

# `coder` — code inlined in a $\text{\LaTeX}$ document<sup>\*</sup>

Jérôme LAURENS<sup>†</sup>

Released 2022/02/07

## Abstract

Usually, documentation is put inside the code, `coder` allows to work the other way round by putting code inside the documentation. This is particularly interesting when different code files share some logic and should be documented all at once. The file `coder-manual` gives different examples. Here is the implementation of the package.

This  $\text{\LaTeX}$  package requires  $\text{Lua}\text{\TeX}$  and may use syntax coloring based on `pygments`.

## 1 Package dependencies

`luacode`, `datetime2`, `xcolor`, `fancyvrb` and dependencies of these packages.

## 2 Similar technologies

The `docstrip` utility offers similar features, it is somehow more powerful than `coder` at the cost of more technicality and less practicality,

The `ydoc.cls` and `skdoc.cls` are full document classes with similar features but many more that are unrelated. `coder` focuses on code inlining and interfaces very well with `pygments` for a smart syntax highlighting.

## 3 Known bugs and limitations

- `coder` does not play well with `docstrip`.

## 4 Namespace and conventions

$\text{\LaTeX}$  identifiers related to `coder` start with `CDR`, including both commands and environments. `expl3` identifiers also start with `CDR`, after and eventual leading `c_`, `l_` or `g_`. `l3keys` module path's first component is either `CDR` or starts with `CDR@`.

`lua` objects (functions and variables) are collected in the `CDR` table automatically created while loading `coder-util.lua` from `coder.sty`.

---

<sup>\*</sup>This file describes version 2022/02/07, last revised 2022/02/07.

<sup>†</sup>E-mail: [jerome.laurens@u-bourgogne.fr](mailto:jerome.laurens@u-bourgogne.fr)

The `c` argument specifier is used here in a more general acception. Normally, it means that the argument is turned to a command sequence name. Here, it means that the argument is a part of something bigger which is turned to a command sequence name.

## 5 Presentation

`coder` is a triptych of three complementary components

1. `coder.sty`, on the  $\text{\LaTeX}$  side,
2. `coder-util.lua`, to store data and call `coder-tool.py`,
3. `coder-tool.py`, to color code with the help of `pygment`.

`coder.sty` mainly declares the `\CDRCode` command and the `CDRBlock` environment. The former allows to insert code chunks as running text whereas the latter allows to insert code snippets as blocks. Moreover, both code chunks can be exported to files, once declared with `\CDRExport` command. The `\CDRSet` command is used to set various parameters, including display engines declared with either `\CDRNewCodeEngine` or `\CDRNewBlockEngine`.

### 5.1 Code flow

The normal code flow is

1. from `coder.sty`,  $\text{\LaTeX}$  parses a code snippet as `\CDRCode` argument of `CDRBlock` environment body, somehow stores it, and calls either `CDR:highlight_code` or `CDR:highlight_block`,
2. `coder-util.lua` reads the content of some command, and store it in a `json` file, together with informations to process this code snippet properly,
3. `coder-tool.py` is asked by `coder-util.lua` to read the `json` file and eventually uses `pygments` to translate the code snippet into dedicated  $\text{\LaTeX}$  coloring commands. These are stored in a `*.pyg.tex` file named after the md5 digest of the original code chunk, a `*.pyg.sty`  $\text{\LaTeX}$  style file is recorded as well. On return, `coder-tool.py` gives to `coder-util.lua` some  $\text{\LaTeX}$  instructions to both input the `*.pyg.sty` and the `*.pyg.tex` file, these are finally executed and the code is displayed with colors. `coder-tool.py` is also partially responsible of code line numbering.

`coder.sty` only exchanges with `coder.sty` using `\directlua` and `tex.print`. `coder-tool.py` in turn only exchanges with `coder.sty`: we put in `coder-tool.py` as few  $\text{\LaTeX}$  logic as possible. It receives instructions from `coder.sty` as command line arguments, options, `pygments` options and `fancyvrb` options.

### 5.2 File exports

1. The `\CDRExport` command declares a file path, a list of tags and other useful information like a coding language. These data are saved as export records by `coder-util.lua`.
2. When some `tags={...}` have been given to the `CDRBlock` environment, the `coder-util.lua` records the corresponding code chunk and its associate tags for later save.

3. Once the typesetting process is complete, `coder-util.lua`'s `CDR_export_...` methods are called to save all the files externally. For each export record, `coder-util.lua` collects all the chunks with the same tag and save them at the proper location.

### 5.3 Display engine

The display management is partly delegated to other packages. `coder.sty` provides default engines for running code and code blocks, and new engines can be declared with `\CDRNewCodeEngine` and `\CDRNewBlockEngine`.

### 5.4 L<sup>A</sup>T<sub>E</sub>X user interface

The first required argument of both commands and environment is a `<key[=value] controls>` list managed by `l3keys`. Each command requires its own `l3keys` module but some `<key[=value] controls>` are shared between modules.

### 5.5 Properties and inheritance

Properties cover various informations, from the language of the code, to the color and font. They are uniquely identified by a path, and may be defined at the *global* level or at the *tag* level.

**the *global* level** is set by `\CDRSet` and `\CDRExport`, it consists of global variables,

**the *tag* level** is set by `\CDRSet`, `\CDRCode` and `CDRBlock` environment.

Each processed code chunk has a list of associate tags. Most tag inherits from default ones.

## 6 Options

Key-value options allow the user, `coder.sty`, `coder-util.lua` and `CDRPy` to exchange data. What the user is allowed to do is detailed in [coder-manual.pdf](#).

### 6.1 fancyvrb

These are `fancyvrb` options verbatim. The `fancyvrb` manual has more details, only some parts are reproduced hereafter. All of these options may not be relevant for all situations. Some of them make no sense in `code` mode, whereas others may not be compatible with the display engine.

- **formatcom**=`<command>` execute before printing verbatim text. Initially empty. Ignored in `code` mode.
- **fontfamily**=`<family name>` font family to use. `tt`, `courier` and `helvetica` are pre-defined. Initially `tt`.
- **fontsize**=`<font size>` size of the font to use. If you use the `relsize` package as well, you can require a change of the size proportional to the current one (for instance: `fontsize=\relsize{-2}`). Initially `auto`: the same as the current font.

- **fontshape**=*<font shape>* font shape to use. Initially **auto**: the same as the current font.
- **showspaces**[=**true**|**false**] print a special character representing each space. Initially **false**: spaces not shown.
- **showtabs**=**true**|**false** explicitly show tab characters. Initially **false**: tab characters not shown.
- **obeytabs**=**true**|**false** position characters according to the tabs. Initially **false**: tab characters are added to the current position.
- **tabsize**=*<integer>* number of spaces given by a tab character, Initially 2 (8 for **fancyvrb**).
- **defineactive**=*<macro>* to define the effect of active characters. This allows to do some devious tricks, see the **fancyvrb** package. Initially empty.
- ✓ **relabel**=*<label>* define a label to be used with **\pageref**. Initially empty.
- **commentchar**=*<character>* lines starting with this character are ignored. Initially empty.
- **gobble**=*<integer>* number of characters to suppress at the beginning of each line (from 0 to 9), mainly useful when environments are indented. Only **block** mode.
- **frame**=**none**|**leftline**|**topline**|**bottomline**|**lines**|**single** type of frame around the verbatim environment. With **leftline** and **single** modes, a space of a length given by the L<sup>A</sup>T<sub>E</sub>X **\fboxsep** macro is added between the left vertical line and the text. Initially **none**: no frame.
- **label**={ [*<top string>*] *<string>* } label(s) to print on top, bottom or both, frame lines. If the label(s) contains special characters, comma or equal sign, it must be placed inside a group. If an optional *<top string>* is given between square brackets, it will be used for the top line and *<string>* for the bottom line. Otherwise, *<string>* is used for both the top or bottom lines. Label(s) are printed only if the **frame** parameter is one of **topline**, **bottomline**, **lines** or **single**. Initially empty: no label.
- **labelposition**=**none**|**topline**|**bottomline**|**all** position where to print the label(s) when defined. When options happen to be contradictory, like **frame=topline** and **labelposition=bottomline**, nothing is displayed. Initially **none** when no labels are defined, **topline** for one label and **all** otherwise.
- **numbers**=**none**|**left**|**right** numbering of the verbatim lines. If requested, this numbering is done outside the verbatim environment. Initially **none**: no numbering.
- **numbersep**=*<dimension>* gap between numbers and verbatim lines. Initially 12pt.
- **firstnumber**=**auto**|**last**|*<integer>* number of the first line. **last** means that the numbering is continued from the previous verbatim environment. If an integer is given, its value will be used to start the numbering. Initially **auto**: numbering starts from 1.

- **stepnumber**=*<integer>* interval at which line numbers are printed. Initially 1: all lines are numbered.
- **numberblanklines**[=*true|false*] to number or not the white lines (really empty or containing blank characters only). Initially **true**: all lines are numbered.
- **firstline**=*<integer>* first line to print. Initially empty: all lines from the first are printed.
- **lastline**=*<integer>* last line to print. Initially empty: all lines until the last one are printed.
- **baselinestretch**=*auto|<dimension>* value to give to the usual `\baselinestretch`  $\text{\LaTeX}$  parameter. Initially **auto**: its current value just before the verbatim command.
- ⊘ **commandchars**=*<three characters>* characters which define the character which starts a macro and marks the beginning and end of a group; thus lets us introduce escape sequences in verbatim code. Of course, it is better to choose special characters which are not used in the verbatim text. Private to **coder**, unavailable to users.
- **xleftmargin**=*<dimension>* indentation to add at the start of each line. Initially **0pt**: no left margin.
- **xrightmargin**=*<dimension>* right margin to add after each line. Initially **0pt**: no right margin.
- **resetmargins**[=*true|false*] reset the left margin, which is useful if we are inside other indented environments. Initially **true**.
- **hfuzz**=*<dimension>* value to give to the  $\text{\TeX}$  `\hfuzz` dimension for text to format. This can be used to avoid seeing some unimportant overfull box messages. Initially **2pt**.
- **samepage**[=*true|false*] in very special circumstances, we may want to make sure that a verbatim environment is not broken, even if it does not fit on the current page. To avoid a page break, we can set the **samepage** parameter to **true**. Initially **false**.

## 6.2 pygments options

These are **pygments**'s `LatexFormatter` options, used only by `coder-util.lua` to communicate with `coder-tool.py`.

- **style**=*<name>* the **pygments** style to use. Initially **default**.
- ⊘ **full** Tells the formatter to output a **full** document, i.e. a complete self-contained document (default: **false**). Forbidden.
- ⊘ **title** If **full** is true, the title that should be used to caption the document (default empty). Forbidden.
- ⊘ **encoding** If given, must be an encoding name. This will be used to convert the Unicode token strings to byte strings in the output. If it is `or None`, Unicode strings will be written to the output file, which most file-like objects do not support (default: `None`).

- ⊘ **outencoding** Overrides `encoding` if given.
- ⊘ **docclass** If the `full` option is enabled, this is the document class to use (default: `article`). Forbidden.
- ⊘ **preamble** If the `full` option is enabled, this can be further preamble commands, e.g. `"\usepackage"` (default `empty`). Forbidden.
- ⊘ **linenos**[=`true`|`false`] If set to `true`, output line numbers. Initially `false`: no numbering. Ignored in `code` mode.
- ⊘ **linenostart**=`<integer>` The line number for the first line. Initially 1: numbering starts from 1. Ignored in `code` mode.
- ⊘ **linenostep**=`<integer>` If set to a number  $n > 1$ , only every  $n$ th line number is printed. Ignored in `code` mode. Additional options given to the `Verbatim` environment (see the `fancyvrb` docs for possible values). Initially `empty`.
- ⊘ **verboptions** Forbidden.
- **commandprefix**=`<text>` The LaTeX commands used to produce colored output are constructed using this prefix and some letters. Initially `PY`.
- **texcomments**[=`true`|`false`] If set to `true`, enables LaTeX comment lines. That is, LaTeX markup in comment tokens is not escaped so that LaTeX can render it. Initially `false`. Ignored in `code` mode.
- **mathescape**[=`true`|`false`] If set to `true`, enables LaTeX math mode escape in comments. That is, `$...$` inside a comment will trigger math mode. Initially `false`.
- **escapeinside**=`<before>``<after>` If set to a string of length 2, enables escaping to LaTeX. Text delimited by these 2 characters is read as LaTeX code and typeset accordingly. It has no effect in string literals. It has no effect in comments if `texcomments` or `mathescape` is set. Initially `empty`.
- ⚙ **envname**=`<name>` Allows you to pick an alternative environment name replacing `Verbatim`. The alternate environment still has to support `Verbatim`'s option syntax. Initially `Verbatim`.

### 6.3 LaTeX

These are options used by `coder.sty` to pass data to `coder-tool.py`. All values are required, possibly empty.

