The codedescribe and codelisting Packages
Version 1.0

Alceu Frigeri[*]

2023/05/11

Abstract

This documentation package is designed to be ‘as class independent as possible’, depending only on `expl3`, `scontents` and `listing`. 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

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.

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).

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`.

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

1.2 Current Version

This doc regards to *codedescribe* version 1.0 and *codelisting* version 1.0. 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 2 main commands: `\tscode` and `\tsdemo` and 1 auxiliary command: `\setcodekeys`.

2.1 In Memory Code Storage

Thanks to *scontents* (*expl3* based) it's possible to store LaTeX 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 setting ⟨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*`

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

`\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.

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: . . . .

2.2.1 Code Keys

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

settexcs
texcs
texcsstyle

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

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

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

Same for other *keywords* sets.

setemph
emph
emphstyle

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

for some extra emphasis sets.

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 2_εcode (`\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 is drawn. The Default, 1, equals to `\arrayrulewidth` (usually 0.4pt).

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 LaTeX code listing.

3.2 ⟨obj-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 a ⟨format-group⟩ / ⟨obj-type⟩, which are:

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 TF,
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`),
rbracket	defines the right bracket (when using `\tsargs`),
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⟩

Based on the *format-keys* the following *format-groups* are pre-defined:

meta	which sets *meta* and *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.

3.2.3 ⟨obj-types⟩

Finally, based on those *format-groups* the *obj-type* keys are pre-defined:

arg, meta	based on (group) *meta*
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:

`\defgroupfmt`
new: 2023/05/17

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

⟨format-group⟩ is the name of the new group (or 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* 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*).

`\defobjectfmt`
new: 2023/05/17

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

⟨obj-type⟩ is the name of the new ⟨object⟩ being defined (or redefined, in case of one already known/defined), ⟨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: 2023/05/1
NB: this is an example

```
\begin{codedescribe} [⟨obj-type⟩] {⟨csv-list⟩}
...
\end{codedescribe}
```

This is the main environment to describe *Macros*, *Functions*, *Variable*, *Environments* and *etc.* ⟨csv-list⟩ is typeset in the margin. The optional ⟨obj-type⟩ defines the object-type format.

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

Note: Besides that, one can use the keys *`new`*, *`update`* and *`note`* to further customize it as: `\begin {codedescribe}[new=2023/05/01,update=2023/05/1,note={this is an example}]`

`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=2023/05/1,note={this is an example}]{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.

3.4 Commands

`\typesetobj`
`\tsobj`

```
\typesetobj [⟨obj-type⟩] {⟨csv-list⟩}
\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`
`\tsargs`

```
\typesetargs [⟨obj-type⟩] {⟨csv-list⟩}
\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 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⟩}`

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

\typesetmeta
\tsmeta

`\typesetmeta {⟨name⟩}`
`\tsmeta {⟨name⟩}`

Those 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.

\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.

3.5 Auxiliar 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

`\typesettitle {⟨title-keys⟩}`
`\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.