{settexcs}, \texttt{settexcs2} and \texttt{settexcs3} \\
\texttt{texcs} & \texttt{texcs}, \texttt{texcs2} and \texttt{texcs3} \\
\texttt{texcsstyle} & \texttt{texcsstyle}, \texttt{texcs2style} and \texttt{texcs3style} \\
\end{tabular}

Those define sets of LATEX commands (csnames), the \texttt{set} variants initialize/redefine those sets (an empty list, clears the set), while the others extend those sets. The \texttt{style} ones redefine the command display style (an empty \texttt{value} resets the style to it’s default).

\begin{tabular}{ll}
\texttt{setkeywd} & \texttt{setkeywd}, \texttt{setkeywd2} and \texttt{setkeywd3} \\
\texttt{keywd} & \texttt{keywd}, \texttt{keywd2} and \texttt{keywd3} \\
\texttt{keywdstyle} & \texttt{keywdstyle}, \texttt{keywd2style} and \texttt{keywd3style} \\
\end{tabular}

Same for other keywords sets.

\begin{tabular}{ll}
\texttt{setemph} & \texttt{setemph}, \texttt{setemph2} and \texttt{setemph3} \\
\texttt{emph} & \texttt{emph}, \texttt{emph2} and \texttt{emph3} \\
\texttt{emphstyle} & \texttt{emphstyle}, \texttt{emph2style} and \texttt{emph3style} \\
\end{tabular}

for some extra emphasis sets.

\begin{tabular}{ll}
\texttt{numbers} & \texttt{numbers} and \texttt{numberstyle} \\
\texttt{numberstyle} & possible values are \texttt{none} (default) and \texttt{left} (to add tinny numbers to the left of the listing). With \texttt{numberstyle} one can redefine the numbering style.
\end{tabular}

\begin{tabular}{ll}
\texttt{stringstyle} & \texttt{stringstyle} and \texttt{commentstyle} \\
\texttt{codestyle} & to redefine \texttt{strings} and \texttt{comments} formatting style.
\end{tabular}
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 different package for \texttt{LATEX} code listing.

3.2 Object Type keys

The applied text format is defined in terms of \texttt{⟨obj-types⟩}, which are defined in terms of \texttt{⟨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 \texttt{⟨format-keys⟩} used to define \texttt{⟨format-groups⟩} and \texttt{⟨obj-types⟩}, which are:

- \texttt{meta} to typeset between angles,
- \texttt{xmeta} to typeset \texttt{verbatim} between angles,
- \texttt{verb} to typeset \texttt{verbatim*},
- \texttt{xverb} to typeset \texttt{verbatim*}, suppressing all spaces,
- \texttt{code} to typeset \texttt{verbatim*}, suppressing all spaces and replacing a TF by TF,
- \texttt{nofmt} in case of a redefinition, to remove the 'base' formatting,
- \texttt{slshape} to use a slanted font shape,
- \texttt{itshape} to use an italic font shape,
- \texttt{noShape} in case of a redefinition, to remove the 'base' shape,
- \texttt{lbracket} defines the left bracket (when using \texttt{\texttt{\LaTeX}sargs}). \textbf{Note:} this key must have an associated value,
rbracket defines the right bracket (when using `\targs`). 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:

- **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 `slshape` and `color`
- **env**: 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.

### 3.2.3 Object Types

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

- **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

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

```latex
\defgroupfmt \defobjectfmt
```

(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 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 {⟨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. (format-keys) allows for further differentiation.

For instance, the many optional (`*arg`) are defined as follow:
### 3.3 Environments

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] \end{codedescribe} 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).

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

\begin{verbatim}
\begin{codedescribe}[env,new=2023/05/01,update=2023/05/01,note={this is an example},update=2024/02/16]{codedescribe}
\begin{codesyntax}
\tmacro\{begin\{codedescribe\}\{obj-type\}\{csv-list\}\end{codesyntax}
\end{codedescribe}
\end{verbatim}

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, oth-
wise (item-name) will be typed in the marginpar.

3.4 Commands

\setobj \setobj [[obj-type]]{csv-list}
\setobj [[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 \setobj [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 \setobj [env,comma] {}.

\setargs \setargs [[obj-type]]{csv-list}
\setargs [[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.

\setmacro \setmacro {(macro-list)}{margs-list}
\setmacro {(macro-list)}{margs-list}
This is just a short-cut for
\setobj[code]macro-list \setargs[macro-list] \setargs[marg]margs-list.

\setmeta \setmeta {name}
\setmeta {name}
Those will just typeset (name) between left/right 'angles' (no other formatting).

\setverb \setverb [[obj-type]]{(verbatim text)}
\setverb [[obj-type]]{(verbatim text)}
Typesets (verbatim text) as is (verbatim...). (obj-type) defines the used format. The dif-
fERENCE with \setobj [verb](something) is that (verbatim text) can contain commas (which,
otherwise, would be interpreted as a list separator in \setobj).

Note: This is meant for short expressions, and not multi-line, complex code
(one is better off, then, using 2.2). (verbatim text) must be balanced! other-
wise, some low level \TeX errors may pop out.

\setmarginnote \setmarginnote {note}
\setmarginnote {note}
Typesets a small note at the margin.
The environment body will be typeset as a text note. \textit{NB} (defaults to Note:) is the note begin (in boldface). For instance:

\begin{tsremark}
\[\text{(NB)}\]
\end{tsremark}

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

### 3.5 Auxiliary Command / Environment

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

\texttt{typesettitle} \texttt{tstitle} \texttt{\textbackslash typesettitle \{(title-keys)\}} \texttt{\textbackslash tstitle \{(title-keys)\}}

This is based on the \texttt{\textbackslash maketitle} from the \texttt{article} class. The \langle title-keys \rangle are:

- \texttt{title} The used title.
- \texttt{author} Author’s name. It’s possible to use \texttt{\textbackslash footnote} command in it.
- \texttt{date} Title’s date.

\texttt{tsabstract} \texttt{\begin{tsabstract}\ldots\end{tsabstract}}

This is the \texttt{abstract} environment from the \texttt{article} class.

\texttt{typesetdate} \texttt{tsdate} \texttt{\textbackslash typesetdate} \texttt{\textbackslash tsdate}

new: 2023/05/16

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