- **tags** `clist` of tag names, used for line numbering.
- **inline** `true` when inline code is concerned, `false` otherwise.
- **already\_style** `true` when the style has already been defined, `false` otherwise,
- **sty\_template** LaTeX source text where `<placeholder:style_defs>` must be replaced by the style definitions provided by `pygments`. It may include the style name.
- **code\_template** LaTeX source text where `<placeholder:highlighted>` should be replaced by the highlighted code provided by `pygments`.

- **block\_template** L<sup>A</sup>T<sub>E</sub>X source text where `<placeholder:count>` should be replaced by the count of numbered lines (not all lines may be numbered) and `<placeholder:highlighted>` should be replaced by the highlighted code provided by `pygments`.

All the line templates below are L<sup>A</sup>T<sub>E</sub>X source text where `<placeholder:number>` should be replaced by a line number and `<placeholder:line>` should be replaced by the highlighted line code provided by `pygments`. They should not include a trailing newline char.

- **single\_line\_template** It may contain tag related information and number as well. When the block consists of only one line.
- **first\_line\_template** When the block consists of more than one line. If the tag information is required or new, display only the tag. Display the number if required, otherwise.
- **second\_line\_template** If the first line did not, display the line number, but only when required.
- **black\_line\_template** for numbered lines,
- **white\_line\_template** for unnumbered lines,

## File I

# coder-util.lua implementation

## 1 Usage

This lua library is loaded by `coder.sty` with the instruction `CDR=require(coder-util)`. In the sequel, the syntax to call class methods and instance methods are presented with either a `CDR.` or a `CDR:` prefix. This is what is used in the library for convenience. Of course either a `self.` or a `self:` prefix would be possible.

## 2 Declarations

```
1 %<*lua>
2 local lfs    = _ENV.lfs
3 local tex    = _ENV.tex
4 local token  = _ENV.token
5 local rep    = string.rep
6 local lpeg   = require("lpeg")
7 local P, Cg, Cp, V = lpeg.P, lpeg.Cg, lpeg.Cp, lpeg.V
8 require("lualibs.lua")
9 local json   = _ENV.utilities.json
```

## 3 General purpose material

`CDR_PY_PATH` Location of the `coder-tool.py` utility. This will cause an error if `kpsewhich` is not available. The PATH must be properly set up.

```

10 local CDR_PY_PATH = io.popen(
11   [[kpsewhich coder-tool.py]]
12 ):read('a'):match("^%s*(.)%s*$")


```

*(End definition for CDR\_PY\_PATH. This variable is documented on page ??.)*

---

**escape**     $\langle \text{variable} \rangle = \text{CDR.escape}(\langle \text{string} \rangle)$

---

 Escape the given string. NEVER USED.

```

13 local function escape(s)
14   s = s:gsub('\\', '\\\\')
15   s = s:gsub('\r', '\\r')
16   s = s:gsub('\n', '\\n')
17   s = s:gsub('"', '\\\"')
18   return s
19 end

```

---

**make\_directory**     $\langle \text{variable} \rangle = \text{CDR.make\_directory}(\langle \text{string path} \rangle)$

---

Make a directory at the given path.

```

20 local function make_directory(path)
21   local mode,_,__ = lfs.attributes(path,"mode")
22   if mode == "directory" then
23     return true
24   elseif mode ~= nil then
25     return nil,path.." exist and is not a directory",1
26   end
27   if os["type"] == "windows" then
28     path = path:gsub("/", "\\")
29     __,__,__ = os.execute(
30       "if not exist " .. path .. "\\nul " .. "mkdir " .. path
31     )
32   else
33     __,__,__ = os.execute("mkdir -p " .. path)
34   end
35   mode = lfs.attributes(path,"mode")
36   if mode == "directory" then
37     return true
38   end
39   return nil,path.." exist and is not a directory",1
40 end

```

**dir\_p**    The directory where the auxiliary pygments related files are saved, in general  $\langle \text{jobname} \rangle.\text{pygd}/$ .

*(End definition for dir\_p. This variable is documented on page ??.)*

**json\_p**    The path of the JSON file used to communicate with coder-tool.py, in general  $\langle \text{jobname} \rangle.\text{pygd}/\langle \text{jobname} \rangle$

*(End definition for json\_p. This variable is documented on page ??.)*



```

41 local dir_p, json_p
42 local jobname = tex.jobname
43 dir_p = './'..jobname..'pygd/'
44 if make_directory(dir_p) == nil then
45   dir_p = './'
46   json_p = dir_p..jobname..'pyg.json'
47 else
48   json_p = dir_p..'input.pyg.json'
49 end

```

---

**print\_file\_content**    CDR.print\_file\_content(*<macro name>*)

---

The command named *<macro name>* contains the path to a file. Read the content of that file and print the result to the T<sub>E</sub>X stream.

```

50 local function print_file_content(name)
51   local p = token.get_macro(name)
52   local fh = assert(io.open(p, 'r'))
53   s = fh:read('a')
54   fh:close()
55   tex.print(s)
56 end

```

---

**load\_exec**    CDR.load\_exec(*<lua code chunk>*)

---

Class method. Loads the given *<lua code chunk>* and execute it. On error, messages are printed.

```

57 local function load_exec(chunk)
58   local func, err = load(chunk)
59   if func then
60     local ok, err = pcall(func)
61     if not ok then
62       print("coder-util.lua Execution error:", err)
63       print('chunk:', chunk)
64     end
65   else
66     print("coder-util.lua Compilation error:", err)
67     print('chunk:', chunk)
68   end
69 end

```

---

**safe\_equals**    *<variable>* = safe\_equals(*<string>*)

---

Class method. Returns an *<=...=>* string as *<ans>* exactly composed of sufficiently many = signs such that *<string>* contains neither sequence [*<ans>*] nor [*<ans>*].

```

70 local eq_pattern = P({ Cp() * P('=')^1 * Cp() + 1 * V(1) })
71 local function safe_equals(s)
72   local i, j = 0, 0
73   local max = 0
74   while true do

```

```

75     i, j = eq_pattern:match(s, j)
76     if i == nil then
77         return rep("'", max + 1)
78     end
79     i = j - i
80     if i > max then
81         max = i
82     end
83 end
84 end

```

---

**load\_exec\_output**    CDR:load\_exec\_output(*lua code chunk*)

---

Instance method to parse the *lua code chunk* string for commands and execute them. The patterns being searched are enclosed within opening <<<<< and closing >>>>>, each containing 5 characters,

?TEX:<TeX instructions> the <TeX instructions> are executed asynchronously once the control comes back to T<sub>E</sub>X.

!LUA:<!Lua instructions> the <!Lua instructions> are executed synchronously. When not properly designed, these instruction may cause a forever loop on execution, for example, they must not use CDR:highlight\_code.

?LUA:<?Lua instructions> these <?Lua instructions> are executed asynchronously once the control comes back to T<sub>E</sub>X through a call to \directlua, which means that they will wait until any previous asynchronous <?TeX instructions> or <?Lua instructions> completes.

```

85 local parse_pattern
86 do
87     local tag = P('!'') + '?'
88     local stp = '>>>>>'
89     local cmd = P(1)^0 - stp
90     parse_pattern = P({
91         '<<<<<' * Cg(tag - 'LUA:') * ':' * Cg(cmd) * stp * Cp() + 1 * V(1)
92     })
93 end
94 local function load_exec_output(self, s)
95     local i, tag, cmd
96     i = 0
97     while true do
98         tag, cmd, i = parse_pattern:match(s, i)
99         if tag == '!' then
100             self.load_exec(cmd)
101         elseif tag == '?' then
102             local eqs = self.safe_equals(cmd)
103             cmd = '[' .. eqs .. '[' .. cmd .. ']' .. eqs .. ']'
104             tex.print([[
105 \directlua{CDR:load_exec([]) .. cmd .. []}]%
106 ]])
107         else
108             return

```

```

109     end
110   end
111 end

```

## 4 Properties

This is one of the channels from `coder.sty` to `coder-util.lua`.

---

**options\_reset**    `CDR:options_reset()`

Instance method. This is called by `coder.sty`'s `\CDR_to_lua`.

```

112 local function options_reset(self)
113   self['.options'] = {}
114 end

```

---

**option\_add**    `CDR:option_add(<key>, <value name>)`

Instance method. This is called by `coder.sty`'s `\CDR_to_lua`.

```

115 local function option_add(self, key, value_name)
116   local p = self['.options']
117   p[key] = token.get_macro(assert(value_name))
118 end

```

## 5 Hiligting

### 5.1 Code

---

**highlight\_code**    `CDR:highlight_code(<code var>)`

Highlight the code in `str` variable named `<code var name>`. Build a configuration table with all data necessary for the processing, save it as a JSON file and launch `coder-tool.py` with the proper arguments.

```

119 local function highlight_code(self, code_name)
120   local code = assert(tex.token(assert(code_name)))
121   local config = {
122     ['__cls__'] = 'Arguments'
123   }
124   local texopts = {
125     ['__cls__'] = 'TeXOpts'
126   }
127   texopts.sty_template = [[
128 \makeatletter
129 \CDR@StyleDefine {}..pygopts.style..[] {%
130 <placeholder:style_defs>%
131 }%
132 \makeatother
133 ]]
134   texopts.white_line_template = [[<placeholder:line>]]

```

```

135 texopts.black_line_template = [[
136   \CDR@Number{<placeholder:number>><placeholder:line>}]
137 texopts.single_line_template = [[\CDR@Number{<placeholder:number>><placeholder:line>}]
138 texopts.first_line_template = [[<placeholder:line>]]
139 texopts.second_line_template = [[<placeholder:line>]]
140 config['texopts'] = texopts
141 local fv_opts = {
142   ['__cls__'] = 'FVOpts'
143 }
144 config['fv_opts'] = fv_opts
145 local pyg_opts = {
146   ['__cls__'] = 'PygOpts'
147 }
148 config['pyg_opts'] = pyg_opts
149
150 end

```

## 5.2 Block

---

**process\_block\_new** CDR:process\_block\_new(*<tags clist>*)

---

Records the *<tags clist>* to prepare block highlighting.

```

151 local function process_block_new(self, tags_clist)
152   local t = {}
153   for tag in string.gmatch(tags_clist, '([^\,]+)') do
154     t[#t+1]=tag
155   end
156   self['block tags'] = tags_clist
157   self['.lines'] = {}
158 end

```

---

**process\_line** CDR:process\_line(*<line variable name>*)

---

Store the content of the given named variable.

```

159 local function process_line(self, line_variable_name)
160   local line = assert(tex.token(assert(line_variable_name)))
161   local ll = self['.lines']
162   ll[#ll+1] = line
163   local lt = self['lines by tag'] or {}
164   self['lines by tag'] = lt
165   for tag in self['block tags']:gmatch('([^\,]+)') do
166     ll = lt[tag] or {}
167     lt[tag] = ll
168     ll[#ll+1] = line
169   end
170 end

```

---

**highlight\_code** CDR:highlight\_block(*<block var name>*)

---

Highlight the code in *str* variable named *<block var name>*. Build a configuration table with all data necessary for the processing, save it as a JSON file and launch `coder-tool.py` with the proper arguments.

```

171 local function hilight_block(self, block_name)
172 end

```

## 6 Exportation

For each file to be exported, `coder.sty` calls `export_file` to initialte the exportation. Then it calls `export_file_info` to share the `tags`, `raw`, `preamble`, `postamble` data. Finally, `export_complete` is called to complete the exportation.

---

**export\_file**     CDR:export\_file(*<file name var>*)

This is called at export time. *<file name var>* is the name of an `str` variable containing the file name.

```

173 local function export_file(self, file_name)
174   self['.name'] = assert(tex.token(assert(file_name)))
175   self['.export'] = {}
176 end

```

---

**export\_file\_info**     CDR:export\_file\_info(*<key>*, *<value name var>*)

This is called at export time. *<value name var>* is the name of an `str` variable containing the value.

```

177 local function export_file_info(self, key, value)
178   local export = self['.export']
179   value = assert(token.get_macro(assert(value)))
180   export[key] = value
181 end

```

---

**export\_complete**     CDR:export\_complete()

This is called at export time.

```

182 local function export_complete(self)
183   local name = self['.name']
184   local export = self['.export']
185   local records = self['.records']
186   local tt = {}
187   local s = export.preamble
188   if s then
189     tt[#tt+1] = s
190   end
191   for _,tag in ipairs(export.tags) do
192     s = records[tag]:concat('\n')
193     tt[#tt+1] = s
194     records[tag] = { [1] = s }
195   end
196   s = export.postamble
197   if s then
198     tt[#tt+1] = s
199   end

```

```

200 if #tt>0 then
201   local fh = assert(io.open(name,'w'))
202   fh:write(tt:concat('\n'))
203   fh:close()
204 end
205 self['.file'] = nil
206 self['.exportation'] = nil
207 end

```

## 7 Caching

We save some computation time by pygmentizing files only when necessary. The `codertool.py` is expected to create a `*.pyg.sty` file for a style and a `*.pyg.tex` file for highlighted code. These files are cached during one whole L<sup>A</sup>T<sub>E</sub>X run and possibly between different L<sup>A</sup>T<sub>E</sub>X runs. Lua keeps track of both the style files created and highlighted code files created.

<code>cache_clean_all</code>	<code>CDR:cache_clean_all()</code>
<code>cache_record</code>	<code>CDR:cache_record(<i>&lt;style name.pyg.sty&gt;</i>, <i>&lt;digest.pyg.tex&gt;</i>)</code>
<code>cache_clean_unused</code>	<code>CDR:cache_clean_unused()</code>

Instance methods. `cache_clean_all` removes any file in the cache directory named `<jobname>.pygd`. This is automatically executed at the beginning of the document processing when there is no aux file. This can also be executed on demand with `\directlua{CDR:cache_clean_all()}`. The `cache_record` method stores both `<style name.pyg.sty>` and `<digest.pyg.tex>`. These are file names relative to the `<jobname>.pygd` directory. `cache_clean_unused` removes any file in the cache directory `<jobname>.pygd` except the ones that were previously recorded. This is executed at the end of the document processing.

```

208 local function cache_clean_all(self)
209   local to_remove = {}
210   for f in lfs.dir(dir_p) do
211     to_remove[f] = true
212   end
213   for k,_ in pairs(to_remove) do
214     os.remove(dir_p .. k)
215   end
216 end
217 local function cache_record(self, style, colored)
218   self['.style_set'][style] = true
219   self['.colored_set'][colored] = true
220 end
221 local function cache_clean_unused(self)
222   local to_remove = {}
223   for f in lfs.dir(dir_p) do
224     if not self['.style_set'][f] and not self['.colored_set'][f] then
225       to_remove[f] = true
226     end
227   end
228   for k,_ in pairs(to_remove) do
229     os.remove(dir_p .. k)
230   end
231 end

```

`_DESCRIPTION` Short text description of the module.

```
232 local _DESCRIPTION = [[Global coder utilities on the lua side]]  
  
    (End definition for _DESCRIPTION. This variable is documented on page ??.)
```

## 8 Return the module

```
233 return {  
  
    Known fields are  
  
234     _DESCRIPTION      = _DESCRIPTION,  
  
    _VERSION to store <version string>,  
  
235     _VERSION          = token.get_macro('fileversion'),  
  
    date to store <date string>,  
  
236     date              = token.get_macro('filedate'),  
  
237     CDR_PY_PATH       = CDR_PY_PATH,  
  
    escape  
  
238     escape            = escape,  
  
    make__directory  
  
239     make_directory    = make_directory,  
  
    load__exec  
  
240     load_exec         = load_exec,  
  
241     load_exec_output  = load_exec_output,  
  
    record__line  
  
242     record_line       = function(self,line) end,  
  
    hilight__code  
  
243     hilight_code      = hilight_code,  
  
    process__block__new, hilight__block  
  
244     process_block_new = process_block_new,  
245     hilight_block     = hilight_block,  
  
    cache__clean__all
```

```

246  cache_clean_all    = cache_clean_all,

    cache__record

247  cache_record       = cache_record,

    cache__clean__unused

248  cache_clean_unused = cache_clean_unused,

249  options_reset      = options_reset,

250  option_add          = option_add,

```

### Internals

```

251  ['.style_set']      = {},
252  ['.colored_set']    = {},
253  ['.options']        = {},
254  ['.export']         = {},
255  ['.name']           = nil,

```

`already` false at the beginning, true after the first call of `coder-tool.py`

```

256  already             = false,

257  }

258  %</lua>

```

## File II

# coder-tool.py implementation

The standard header is managed specially because of the way `docstrip` automatically adds some header when extracting stuff from an archive. The next two lines are added by `docstrip` at the top of the preamble.

```

1  %<*py>
2  #! /usr/bin/env python3
3  # -*- coding: utf-8 -*-
4  %</py>

```

## 1 Usage

Run: `coder-tool.py -h`.



## 2 Header and global declarations

```
5 %<*py>
6 __version__ = '0.10'
7 __YEAR__ = '2022'
8 __docformat__ = 'restructuredtext'
9
10 from posixpath import split
11 import sys
12 import argparse
13 import re
14 from pathlib import Path
15 import hashlib
16 import json
17 from pygments import highlight as hilight
18 from pygments.formatters.latex import LatexEmbeddedLexer, LatexFormatter
19 from pygments.lexers import get_lexer_by_name
20 from pygments.util import ClassNotFound
21 from pygments.util import guess_decode
```

## 3 Options classes

Object is used to turn a dictionary into a full fledged object. The real class is given by the `__cls__` key.

```
22 class Options(object):
23     @staticmethod
24     def ensure_bool(x):
25         if x == True or x == False: return x
26         x = x[0:1]
27         return x == 'T' or x == 't'
28
29     def __new__(cls, d={}, *args, **kwargs):
30         __cls__ = d.get('__cls__', 'arguments')
31         if __cls__ == 'PygOpts':
32             return super(Options, cls).__new__(
33                 Controller.PygOpts, *args, **kwargs
34             )
35         elif __cls__ == 'FVOpts':
36             return super(Options, cls).__new__(
37                 Controller.FVOpts, *args, **kwargs
38             )
39         elif __cls__ == 'TeXOpts':
40             return super(Options, cls).__new__(
41                 Controller.TexOpts, *args, **kwargs
42             )
43         else:
44             return super(Options, cls).__new__(
45                 Controller.Arguments, *args, **kwargs
46             )
47     def __init__(self, d={}):
48         for k, v in d.items():
49             if type(v) == str:
```

```

49         if v.lower() == 'true':
50             setattr(self, k, True)
51             continue
52         elif v.lower() == 'false':
53             setattr(self, k, False)
54             continue
55         setattr(self, k, v)
56     def __repr__(self):
57         return f"{object['__repr__'](self)}: {self['__dict__']}"

```

### 3.1 TeXOpts nested class

```

58 class TeXOpts(Options):
59     tags = ''
60     inline = True
61     already_style = False

```

The templates are provided by `coder.sty`. The style template wraps the style definitions provided by `pygments`. It may include the style name

```

62     sty_template='<placeholder:style_defs>'
63     code_template = '<placeholder:hilighted>'
64     single_line_template='<placeholder:number><placeholder:line>'
65     first_line_template='<placeholder:number><placeholder:line>'
66     second_line_template='<placeholder:number><placeholder:line>'
67     white_line_template='<placeholder:number><placeholder:line>'
68     black_line_template='<placeholder:number><placeholder:line>'
69     block_template='<placeholder:count><placeholder:hilighted>'
70     def __init__(self, *args, **kwargs):
71         super().__init__(*args, **kwargs)
72         self.inline = self.ensure_bool(self.inline)

```

### 3.2 PygOpts nested class

`pygments LaTeXFormatter` options. Some of them may be deliberately unused. In particular, line numbering is governed by `fancyvrb` options. The description of these options is in a forthcoming section.

```

73 class PygOpts(Options):
74     style = 'default'
75     nobackground = False
76     linenos = False
77     linenostart = 1
78     linenostep = 1
79     commandprefix = 'Py'
80     texcomments = False
81     mathescape = False
82     escapeinside = ""
83     envname = 'Verbatim'
84     lang = 'tex'
85     def __init__(self, *args, **kwargs):
86         super().__init__(*args, **kwargs)
87         self.linenos = Controller.ensure_bool(self.linenos)
88         self.linenostart = abs(int(self.linenostart))

```

```

89     self.linenostep = abs(int(self.linenostep))
90     self.texcomments = Controller.ensure_bool(self.texcomments)
91     self.mathescape = Controller.ensure_bool(self.mathescape)

```

### 3.3 FV nested class

```

92 class FVOpts(Options):
93     gobble = 0
94     tabsize = 4
95     linenosep = 'Opt'
96     commentchar = ''
97     frame = 'none'
98     label = ''
99     labelposition = 'none'
100    numbers = 'left'
101    numbersep = r'\hspace{1ex}'
102    firstnumber = 'auto'
103    stepnumber = 1
104    numberblanklines = True
105    firstline = ''
106    lastline = ''
107    baselinestretch = 'auto'
108    resetmargins = True
109    xleftmargin = 'Opt'
110    xrightmargin = 'Opt'
111    hfuzz = '2pt'
112    samepage = False
113    def __init__(self, *args, **kwargs):
114        super().__init__(*args, **kwargs)
115        self.gobble = abs(int(self.gobble))
116        self.tabsize = abs(int(self.tabsize))
117        if self.firstnumber != 'auto':
118            self.firstnumber = abs(int(self.firstnumber))
119        self.stepnumber = abs(int(self.stepnumber))
120        self.linenostep = abs(int(self.linenostep))
121        self.numberblanklines = Controller.ensure_bool(self.numberblanklines)
122        self.resetmargins = Controller.ensure_bool(self.resetmargins)
123        self.samepage = Controller.ensure_bool(self.samepage)

```

### 3.4 Arguments nested class

```

124 class Arguments(Options):
125     cache = False
126     debug = False
127     code = ""
128     json = ""
129     directory = "."
130     texopts = TeXOpts()
131     pygopts = PygOpts()
132     fv_opts = FVOpts()
133     directory = ""

```

## 4 Controller main class

```
134 class Controller:
```

### 4.1 Computed properties

`self.json_p` The full path to the `json` file containing all the data used for the processing.

*(End definition for `self.json_p`. This variable is documented on page ??.)*

```
135 _json_p = None
136 @property
137 def json_p(self):
138     p = self._json_p
139     if p:
140         return p
141     else:
142         p = self.arguments.json
143         if p:
144             p = Path(p).resolve()
145         self._json_p = p
146     return p
```

`self.pygd_p` The full path to the directory containing the various output files related to `pygments`. When not given inside the `json` file, this is the directory of the `json` file itself. The directory is created when missing.

*(End definition for `self.pygd_p`. This variable is documented on page ??.)*

```
147 _pygd_p = None
148 @property
149 def pygd_p(self):
150     p = self._pygd_p
151     if p:
152         return p
153     p = self.arguments.directory
154     if p:
155         p = Path(p)
156     else:
157         p = self.json_p
158         if p:
159             p = p.parent / p.stem
160         else:
161             p = Path('SHARED')
162     if p:
163         p = p.resolve().with_suffix(".pygd")
164         p.mkdir(exist_ok=True)
165     self._pygd_p = p
166     return p
```

`self.pyg_sty_p` The full path to the style file with definition created by `pygments`.

*(End definition for `self.pyg_sty_p`. This variable is documented on page ??.)*

```

167 @property
168 def pyg_sty_p(self):
169     return (self.pygd_p / self.pygopts.style).with_suffix(".pyg.sty")

```

`self.parser` The correctly set up `argparse` instance.

*(End definition for `self.parser`. This variable is documented on page ??.)*

```

170 @property
171 def parser(self):
172     parser = argparse.ArgumentParser(
173         prog=sys.argv[0],
174         description=''
175 Writes to the output file a set of LaTeX macros describing
176 the syntax highlighting of the input file as given by pygments.
177 ''
178     )
179     parser.add_argument(
180         "-v", "--version",
181         help="Print the version and exit",
182         action='version',
183         version=f'coder-tool version {__version__},'
184         ' (c) {__YEAR__} by Jérôme LAURENS.'
185     )
186     parser.add_argument(
187         "--debug",
188         default=None,
189         help="display informations useful for debugging"
190     )
191     parser.add_argument(
192         "json",
193         metavar="json data file",
194         help=""
195 file name with extension, contains processing information
196 ""
197     )
198     return parser
199

```

## 4.2 Static methods

---

<code>Controller.lua_command</code>	<code>self.lua_command(&lt;(asynchronous lua command)&gt;)</code>
<code>Controller.lua_command_now</code>	<code>self.lua_command_now(&lt;(synchronous lua command)&gt;)</code>

---

Wraps the given command between markers. It will be in the output of the `coder-tool.py`, further captured by `coder-util.lua` and either forwarded to `TEX` or executed synchronously.

```

200 @staticmethod
201 def lua_command(cmd):
202     print(f'<<<<?LUA:{cmd}>>>>')
203 @staticmethod
204 def lua_command_now(cmd):
205     print(f'<<<<!LUA:{cmd}>>>>')

```

## 4.3 Methods

### 4.3.1 `--init--`

---

`--init--` Constructor. Reads the command line arguments.

---

```
206 def __init__(self, argv = sys.argv):
207     argv = argv[1:] if re.match(".*coder\-\tool\.py$", argv[0]) else argv
208     ns = self.parser.parse_args(
209         argv if len(argv) else ['-h']
210     )
211     with open(ns.json, 'r') as f:
212         self.arguments = json.load(
213             f,
214             object_hook=Controller.Object
215         )
216     texopts = self.texopts = self.arguments.texopts
217     pygopts = self.pygopts = self.arguments.pygopts
218     fv_opts = self.fv_opts = self.arguments.fv_opts
219     formatter = self.formatter = LatexFormatter(
220         style=pygopts.style,
221         nobackground = pygopts.nobackground,
222         commandprefix = pygopts.commandprefix,
223         texcomments = pygopts.texcomments,
224         mathescape = pygopts.mathescape,
225         escapeinside = pygopts.escapeinside,
226         envname = u'CDR@Pyg@Verbatim',
227     )
228
229     try:
230         lexer = self.lexer = get_lexer_by_name(self.arguments.lang)
231     except ClassNotFound as err:
232         sys.stderr.write('Error: ')
233         sys.stderr.write(str(err))
234
235     escapeinside = pygopts.escapeinside
236     # When using the LaTeX formatter and the option 'escapeinside' is
237     # specified, we need a special lexer which collects escaped text
238     # before running the chosen language lexer.
239     if len(escapeinside) == 2:
240         left = escapeinside[0]
241         right = escapeinside[1]
242         lexer = self.lexer = LatexEmbeddedLexer(left, right, lexer)
243
244     gobble = fv_opts.gobble
245     if gobble:
246         lexer.add_filter('gobble', n=gobble)
247     tabsize = fv_opts.tabsize
248     if tabsize:
249         lexer.tabsize = tabsize
250     lexer.encoding = ''
251
```

### 4.3.2 get\_pyg\_tex\_p

---

get\_pyg\_tex\_p     $\langle \text{variable} \rangle$  = self.get\_pyg\_tex\_p( $\langle \text{digest string} \rangle$ )

---

The full path of the file where the colored commands created by `pygments` are stored. The digest allows to uniquely identify the code initially colored such that caching is easier.

```
252 def get_pyg_tex_p(self, digest):
253     return (self.pygd_p / digest).with_suffix(".pyg.tex")
```

### 4.3.3 create\_style

---

self.create\_style    self.create\_style()

---

Where the  $\langle \text{style} \rangle$  is created. Does quite nothing if the style is already available.

```
254 def create_style(self):
255     pyg_sty_p = self.pyg_sty_p
256     if self.arguments.cache and pyg_sty_p.exists():
257         print("Already available:", pyg_sty_p)
258         return
259     texopts = self.texopts
260     if texopts.already_style:
261         return
262     formatter = self.formatter
263     style_defs = formatter.get_style_defs() \
264         .replace(r'\makeatletter', '') \
265         .replace(r'\makeatother', '') \
266         .replace('\n', '%\n')
267     sty = self.texopts.sty_template.replace(
268         '<placeholder:style_defs>',
269         style_defs,
270     )
271     with pyg_sty_p.open(mode='w', encoding='utf-8') as f:
272         f.write(sty)
```

### 4.3.4 pygmentize

---

self.pygmentize     $\langle \text{code variable} \rangle$  = self.pygmentize( $\langle \text{code} \rangle$  [, inline= $\langle \text{yorn} \rangle$ ])

---

Where the  $\langle \text{code} \rangle$  is highlighted by `pygments`.

```
273 def pygmentize(self, code):
274     code = highlight(code, self.lexer, self.formatter)
275     m = re.match(
276         r'\begin{CDR@Pyg@Verbatim}.*?\n(.*)\n\end{CDR@Pyg@Verbatim}\s*\Z',
277         code,
278         flags=re.S
279     )
280     assert(m)
281     highlighted = m.group(1)
282     texopts = self.texopts
283     if texopts.inline:
```

```

284     return texopts.code_template.replace('<placeholder:hilighted>',hilighted)
285     fv_opts = self.fv_opts
286     lines = hilighted.split('\n')
287     number = firstnumber = fv_opts.firstnumber
288     stepnumber = fv_opts.stepnumber
289     no = ''
290     numbering = fv_opts.numbers != 'none'
291     ans_code = []
292     def more(template):
293         ans_code.append(template.replace(
294             '<placeholder:number>', f'{number}',
295             ).replace(
296                 '<placeholder:line>', line,
297             ))
298         number += 1
299
300     if len(lines) == 1:
301         line = lines.pop(0)
302         more(texopts.single_line_template)
303     elif len(lines):
304         line = lines.pop(0)
305         more(texopts.first_line_template)
306         line = lines.pop(0)
307         more(texopts.second_line_template)
308         if stepnumber < 2:
309             def template():
310                 return texopts.black_line_template
311         elif stepnumber % 5 == 0:
312             def template():
313                 return texopts.black_line_template if number % \
314                     linenostep == 0 else texopts.white_line_template
315         else:
316             def template():
317                 return texopts.black_line_template if (number - firstnumber) % \
318                     linenostep == 0 else texopts.white_line_template
319
320     for line in lines:
321         more(template())
322
323     hilighted = '\n'.join(ans_code)
324     return texopts.block_template.replace(
325         '<placeholder:count>', f'{counter-firstnumber}'
326     ).replace(
327         '<placeholder:hilighted>', hilighted
328     )
329     """
330     """    ans_code.append(fr'''%
331     """\begin{{CDR@Block/engine/{pygopts.style}}}
332     """\CDRBlock@linenos@used:n {{{','.join(numbers)}}}%
333     """{m.group(1)}{'\n'.join(lines)}{m.group(3)}%
334     """\end{{CDR@Block/engine/{pygopts.style}}}
335     """''')
336     """    ans_code = "".join(ans_code)
337     """    return texopts.block_template.replace('<placeholder:hilighted>',hilighted)

```



#### 4.3.5 create\_pygmented

---

`self.create_pygmented`    `self.create_pygmented()`

---

Call `self.pygmentize` and save the resulting pygmented code at the proper location.

```
338 def create_pygmented(self):
339     arguments = self.arguments
340     code = arguments.code
341     if not code:
342         return False
343     inline = self.texopts.inline
344     h = hashlib.md5(f'{str(code)}:{inline}'.encode('utf-8'))
345     pyg_tex_p = self.get_pyg_tex_p(h.hexdigest())
346     if arguments.cache and pyg_tex_p.exists():
347         print("Already available:", pyg_tex_p)
348         return True
349     code = self.pygmentize(code)
350     with pyg_tex_p.open(mode='w',encoding='utf-8') as f:
351         f.write(code)
352     self.lua_command_now( f'self:input({pyg_tex_p})' )
353 # \CDR_remove:n {{colored:}}%
354 # \input {{ \tl_to_str:n {{}} }}%
355 # \CDR:n {{colored:}}%
356     pyg_sty_p = self.pyg_sty_p
357     if pyg_sty_p.parent.stem != 'SHARED':
358         self.lua_command_now( fr'''
359 CDR:cache_record([=====[{pyg_sty_p.name}]====],[=====[{pyg_tex_p.name}]====])
360 ''' )
361     print("PREMATURE EXIT")
362     exit(1)
```

#### 4.4 Main entry

```
363 if __name__ == '__main__':
364     try:
365         ctrl = Controller()
366         x = ctrl.create_style() or ctrl.create_pygmented()
367         print(f'{sys.argv[0]}: done')
368         sys.exit(x)
369     except KeyboardInterrupt:
370         sys.exit(1)
371 %</py>
```

### File III

## coder.sty implementation

```
1 %<*sty>
2 \makeatletter
```

## 1 Installation test

```
3 \NewDocumentCommand \CDRTest {} {
4   \sys_if_shell:TF {
5     \CDR_has_pygments:F {
6       \msg_warning:nnn
7         { coder }
8         { :n }
9         { No~"pygmentize"~found. }
10    }
11  } {
12    \msg_warning:nnn
13      { coder }
14      { :n }
15      { No~unrestricted~shell~escape~for~"pygmentize".}
16  }
17 }
```

## 2 Messages

```
18 \msg_new:nnn { coder } { unknown-choice } {
19   #1-given-value~'#3'~not-in~#2
20 }
```

## 3 Constants

`\c_CDR_tag` Paths of L3keys modules.  
`\c_CDR_Tags` These are root path components used throughout the package.

```
21 \str_const:Nn \c_CDR_Tags { CDR@Tags }
22 \str_const:Nx \c_CDR_tag { \c_CDR_Tags/tag }
```

*(End definition for \c\_CDR\_tag and \c\_CDR\_Tags. These variables are documented on page ??.)*

`\c_CDR_tag_get` Root identifier for tag properties, used throughout the package.  
`\c_CDR_slash`

```
23 \str_const:Nn \c_CDR_tag_get { CDR@tag@get }
24 \str_const:Nx \c_CDR_slash { \tl_to_str:n {/} }
```

*(End definition for \c\_CDR\_tag\_get and \c\_CDR\_slash. These variables are documented on page ??.)*

## 4 Implementation details

As far as possible, macro making assignments to variables are protected. All variables following `expl3` naming conventions are implementation details and therefore must be considered private.

## 5 Variables

### 5.1 Internal scratch variables

These local variables are used in a very limited scope.

`\l_CDR_bool` Local scratch variable.

25 `\bool_new:N \l_CDR_bool`

*(End definition for \l\_CDR\_bool. This variable is documented on page ??.)*

`\l_CDR_tl` Local scratch variable.

26 `\tl_new:N \l_CDR_tl`

*(End definition for \l\_CDR\_tl. This variable is documented on page ??.)*

`\l_CDR_str` Local scratch variable.

27 `\str_new:N \l_CDR_str`

*(End definition for \l\_CDR\_str. This variable is documented on page ??.)*

`\l_CDR_seq` Local scratch variable.

28 `\seq_new:N \l_CDR_seq`

*(End definition for \l\_CDR\_seq. This variable is documented on page ??.)*

`\l_CDR_prop` Local scratch variable.

29 `\prop_new:N \l_CDR_prop`

*(End definition for \l\_CDR\_prop. This variable is documented on page ??.)*

`\l_CDR_clist` The comma separated list of current chunks.

30 `\clist_new:N \l_CDR_clist`

*(End definition for \l\_CDR\_clist. This variable is documented on page ??.)*

### 5.2 Files

`\l_CDR_in` Input file identifier

31 `\ior_new:N \l_CDR_in`

*(End definition for \l\_CDR\_in. This variable is documented on page ??.)*

`\l_CDR_out` Output file identifier

32 `\iow_new:N \l_CDR_out`

*(End definition for \l\_CDR\_out. This variable is documented on page ??.)*

### 5.3 Global variables

Line number counter for the code chunks.

`\g_CDR_code_int` Chunk number counter.

33 `\int_new:N \g_CDR_code_int`

*(End definition for `\g_CDR_code_int`. This variable is documented on page ??.)*

`\g_CDR_code_prop` Global code property list.

34 `\prop_new:N \g_CDR_code_prop`

*(End definition for `\g_CDR_code_prop`. This variable is documented on page ??.)*

`\g_CDR_chunks_tl` The comma separated list of current chunks. If the next list of chunks is the same as the  
`\l_CDR_chunks_tl` current one, then it might not display.

35 `\tl_new:N \g_CDR_chunks_tl`

36 `\tl_new:N \l_CDR_chunks_tl`

*(End definition for `\g_CDR_chunks_tl` and `\l_CDR_chunks_tl`. These variables are documented on page ??.)*

`\g_CDR_vars` Tree storage for global variables.

37 `\prop_new:N \g_CDR_vars`

*(End definition for `\g_CDR_vars`. This variable is documented on page ??.)*

`\g_CDR_hook_tl` Hook general purpose.

38 `\tl_new:N \g_CDR_hook_tl`

*(End definition for `\g_CDR_hook_tl`. This variable is documented on page ??.)*

`\g/CDR/Chunks/<name>` List of chunk keys for given named code.

*(End definition for `\g/CDR/Chunks/<name>`. This variable is documented on page ??.)*

### 5.4 Local variables

`\l_CDR_keyval_tl` keyval storage.

39 `\tl_new:N \l_CDR_keyval_tl`

*(End definition for `\l_CDR_keyval_tl`. This variable is documented on page ??.)*

`\l_CDR_options_tl` options storage.

40 `\tl_new:N \l_CDR_options_tl`

*(End definition for `\l_CDR_options_tl`. This variable is documented on page ??.)*

`\l_CDR_recorded_tl` Full verbatim body of the CDR environment.

41 `\tl_new:N \l_CDR_recorded_tl`

*(End definition for `\l_CDR_recorded_tl`. This variable is documented on page ??.)*

`\g_CDR_int` Global integer to store linenos locally in time.

42 `\int_new:N \g_CDR_int`

*(End definition for `\g_CDR_int`. This variable is documented on page ??.)*

`\l_CDR_line_tl` Token list for one line.

43 `\tl_new:N \l_CDR_line_tl`

*(End definition for `\l_CDR_line_tl`. This variable is documented on page ??.)*

`\l_CDR_lineno_tl` Token list for lineno display.

44 `\tl_new:N \l_CDR_lineno_tl`

*(End definition for `\l_CDR_lineno_tl`. This variable is documented on page ??.)*

`\l_CDR_name_tl` Token list for chunk name display.

45 `\tl_new:N \l_CDR_name_tl`

*(End definition for `\l_CDR_name_tl`. This variable is documented on page ??.)*

`\l_CDR_info_tl` Token list for the info of line.

46 `\tl_new:N \l_CDR_info_tl`

*(End definition for `\l_CDR_info_tl`. This variable is documented on page ??.)*

## 6 Tag properties

The tag properties concern the code chunks. They are set from different path, such that `\l_keys_path_str` must be properly parsed for that purpose. Commands in this section and the next ones contain `CDR_tag`.

The `<tag names>` starting with a double underscore are reserved by the package.

### 6.1 Helpers

`\g_CDR_tag_path_seq` Global variable to store relative key path. Used for automatic management to know what has been defined explicitly.

47 `\seq_new:N \g_CDR_tag_path_seq`

*(End definition for `\g_CDR_tag_path_seq`. This variable is documented on page ??.)*

---

`\CDR_tag_get_path:cc` ★ `\CDR_tag_get_path:cc {<tag name>} {<relative key path>}`

Internal: return a unique key based on the arguments. Used to store and retrieve values.

48 `\cs_new:Npn \CDR_tag_get_path:cc #1 #2 {`

49 `\c_CDR_tag_get @ #1 / #2 :`

50 `}`

## 6.2 Set

---

<code>\CDR_tag_set:ccn</code> <code>\CDR_tag_set:ccV</code>	<code>\CDR_tag_set:ccn {&lt;tag name&gt;} {&lt;relative key path&gt;} {&lt;value&gt;}</code> Store <i>&lt;value&gt;</i> , which is further retrieved with the instruction <code>\CDR_tag_get:cc {&lt;tag name&gt;} {&lt;relative key path&gt;}</code> . Only <i>&lt;tag name&gt;</i> and <i>&lt;relative key path&gt;</i> containing no @ character are supported. Record the relative key path (the part after the tag name) of the current full key path in <code>g_CDR_tag_path_seq</code> . All the affectations are made at the current $\TeX$ group level. <i>Nota Bene:</i> <code>\cs_generate_variant:Nn</code> is buggy when there is a 'c' argument.
--	---

---

```

51 \cs_new_protected:Npn \CDR_tag_set:ccn #1 #2 #3 {
52   \seq_put_left:Nx \g_CDR_tag_path_seq { #2 }
53   \cs_set:cpn { \CDR_tag_get_path:cc { #1 } { #2 } } { \exp_not:n { #3 } }
54 }
55 \cs_new_protected:Npn \CDR_tag_set:ccV #1 #2 #3 {
56   \exp_args:NnnV
57   \CDR_tag_set:ccn { #1 } { #2 } #3
58 }

```

`\c_CDR_tag_regex` To parse a l3keys full key path.

```

59 \tl_set:Nn \l_CDR_tl { /([~/]*)/(.*)$ } % $
60 \tl_put_left:NV \l_CDR_tl \c_CDR_tag
61 \tl_put_left:Nn \l_CDR_tl { ^ }
62 \exp_args:NNV
63 \regex_const:Nn \c_CDR_tag_regex \l_CDR_tl

```

(End definition for `\c_CDR_tag_regex`. This variable is documented on page ??.)

---

<code>\CDR_tag_set:n</code>	<code>\CDR_tag_set:n {&lt;value&gt;}</code>
-----------------------------	---

---

The value is provided but not the *<dir>* nor the *<relative key path>*, both are guessed from `\l_keys_path_str`. More precisely, `\l_keys_path_str` is expected to read something like `\c_CDR_tag/<tag name>/<relative key path>`, an exception is raised on the contrary. This is meant to be call from `\keys_define:nn` argument. Implementation detail: the last argument is parsed by the last command.

```

64 \cs_new:Npn \CDR_tag_set:n {
65   \exp_args:NnV
66   \regex_extract_once:NnNTF \c_CDR_tag_regex
67     \l_keys_path_str \l_CDR_seq {
68     \CDR_tag_set:ccn
69     { \seq_item:Nn \l_CDR_seq 2 }
70     { \seq_item:Nn \l_CDR_seq 3 }
71   } {
72     \PackageWarning
73     { coder }
74     { Unexpected-key~path~'\l_keys_path_str' }
75     \use_none:n
76   }
77 }

```

---

\CDR\_tag\_set:    \CDR\_tag\_set:

None of  $\langle dir \rangle$ ,  $\langle relative\ key\ path \rangle$  and  $\langle value \rangle$  are provided. The latter is guessed from  $\backslash l\_keys\_value\_tl$ , and  $\backslash CDR\_tag\_set:n$  is called. This is meant to be call from  $\backslash keys\_define:nn$  argument.

```
78 \cs_new:Npn \CDR_tag_set: {  
79   \exp_args:NV  
80   \CDR_tag_set:n \l_keys_value_tl  
81 }
```

---

\CDR\_tag\_set:cn    \CDR\_tag\_set:cn  $\{\langle key\ path \rangle\} \{\langle value \rangle\}$

When the last component of  $\backslash l\_keys\_path\_str$  should not be used to store the  $\langle value \rangle$ , but  $\langle key\ path \rangle$  should be used instead. This last component is replaced and  $\backslash CDR\_tag\_set:n$  is called afterwards. Implementation detail: the second argument is parsed by the last command of the expansion.

```
82 \cs_new:Npn \CDR_tag_set:cn #1 {  
83   \exp_args:NnV  
84   \regex_extract_once:NnNTF \c_CDR_tag_regex  
85     \l_keys_path_str \l_CDR_seq {  
86     \CDR_tag_set:ccn  
87     { \seq_item:Nn \l_CDR_seq 2 }  
88     { #1 }  
89   } {  
90     \PackageWarning  
91       { coder }  
92       { Unexpected-key-path-'\l_keys_path_str' }  
93     \use_none:n  
94   }  
95 }
```

---

\CDR\_tag\_choices:    \CDR\_tag\_choices:

Ensure that the  $\backslash l\_keys\_path\_str$  is set properly. This is where a syntax like  $\backslash keys\_set:nn \{...\} \{ choice/a \}$  is managed.

```
96 \cs_new:Npn \CDR_tag_choices: {  
97   \exp_args:NVV  
98   \str_if_eq:nnT \l_keys_key_tl \l_keys_choice_tl {  
99     \exp_args:NnV  
100    \regex_extract_once:nnNT { ^(.*)/.*$ } % $  
101      \l_keys_path_str \l_CDR_seq {  
102      \str_set:Nx \l_keys_path_str {  
103      \seq_item:Nn \l_CDR_seq 2  
104      }  
105    }  
106  }  
107 }
```

---

\CDR\_tag\_choices\_set:    \CDR\_tag\_choices\_set:

Calls  $\backslash CDR\_tag\_set:n$  with the content of  $\backslash l\_keys\_choice\_tl$  as value. Before, ensure that the  $\backslash l\_keys\_path\_str$  is set properly.

```

108 \cs_new:Npn \CDR_tag_choices_set: {
109   \CDR_tag_choices:
110   \exp_args:NV
111   \CDR_tag_set:n \l_keys_choice_tl
112 }

```

---

$\backslash\text{CDR\_if\_truthy:nTF}$ $\backslash\text{CDR\_if\_truthy:eTF}$	$\backslash\text{CDR\_if\_truthy:nTF}$ $\{\langle\text{token list}\rangle\} \{\langle\text{true code}\rangle\} \{\langle\text{false code}\rangle\}$ Execute $\langle\text{true code}\rangle$ when $\langle\text{token list}\rangle$ is a truthy value, $\langle\text{false code}\rangle$ otherwise. A truthy value is a text which leading character, if any, is none of “fFnN”.
--	---

---

```

113 \prg_new_conditional:Nnn \CDR_if_truthy:n { p, T, F, TF } {
114   \str_compare:nNnTF { \str_lowercase:n { #1 } } = { false } {
115     \prg_return_false:
116   } {
117     \prg_return_true:
118   }
119 }
120 \prg_generate_conditional_variant:Nnn \CDR_if_truthy:n { e } { p, T, F, TF }

```

---

$\backslash\text{CDR\_tag\_boolean\_set:n}$	$\backslash\text{CDR\_tag\_boolean\_set:n}$ $\{\langle\text{choice}\rangle\}$ Calls $\backslash\text{CDR\_tag\_set:n}$ with true if the argument is truthy, false otherwise.
---	---

---

```

121 \cs_new:Npn \CDR_tag_boolean_set:n #1 {
122   \CDR_if_truthy:nTF { #1 } {
123     \CDR_tag_set:n { true }
124   } {
125     \CDR_tag_set:n { false }
126   }
127 }
128 \cs_generate_variant:Nn \CDR_tag_boolean_set:n { x }

```

### 6.3 Retrieving tag properties

Internally, all tag properties are collected with a full key path like  $\backslash\text{c\_CDR\_tag\_get}/\langle\text{tag name}\rangle/\langle\text{relative key path}\rangle$ . When typesetting some code with either the  $\backslash\text{CDRCode}$  command or the  $\text{CDRBlock}$  environment, all properties defined locally are collected under the reserved  $\backslash\text{c\_CDR\_tag\_get}/\_\_\text{local}/\langle\text{relative path}\rangle$  full key paths. The  $\text{l3keys}$  module  $\backslash\text{c\_CDR\_tag\_get}/\_\_\text{local}$  is modified in  $\text{T}_{\text{E}}\text{X}$  groups only. For running text code chunks, this module inherits from

1.  $\backslash\text{c\_CDR\_tag\_get}/\langle\text{tag name}\rangle$  for the provided  $\langle\text{tag name}\rangle$ ,
2.  $\backslash\text{c\_CDR\_tag\_get}/\text{default.code}$
3.  $\backslash\text{c\_CDR\_tag\_get}/\text{default}$
4.  $\backslash\text{c\_CDR\_tag\_get}/\_\_\text{pygments}$
5.  $\backslash\text{c\_CDR\_tag\_get}/\_\_\text{fancyvrb}$
6.  $\backslash\text{c\_CDR\_tag\_get}/\_\_\text{fancyvrb.all}$  when no using pygments



For text block code chunks, this module inherits from

1. \c\_CDR\_tag\_get/⟨name<sub>1</sub>⟩, ..., \c\_CDR\_tag\_get/⟨name<sub>n</sub>⟩ for each tag name of the ordered tags list
2. \c\_CDR\_tag\_get/default.block
3. \c\_CDR\_tag\_get/default
4. \c\_CDR\_tag\_get/\_\_\_pygments
5. \c\_CDR\_tag\_get/\_\_\_pygments.block
6. \c\_CDR\_tag\_get/\_\_\_fancyvrb
7. \c\_CDR\_tag\_get/\_\_\_fancyvrb.block
8. \c\_CDR\_tag\_get/\_\_\_fancyvrb.all when no using pygments

---

```
\CDR_tag_if_exist_here:ccTF * \CDR_tag_if_exist_here:ccTF {⟨tag name⟩} ⟨relative key path⟩ {⟨true code⟩} {⟨false code⟩}
```

---

If the ⟨relative key path⟩ is known within ⟨tag name⟩, the ⟨true code⟩ is executed, otherwise, the ⟨false code⟩ is executed. No inheritance.

```
129 \prg_new_conditional:Nnn \CDR_tag_if_exist_here:cc { T, F, TF } {
130   \cs_if_exist:cTF { \CDR_tag_get_path:cc { #1 } { #2 } } {
131     \prg_return_true:
132   } {
133     \prg_return_false:
134   }
135 }
```

---

```
\CDR_tag_if_exist:ccTF * \CDR_tag_if_exist:ccTF {⟨tag name⟩} ⟨relative key path⟩ {⟨true code⟩} {⟨false code⟩}
```

---

If the ⟨relative key path⟩ is known within ⟨tag name⟩, the ⟨true code⟩ is executed, otherwise, the ⟨false code⟩ is executed if none of the parents has the ⟨relative key path⟩ on its own.

```
136 \prg_new_conditional:Nnn \CDR_tag_if_exist:cc { T, F, TF } {
137   \cs_if_exist:cTF { \CDR_tag_get_path:cc { #1 } { #2 } } {
138     \prg_return_true:
139   } {
140     \seq_if_exist:cTF { \CDR_tag_parent_seq:c { #1 } } {
141       \seq_map_tokens:cn
142         { \CDR_tag_parent_seq:c { #1 } }
143         { \CDR_tag_if_exist_f:cn { #2 } }
144     } {
145       \prg_return_false:
146     }
147   }
148 }
149 \cs_new:Npn \CDR_tag_if_exist_f:cn #1 #2 {
150   \quark_if_no_value:nTF { #2 } {
```

```

151     \seq_map_break:n {
152       \prg_return_false:
153     }
154   } {
155     \CDR_tag_if_exist:ccT { #2 } { #1 } {
156       \seq_map_break:n {
157         \prg_return_true:
158       }
159     }
160   }
161 }

```

---

\CDR\_tag\_get:cc ★ \CDR\_tag\_get:cc {<tag name>} {<relative key path>}

The property value stored for <tag name> and <relative key path>. Takes care of inheritance.

```

162 \cs_new:Npn \CDR_tag_get:cc #1 #2 {
163   \CDR_tag_if_exist_here:ccTF { #1 } { #2 } {
164     \use:c { \CDR_tag_get_path:cc { #1 } { #2 } }
165   } {
166     \seq_if_exist:cT { \CDR_tag_parent_seq:c { #1 } } {
167       \seq_map_tokens:cn
168         { \CDR_tag_parent_seq:c { #1 } }
169         { \CDR_tag_get_f:cn { #2 } }
170     }
171   }
172 }
173 \cs_new:Npn \CDR_tag_get_f:cn #1 #2 {
174   \quark_if_no_value:nF { #2 } {
175     \CDR_tag_if_exist_here:ccT { #2 } { #1 } {
176       \seq_map_break:n {
177         \use:c { \CDR_tag_get_path:cc { #2 } { #1 } }
178       }
179     }
180   }
181 }

```

---

\CDR\_tag\_get:c ★ \CDR\_tag\_get:n {<relative key path>}

The property value stored for the \_\_local <tag name> and <relative key path>. Takes care of inheritance. Implementation detail: the parameter is parsed by the last command of the expansion.

```

182 \cs_new:Npn \CDR_tag_get:c {
183   \CDR_tag_get:cc { __local }
184 }

```

---

\CDR\_tag\_get:cN    \CDR\_tag\_get:cN {<relative key path>} {<tl variable>}

Put in <tl variable> the property value stored for the \_\_local <tag name> and <relative key path>.

```
185 \cs_new:Npn \CDR_tag_get:cN #1 #2 {
186   \tl_set:Nx #2 { \CDR_tag_get:c { #1 } }
187 }
```

---

\CDR\_tag\_get:ccNTF    \CDR\_tag\_get:ccNTF {<tag name>} {<relative key path>} <tl var> {<true code>} {<false code>}

Getter with branching. If the <relative key path> is known, save the value into <tl var> and execute <true code>. Otherwise, execute <false code>.

```
188 \prg_new_conditional:Nnn \CDR_tag_get:ccN { T, F, TF } {
189   \CDR_tag_if_exist:nnTF { #1 } { #2 } {
190     \tl_set:Nx #3 \CDR_tag_get:cc { #1 } { #2 }
191     \prg_return_true:
192   } {
193     \prg_return_false:
194   }
195 }
```

## 6.4 Inheritance

When a child inherits from a parent, all the keys of the parent that are not inherited are made available to the child (inheritance does not jump over generations).

---

\CDR\_tag\_parent\_seq:c \*    \CDR\_tag\_parent\_seq:c {<tag name>}

Return the name of the sequence variable containing the list of the parents. Each child has its own sequence of parents.

```
196 \cs_new:Npn \CDR_tag_parent_seq:c #1 {
197   g_CDR:parent.tag @ #1 _seq
198 }
```

---

\CDR\_tag\_inherit:cn    \CDR\_tag\_inherit:cn {<child name>} {<parent names comma list>}

Set the parents of <child name> to the given list.

```
199 \cs_new:Npn \CDR_tag_inherit:cn #1 #2 {
200   \seq_set_from_clist:cn { \CDR_tag_parent_seq:c { #1 } } { #2 }
201   \seq_remove_duplicates:c \l_CDR_tl
202   \seq_remove_all:cn \l_CDR_tl {}
203   \seq_put_right:cn \l_CDR_tl { \q_no_value }
204 }
205 \cs_new:Npn \CDR_tag_inherit:cx {
206   \exp_args:Nnx \CDR_tag_inherit:cn
207 }
208 \cs_new:Npn \CDR_tag_inherit:cV {
209   \exp_args:NnV \CDR_tag_inherit:cn
210 }
```

## 7 Cache management

If there is no `<jobname>.aux` file, there should be no cached files either, `coder-util.lua` is asked to clean all of them, if any.

```
211 \AddToHook { begindocument/before } {
212   \IfFileExists {./\jobname.aux} {} {
213     \lua_now:n {CDR:cache_clean_all()}
214   }
215 }
```

At the end of the document, `coder-util.lua` is asked to clean all unused cached files that could come from a previous process.

```
216 \AddToHook { enddocument/end } {
217   \lua_now:n {CDR:cache_clean_unused()}
218 }
```

## 8 Utilities

---

<code>\CDR_clist_map_inline:Nnn</code>	<code>\CDR_clist_map_inline:Nnn &lt;clist var&gt; {&lt;empty code&gt;} {&lt;non empty code&gt;}</code> Execute <code>&lt;empty code&gt;</code> when the list is empty, otherwise call <code>\clist_map_inline:Nn</code> with <code>&lt;non empty code&gt;</code> .
--	---

```
219 \cs_new:Npn \CDR_clist_map_inline:Nnn #1 #2 {
220   \clist_if_empty:NTF #1 {
221     #2
222     \use_none:n
223   } {
224     \clist_map_inline:Nn #1
225   }
226 }
```

---

<code>\CDR_if_block_p: *</code>	<code>\CDR_if_block:TF {&lt;true code&gt;} {&lt;false code&gt;}</code>
<code>\CDR_if_block:TF *</code>	Execute <code>&lt;true code&gt;</code> when inside a code block, <code>&lt;false code&gt;</code> when inside an inline code. Raises an error otherwise.

```
227 \prg_new_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
228   \PackageError
229     { coder }
230     { Conditional-not-available }
231 }
```

---

<code>\CDR_process_record:</code>	Record the current line or not. The default implementation does nothing and is meant to be defines locally.
-----------------------------------	---

```
232 \cs_new:Npn \CDR_process_record: {}
```

## 9 13keys modules for code chunks

All these modules are initialized at the beginning of the document using the `__initialize` meta key.

### 9.1 Utilities

---

```
\CDR_tag_keys_define:nn \CDR_tag_keys_define:nn {< module base >} {< keyval list >}
```

---

The `<module>` is uniquely based on `<module base>` before forwarding to `\keys_define:nn`.

```
233 \cs_generate_variant:Nn \keys_define:nn { Vn, xn }
234 \cs_new:Npn \CDR_tag_keys_define:nn #1 {
235   \keys_define:xn { \c_CDR_tag / \exp_not:n { #1 } }
236 }
237 \cs_generate_variant:Nn \CDR_tag_keys_define:nn { nx }
```

---

```
\CDR_tag_keys_set:nn \CDR_tag_keys_set:nn {<module base>} {<keyval list>}
```

---

The `<module>` is uniquely based on `<module base>` before forwarding to `\keys_set:nn`.

```
238 \cs_new:Npn \CDR_tag_keys_set:nn #1 {
239   \exp_args:Nx
240   \keys_set:nn { \c_CDR_tag / \exp_not:n { #1 } }
241 }
```

#### 9.1.1 Handling unknown tags

While using `\keys_set:nn` and variants, each time a full key path matching the pattern `\c_CDR_tag/<tag name>/<relative key path>` is not recognized, we assume that the client implicitly wants a tag with the given `<tag name>` to be defined. For that purpose, we collect unknown keys with `\keys_set_known:nnnN` then process them to find each `<tag name>` and define the new tag accordingly. A similar situation occurs for display engine options where the full key path reads `\c_CDR_tag/<tag name>/<engine name>` engine options where `<engine name>` is not known in advance.

---

```
\CDR_keys_set_known:nnN \CDR_keys_set_known:nnN {<module>} {<key[=value] items>} <tl var>
```

---

Wrappers over `\keys_set_known:nnnN` where the `<root>` is also the `<module>`.

```
242 \cs_new:Npn \CDR_keys_set_known:nnN #1 #2 {
243   \keys_set_known:nnnN { #1 } { #2 } { #1 }
244 }
245 \cs_generate_variant:Nn \CDR_keys_set_known:nnN { x, VV }
```

---

```
\CDR_tag_keys_set_known:nnN \CDR_tag_keys_set_known:nnN {<tag name>} {<key[=value] items>} <tl var>
```

---

Wrappers over `\keys_set_known:nnnN` where the module is given by `\c_CDR_tag/<tag name>`. *Implementation detail* the remaining arguments are absorbed by the last macro.

```

246 \cs_generate_variant:Nn \keys_set_known:nnnN { VVV, nVx }
247 \cs_new:Npn \CDR_tag_keys_set_known:nnN #1 {
248   \CDR_keys_set_known:xnN { \c_CDR_tag / \exp_not:n { #1 } }
249 }
250 \cs_generate_variant:Nn \CDR_tag_keys_set_known:nnN { nV }

```

`\c_CDR_provide_regex` To parse a l3keys full key path.

```

251 \tl_set:Nn \l_CDR_tl { /([~/*])(?:/(.))*? $ } % $
252 \tl_put_left:NV \l_CDR_tl \c_CDR_tag
253 \tl_put_left:Nn \l_CDR_tl { ^ }
254 \exp_args:NNV
255 \regex_const:Nn \c_CDR_provide_regex \l_CDR_tl

```

*(End definition for \c\_CDR\_provide\_regex. This variable is documented on page ??.)*

---

```

\CDR_tag_provide_from_clist:n    \CDR_tag_provide_from_clist:n {<deep comma list>}
\CDR_tag_provide_from_keyval:n  \CDR_tag_provide_from_keyval:n {<key-value list>}

```

---

`<deep comma list>` has format `tag/<tag name comma list>`. Parse the `<key-value list>` for full key path matching `tag/<tag name>/<relative key path>`, then ensure that `\c_CDR_tag/<tag name>` is a known full key path. For that purpose, we use `\keyval_parse:nnn` with two `\CDR_tag_provide:` helper.

Notice that a tag name should contain no `'/`.

```

256 \regex_const:Nn \c_CDR_engine_regex { ^[~/]*\sengine\soptions$ }
257 \cs_new:Npn \CDR_tag_provide_from_clist:n #1 {
258   \exp_args:NNx
259   \regex_extract_once:NnNTF \c_CDR_provide_regex {
260     \c_CDR_Tags / #1
261   } \l_CDR_seq {
262     \tl_set:Nx \l_CDR_tl { \seq_item:Nn \l_CDR_seq 3 }
263     \exp_args:Nx
264     \clist_map_inline:nn {
265       \seq_item:Nn \l_CDR_seq 2
266     } {
267       \exp_args:NV
268       \keys_if_exist:nnF \c_CDR_tag { ##1 } {
269         \CDR_keys_inherit:Vnn \c_CDR_tag { ##1 } {
270           __pygments, __pygments.block,
271           default.block, default.code, default,
272           __fancyvrb, __fancyvrb.block, __fancyvrb.all
273         }
274         \keys_define:Vn \c_CDR_tag {
275           ##1 .code:n = \CDR_tag_keys_set:nn { ##1 } { #####1 },
276           ##1 .value_required:n = true,
277         }
278       }
279       \exp_args:NxV
280       \keys_if_exist:nnF { \c_CDR_tag / ##1 } \l_CDR_tl {
281         \exp_args:NNV
282         \regex_match:NnT \c_CDR_engine_regex
283         \l_CDR_tl {
284           \CDR_tag_keys_define:nx { ##1 } {
285             \l_CDR_tl .code:n = \exp_not:n { \CDR_tag_set:n { #####1 } },

```

```

286         \l_CDR_tl .value_required:n = true,
287     }
288 }
289 }
290 }
291 } {
292     \regex_match:NnT \c_CDR_engine_regex { #1 } {
293         \CDR_tag_keys_define:nn { default } {
294             #1 .code:n = \CDR_tag_set:n { ##1 },
295             #1 .value_required:n = true,
296         }
297     }
298 }
299 }
300 \cs_new:Npn \CDR_tag_provide_from_clist:nn #1 #2 {
301     \CDR_tag_provide_from_clist:n { #1 }
302 }
303 \cs_new:Npn \CDR_tag_provide_from_keyval:n {
304     \keyval_parse:nnn {
305         \CDR_tag_provide_from_clist:n
306     } {
307         \CDR_tag_provide_from_clist:nn
308     }
309 }
310 \cs_generate_variant:Nn \CDR_tag_provide_from_keyval:n { V }

```

## 9.2 pygments

These are `pygments`'s `LatexFormatter` options, that are not covered by `__fancyvrb`. They are made available at the end user level, but may not be relevant when `pygments` is not used.

### 9.2.1 Utilities

---

<code>\CDR_has_pygments_p: *</code> <code>\CDR_has_pygments:<u>TF</u> *</code>	<code>\CDR_has_pygments:TF {&lt;true code&gt;} {&lt;false code&gt;}</code> Execute <i>&lt;true code&gt;</i> when <code>pygments</code> is available, <i>&lt;false code&gt;</i> otherwise. <i>Implementation detail:</i> we define the conditionals and set them afterwards.
---	--

---

```

311 \sys_get_shell:nnN {which-pygmentize} {} \l_CDR_tl
312 \prg_new_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } { }
313 \tl_if_in:NnTF \l_CDR_tl { pygmentize } {
314     \prg_set_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } {
315         \prg_return_true:
316     }
317 } {
318     \prg_set_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } {
319         \prg_return_false:
320     }
321 }

```

### 9.2.2 `__pygment` `l3keys` module

```

322 \CDR_tag_keys_define:nn { __pygments } {
  ● lang=<language name> where <language name> is recognized by pygments, including a
    void string,

323   lang .code:n = \CDR_tag_set:,
324   lang .value_required:n = true,

  ● pygments[=true|false] whether pygments should be used for syntax coloring. Initially
    true if pygments is available, false otherwise.

325   pygments .code:n = \CDR_tag_boolean_set:x { #1 },

  ● style=<style name> where <style name> is recognized by pygments, including a void
    string,

326   style .code:n = \CDR_tag_set:,
327   style .value_required:n = true,

  ● commandprefix=<text> The  $\text{\LaTeX}$  commands used to produce colored output are con-
    structed using this prefix and some letters. Initially PY.

328   commandprefix .code:n = \CDR_tag_set:,
329   commandprefix .value_required:n = true,

  ● mathescape[=true|false] If set to true, enables  $\text{\LaTeX}$  math mode escape in comments.
    That is, $...$ inside a comment will trigger math mode. Initially false.

330   mathescape .code:n = \CDR_tag_boolean_set:x { #1 },
331   mathescape .default:n = true,

  ● escapeinside=<before><after> If set to a string of length 2, enables escaping to  $\text{\LaTeX}$ .
    Text delimited by these 2 characters is read as  $\text{\LaTeX}$  code and typeset accordingly.
    It has no effect in string literals. It has no effect in comments if texcomments or
    mathescape is set. Initially empty.

332   escapeinside .code:n = \CDR_tag_set:,
333   escapeinside .value_required:n = true,

  ● __initialize Initializer.

334   __initialize .meta:n = {
335     lang = tex,
336     pygments = \CDR_has_pygments:TF { true } { false },
337     style=default,
338     commandprefix=PY,
339     mathescape=false,
340     escapeinside=,
341   },
342   __initialize .value_forbidden:n = true,

343 }
344 \AtBeginDocument{
345   \CDR_tag_keys_set:nn { __pygments } { __initialize }
346 }

```



### 9.2.3 \c\_CDR\_tag / \_\_pygments.block l3keys module

```
347 \CDR_tag_keys_define:nn { __pygments.block } {
```

- **texcomments**[=*true|false*] If set to *true*, enables L<sup>A</sup>T<sub>E</sub>X comment lines. That is, L<sup>A</sup>T<sub>E</sub>X markup in comment tokens is not escaped so that L<sup>A</sup>T<sub>E</sub>X can render it. Initially *false*.

```
348 texcomments .code:n = \CDR_tag_boolean_set:x { #1 },
349 texcomments .default:n = true,
```

- **\_\_initialize** Initializer.

```
350 __initialize .meta:n = {
351 texcomments=false,
352 },
353 __initialize .value_forbidden:n = true,

354 }
355 \AtBeginDocument{
356 \CDR_tag_keys_set:nn { __pygments.block } { __initialize }
357 }
```

## 9.3 Specific to coder

### 9.3.1 default l3keys module

```
358 \CDR_tag_keys_define:nn { default } {
```

Keys are:

- **post processor**=*<command>* the command for pygments post processor. This is a string where every occurrence of “%%file%%” is replaced by the full path of the \*.pyg.tex file to be post processed and then executed as terminal instruction. Initially empty.

```
359 post~processor .code:n = \CDR_tag_set:,
360 post~processor .value_required:n = true,
```

- **parskip** the value of the \parskip in code blocks,

```
361 parskip .code:n = \CDR_tag_set:,
362 parskip .value_required:n = true,
```

- **engine**=*<engine name>* to specify the engine used to display inline code or blocks. Initially default.

```
363 engine .code:n = \CDR_tag_set:,
364 engine .value_required:n = true,
```

- **default engine options**=*<default engine options>* to specify the corresponding options,

```
365 default~engine~options .code:n = \CDR_tag_set:,
366 default~engine~options .value_required:n = true,
```

● **<engine name> engine options=<engine options>** to specify the options for the named engine,

● **\_\_initialize** to initialize storage properly. We cannot use **.initial:n** actions because the **\l\_keys\_path\_str** is not set up properly.

```
367 __initialize .meta:n = {
368     post-processor = ,
369     parskip = \the\parskip,
370     engine = default,
371     default~engine~options = ,
372 },
373 __initialize .value_forbidden:n = true,
374 }
375 \AtBeginDocument{
376     \CDR_tag_keys_set:nn { default } { __initialize }
377 }
```

### 9.3.2 default.code l3keys module

Void for the moment.

```
378 \CDR_tag_keys_define:nn { default.code } {
```

Known keys include:

● **\_\_initialize** to initialize storage properly. We cannot use **.initial:n** actions because the **\l\_keys\_path\_str** is not set up properly.

```
379 __initialize .meta:n = {
380 },
381 __initialize .value_forbidden:n = true,
382 }
383 \AtBeginDocument{
384     \CDR_tag_keys_set:nn { default.code } { __initialize }
385 }
```

### 9.3.3 default.block l3keys module

```
386 \CDR_tag_keys_define:nn { default.block } {
```

Known keys include:

● **show tags[=true|false]** to enable/disable the display of the code chunks tags. Initially true.

● **tags=<tag name comma list>** to export and display.

```
387 tags .code:n = {
388     \clist_set:Nn \l_CDR_tags_clist { #1 }
389     \clist_remove_duplicates:N \l_CDR_tags_clist
390     \exp_args:NV
391     \CDR_tag_set:n \l_CDR_tags_clist
392 },
```

```
393 show~tags .code:n = \CDR_tag_boolean_set:x { #1 },
```

- **only top[=true|false]** to avoid chunk tags repetitions, if on the same page, two consecutive code chunks have the same tag names, the second names are not displayed.

```
394 only~top .code:n = \CDR_tag_boolean_set:x { #1 },
```

- **use margin[=true|false]** to use the margin to display line numbers and tag names, or not,

```
395 use~margin .code:n = \CDR_tag_boolean_set:x { #1 },
```

- **tags format=<format>** , where <format> is used to display the tag names (mainly font, size and color),

```
396 tags~format .code:n = \CDR_tag_set:,
397 tags~format .value_required:n = true,
```

- **blockskip** the separation with the surrounding text, above and below. Initially \topsep.

```
398 blockskip .code:n = \CDR_tag_set:,
399 blockskip .value_required:n = true,
```

- **\_\_initialize** the separation with the surrounding text. Initially \topsep.

```
400 __initialize .meta:n = {
401   tags = ,
402   show~tags = true,
403   only~top = true,
404   use~margin = true,
405   tags~format = {
406     \sffamily
407     \scriptsize
408     \color{gray}
409   },
410   blockskip = \topsep,
411 },
412 __initialize .value_forbidden:n = true,

413 }
414 \AtBeginDocument{
415   \CDR_tag_keys_set:nn { default.block } { __initialize }
416 }
```

## 9.4 fancyvrb

These are fancyvrb options verbatim. The fancyvrb manual has more details, only some parts are reproduced hereafter. All of these options may not be relevant for all situations. Some of them make no sense in code mode, whereas others may not be compatible with the display engine.

#### 9.4.1 \c\_CDR\_tag/\_\_fancyvrb l3keys module

```
417 \CDR_tag_keys_define:nn { __fancyvrb } {
```

- **formatcom**=*<command>* execute before printing verbatim text. Initially empty. Ignored in code mode.

```
418   formatcom .code:n = \CDR_tag_set:,
419   formatcom .value_required:n = true,
```

- **fontfamily**=*<font name>* font family to use. tt, courier and helvetica are predefined. Initially tt.

```
420   fontfamily .code:n = \CDR_tag_set:,
421   fontfamily .value_required:n = true,
```

- **fontsize**=*<font size>* size of the font to use. If you use the relsize package as well, you can require a change of the size proportional to the current one (for instance: **fontsize**=\relsize{-2}). Initially auto: the same as the current font.

```
422   fontsize .code:n = \CDR_tag_set:,
423   fontsize .value_required:n = true,
```

- **fontshape**=*<font shape>* font shape to use. Initially auto: the same as the current font.

```
424   fontshape .code:n = \CDR_tag_set:,
425   fontshape .value_required:n = true,
```

- **fontseries**=*<series name>* L<sup>A</sup>T<sub>E</sub>X font series to use. Initially auto: the same as the current font.

```
426   fontseries .code:n = \CDR_tag_set:,
427   fontseries .value_required:n = true,
```

- **showspaces**[=true|false] print a special character representing each space. Initially false: spaces not shown.

```
428   showspaces .code:n = \CDR_tag_boolean_set:x { #1 },
```

- **showtabs**=true|false explicitly show tab characters. Initially false: tab characters not shown.

```
429   showtabs .code:n = \CDR_tag_boolean_set:x { #1 },
```

- **obeytabs**=true|false position characters according to the tabs. Initially false: tab characters are added to the current position.


```
430   obeytabs .code:n = \CDR_tag_boolean_set:x { #1 },
```

- **tabsize**=*<integer>* number of spaces given by a tab character, Initially 2 (8 for fancyvrb).

```

431  tabsize .code:n = \CDR_tag_set:,
432  tabsize .value_required:n = true,

```

 **defineactive**=*<macro>* to define the effect of active characters. This allows to do some devious tricks, see the fancyvrb package. Initially empty.

```

433  defineactive .code:n = \CDR_tag_set:,
434  defineactive .value_required:n = true,


```

 **relabel**=*<label>* define a label to be used with \pageref. Initially empty.

```

435  relabel .code:n = \CDR_tag_set:,
436  relabel .value_required:n = true,

```

 **\_\_initialize** Initialization.

```

437  __initialize .meta:n = {
438    formatcom = ,
439    fontfamily = tt,
440    fontsize = auto,
441    fontseries = auto,
442    fontshape = auto,
443    showspace = false,
444    showtabs = false,
445    obeytabs = false,
446    tabsize = 2,
447    defineactive = ,
448    relabel = ,
449  },
450  __initialize .value_forbidden:n = true,
451 }
452 \AtBeginDocument{
453   \CDR_tag_keys_set:nn { __fancyvrb } { __initialize }
454 }

```


#### 9.4.2 \_\_fancyvrb.block l3keys module

Block specific options.

```

455 \CDR_tag_keys_define:nn { __fancyvrb.block } {


```

 **commentchar**=*<character>* lines starting with this character are ignored. Initially empty.

```

456  commentchar .code:n = \CDR_tag_set:,
457  commentchar .value_required:n = true,

```

 **gobble**=*<integer>* number of characters to suppress at the beginning of each line (from 0 to 9), mainly useful when environments are indented. Only block mode.

```

458  gobble .choices:nn = {
459    0,1,2,3,4,5,6,7,8,9
460  } {
461    \CDR_tag_choices_set:
462  },

```

- **frame=none|leftline|topline|bottomline|lines|single** type of frame around the verbatim environment. With **leftline** and **single** modes, a space of a length given by the L<sup>A</sup>T<sub>E</sub>X `\fboxsep` macro is added between the left vertical line and the text. Initially **none**: no frame.

```
463 frame .choices:nn =
464   { none, leftline, topline, bottomline, lines, single }
465   { \CDR_tag_choices_set: },
```

- **label={[<top string>]<string>}** label(s) to print on top, bottom or both, frame lines. If the label(s) contains special characters, comma or equal sign, it must be placed inside a group. If an optional **<top string>** is given between square brackets, it will be used for the top line and **<string>** for the bottom line. Otherwise, **<string>** is used for both the top or bottom lines. Label(s) are printed only if the **frame** parameter is one of **topline**, **bottomline**, **lines** or **single**. Initially empty: no label.

```
466 label .code:n = \CDR_tag_set:,
467 label .value_required:n = true,
```

- **labelposition=none|topline|bottomline|all** position where to print the label(s) when defined. When options happen to be contradictory, like **frame=topline** and **labelposition=bottomline**, nothing is displayed. Initially **none** when no labels are defined, **topline** for one label and **all** otherwise.

```
468 labelposition .choices:nn =
469   { none, topline, bottomline, all }
470   { \CDR_tag_choices_set: },
```

- **numbers=none|left|right** numbering of the verbatim lines. If requested, this numbering is done outside the verbatim environment. Initially **none**: no numbering.

```
471 numbers .choices:nn =
472   { none, left, right }
473   { \CDR_tag_choices_set: },
```

- **numbersep=<dimension>** gap between numbers and verbatim lines. Initially 12pt.

```
474 numbersep .code:n = \CDR_tag_set:,
475 numbersep .value_required:n = true,
```

- **firstnumber=auto|last|<integer>** number of the first line. **last** means that the numbering is continued from the previous verbatim environment. If an integer is given, its value will be used to start the numbering. Initially **auto**: numbering starts from 1.

```
476 firstnumber .code:n = {
477   \regex_match:nnTF { ^(+|-)?\d+$ } { #1 } { % $
478     \CDR_tag_set:
479   } {
480     \str_case:nnF { #1 } {
481       { auto } { \CDR_tag_set: }

```

```

482     { last } { \CDR_tag_set: }
483   } {
484     \PackageWarning
485     { CDR }
486     { Value-‘#1’~not~in~auto,~last. }
487   }
488 }
489 },
490 firstnumber .value_required:n = true,

```

● **stepnumber**=*<integer>* interval at which line numbers are printed. Initially 1: all lines are numbered.

```

491 stepnumber .code:n = \CDR_tag_set:,
492 stepnumber .value_required:n = true,

```

● **numberblanklines**=[*true|false*] to number or not the white lines (really empty or containing blank characters only). Initially *true*: all lines are numbered.

```

493 numberblanklines .code:n = \CDR_tag_boolean_set:x { #1 },

```

● **firstline**=*<integer>* first line to print. Initially empty: all lines from the first are printed.

```

494 firstline .code:n = \CDR_tag_set:,
495 firstline .value_required:n = true,

```

● **lastline**=*<integer>* last line to print. Initially empty: all lines until the last one are printed.

```

496 lastline .code:n = \CDR_tag_set:,
497 lastline .value_required:n = true,

```

● **baselinestretch**=*auto|<dimension>* value to give to the usual `\baselinestretch` L<sup>A</sup>T<sub>E</sub>X parameter. Initially *auto*: its current value just before the verbatim command.

```

498 baselinestretch .code:n = \CDR_tag_set:,
499 baselinestretch .value_required:n = true,

```

⊘ **commandchars**=*<three characters>* characters which define the character which starts a macro and marks the beginning and end of a group; thus lets us introduce escape sequences in verbatim code. Of course, it is better to choose special characters which are not used in the verbatim text. Private to `coder`, unavailable to users.

● **xleftmargin**=*<dimension>* indentation to add at the start of each line. Initially *Opt*: no left margin.

```

500 xleftmargin .code:n = \CDR_tag_set:,
501 xleftmargin .value_required:n = true,

```

● **xrightmargin**=*<dimension>* right margin to add after each line. Initially *Opt*: no right margin.

```

502 xrightmargin .code:n = \CDR_tag_set:,
503 xrightmargin .value_required:n = true,

```

🔴 **resetmargins[=true|false]** reset the left margin, which is useful if we are inside other indented environments. Initially **true**.

```

504 resetmargins .code:n = \CDR_tag_boolean_set:x { #1 },

```

🔴 **hfuzz=<dimension>** value to give to the T<sub>E</sub>X \hfuzz dimension for text to format. This can be used to avoid seeing some unimportant overfull box messages. Initially 2pt.

```

505 hfuzz .code:n = \CDR_tag_set:,
506 hfuzz .value_required:n = true,

```

🔴 **samepage[=true|false]** in very special circumstances, we may want to make sure that a verbatim environment is not broken, even if it does not fit on the current page. To avoid a page break, we can set the samepage parameter to **true**. Initially **false**.

```

507 samepage .code:n = \CDR_tag_boolean_set:x { #1 },

```

✅ **\_\_initialize** Initialization.

```

508 __initialize .meta:n = {
509   commentchar = ,
510   gobble = 0,
511   frame = none,
512   label = ,
513   labelposition = none,% auto?
514   numbers = left,
515   numbersep = \hspace{1ex},
516   firstnumber = auto,
517   stepnumber = 1,
518   numberblanklines = true,
519   firstline = ,
520   lastline = ,
521   baselinestretch = auto,
522   resetmargins = true,
523   xleftmargin = 0pt,
524   xrightmargin = 0pt,
525   hfuzz = 2pt,
526   samepage = false,
527 },
528 __initialize .value_forbidden:n = true,
529 }
530 \AtBeginDocument{
531   \CDR_tag_keys_set:nn { __fancyvrb.block } { __initialize }
532 }

```

#### 9.4.3 \_\_fancyvrb.all l3keys module

Options available when pygments is not used.

```

533 \CDR_tag_keys_define:nn { __fancyvrb.all } {

```



- **commandchars**=*<three characters>* characters that define the character that starts a macro and marks the beginning and end of a group; allows to introduce escape sequences in the verbatim code. Of course, it is better to choose special characters that are not used in the verbatim text! Initially **none**. Ignored in **pygments** mode.

```
534 commandchars .code:n = \CDR_tag_set:,
535 commandchars .value_required:n = true,
```

- **codes**=*<macro>* to specify catcode changes. For instance, this allows us to include formatted mathematics in verbatim text. Initially empty. Ignored in **pygments** mode.

```
536 codes .code:n = \CDR_tag_set:,
537 codes .value_required:n = true,
```

- ✓ **\_\_initialize** Initialization.

```
538 __initialize .meta:n = {
539     commandchars = ,
540     codes = ,
541 },
542 __initialize .value_forbidden:n = true,

543 }
544 \AtBeginDocument{
545     \CDR_tag_keys_set:nn { __fancyvrb.all } { __initialize }
546 }
```

## 10 \CDRSet

---

**\CDRSet**    \CDRSet {<key[=value] list>}  
              \CDRSet {only description=true, font family=tt}  
              \CDRSet {tag/default.code/font family=sf}

---

To set up the package. This is executed at least once at the end of the preamble. The unique mandatory argument of **\CDRSet** is a list of *<key>[=*<value>*]* items defined by the **CDR@Set l3keys** module.

### 10.1 CDR@Set l3keys module

```
547 \keys_define:nn { CDR@Set } {
```

- **only description** to typeset only the description section and ignore the implementation section.

```
548 only~description .choices:nn = { false, true, {} } {
549     \int_compare:nNnTF \l_keys_choice_int = 1 {
550         \prg_set_conditional:Nnn \CDR_if_only_description: { p, T, F, TF } { \prg_return_true: }
551     } {
552         \prg_set_conditional:Nnn \CDR_if_only_description: { p, T, F, TF } { \prg_return_false: }
553     }
554 },
555 only~description .initial:n = false

556 }
```

## 10.2 Branching

---

```
\CDR_if_only_description_p: * \CDR_if_only_description:TF {\true code}\{\false code}\}
\CDR_if_only_description:TF *
```

---

Execute `\true code` when only the description is expected, `\false code` otherwise.  
*Implementation detail:* the functions are defined as part of the `CDR@Set l3keys` module.

## 10.3 Implementation

---

```
\CDR_check_unknown:N \CDR_check_unknown:N {\tl variable}\}
```

---

In normal situation, the argument is expected to be empty. When the argument is not empty, send a package warning for each key.

```
557 \exp_args_generate:n { xV, nnV }
558 \cs_new:Npn \CDR_check_unknown:N #1 {
559   \tl_if_empty:NF #1 {
560     \cs_set:Npn \CDR_check_unknown:n ##1 {
561       \PackageWarning
562         { coder }
563         { Unknow~key~'##1' }
564     }
565     \cs_set:Npn \CDR_check_unknown:nn ##1 ##2 {
566       \CDR_check_unknown:n { ##1 }
567     }
568     \exp_args:NnnV
569     \keyval_parse:nnn {
570       \CDR_check_unknown:n
571     } {
572       \CDR_check_unknown:nn
573     } #1
574   }
575 }

576 \NewDocumentCommand \CDRSet { m } {
577   \CDR_keys_set_known:nnN { CDR@Set } { #1 } \l_CDR_keyval_tl
578   \clist_map_inline:nn {
579     __pygments, __pygments.block,
580     default.block, default.code, default,
581     __fancyvrb, __fancyvrb.block, __fancyvrb.all
582   } {
583     \CDR_tag_keys_set_known:nVN { ##1 } \l_CDR_keyval_tl \l_CDR_keyval_tl
584   }
585   \CDR_keys_set_known:VVN \c_CDR_Tags \l_CDR_keyval_tl \l_CDR_keyval_tl
586   \CDR_tag_provide_from_keyval:V \l_CDR_keyval_tl
587   \CDR_tag_keys_set_known:nVN { default } \l_CDR_keyval_tl \l_CDR_keyval_tl
588   \CDR_keys_set_known:VVN \c_CDR_Tags \l_CDR_keyval_tl \l_CDR_keyval_tl
589   \CDR_check_unknown:N \l_CDR_keyval_tl
590 }
```

## 11 \CDRExport

---

**\CDRExport**    \CDRExport {<key[=value] controls>}

---

The <key>[=<value>] controls are defined by CDR@Export l3keys module.

### 11.1 Storage

**\c\_CDR\_tag\_get**    Root identifier for tag properties, used throughout the package.  
**\c\_CDR\_slash**

```
591 \str_const:Nn \c_CDR_export_get { CDR@export@get }
      (End definition for \c_CDR_tag_get and \c_CDR_slash. These variables are documented on page ??.)
```

---

**\CDR\_export\_get\_path:cc** ★    \CDR\_tag\_export\_path:cc {<file name>} {<relative key path>}

---

Internal: return a unique key based on the arguments. Used to store and retrieve values.

```
592 \cs_new:Npn \CDR_export_get_path:cc #1 #2 {
593   \c_CDR_export_get @ #1 / #2 :
594 }
```

---

**\CDR\_export\_set:ccn**    \CDR\_export\_set:ccn {<file name>} {<relative key path>} {<value>}

---

**\CDR\_export\_set:Vcn**    Store <value>, which is further retrieved with the instruction \CDR\_get\_get:cc {<file name>} {<relative key path>}. All the affectations are made at the current T<sub>E</sub>X group level.  
**\CDR\_export\_set:VcV**

```
595 \cs_new_protected:Npn \CDR_export_set:ccn #1 #2 #3 {
596   \cs_set:cpn { \CDR_export_get_path:cc { #1 } { #2 } } { \exp_not:n { #3 } }
597 }
598 \cs_new_protected:Npn \CDR_export_set:Vcn #1 {
599   \exp_args:NV
600   \CDR_export_set:ccn { #1 }
601 }
602 \cs_new_protected:Npn \CDR_export_set:VcV #1 #2 #3 {
603   \exp_args:NVnV
604   \CDR_export_set:ccn #1 { #2 } #3
605 }
```

---

**\CDR\_export\_if\_exist:ccTF** ★    \CDR\_export\_if\_exist:ccTF {<file name>} {<relative key path>} {<true code>} {<false code>}

---

If the <relative key path> is known within <file name>, the <true code> is executed, otherwise, the <false code> is executed.

```
606 \prg_new_conditional:Nnn \CDR_export_if_exist:cc { p, T, F, TF } {
607   \cs_if_exist:cTF { \CDR_export_get_path:cc { #1 } { #2 } } {
608     \prg_return_true:
609   } {
610     \prg_return_false:
611   }
612 }
```

---

```
\CDR_export_get:cc ★ \CDR_export_get:cc {<file name>} {<relative key path>}
```

---

The property value stored for <file name> and <relative key path>.

```
613 \cs_new:Npn \CDR_export_get:cc #1 #2 {
614   \CDR_export_if_exist:ccT { #1 } { #2 } {
615     \use:c { \CDR_export_get_path:cc { #1 } { #2 } }
616   }
617 }
```

---

```
\CDR_export_get:ccNTF \CDR_export_get:ccNTF {<file name>} {<relative key path>}
<tl var> {<true code>} {<false code>}
```

---

Get the property value stored for <file name> and <relative key path>, copy it to <tl var>. Execute <true code> on success, <false code> otherwise.

```
618 \prg_new_protected_conditional:Nnn \CDR_export_get:ccN { T, F, TF } {
619   \CDR_export_if_exist:ccTF { #1 } { #2 } {
620     \tl_set:Nx #3 { \CDR_export_get:cc { #1 } { #2 } }
621     \prg_return_true:
622   } {
623     \prg_return_false:
624   }
625 }
```

`\g_CDR_export_prop` Global storage for <file name>=<file export info>

```
626 \prop_new:N \g_CDR_export_prop
```

*(End definition for \g\_CDR\_export\_prop. This variable is documented on page ??.)*

`\l_CDR_file_tl` Store the file name used for exportation, used as key in the above property list.

```
627 \tl_new:N \l_CDR_file_tl
```

*(End definition for \l\_CDR\_file\_tl. This variable is documented on page ??.)*

`\l_CDR_tags_clist` Used by CDR@Export l3keys module to temporarily store tags during the export declaration.

```
628 \clist_new:N \l_CDR_tags_clist
```

*(End definition for \l\_CDR\_tags\_clist. This variable is documented on page ??.)*

`\l_CDR_export_prop` Used by CDR@Export l3keys module to temporarily store properties. *Nota Bene*: nothing similar with `\g_CDR_export_prop` except the name.

```
629 \prop_new:N \l_CDR_export_prop
```

*(End definition for \l\_CDR\_export\_prop. This variable is documented on page ??.)*

## 11.2 CDR@Export l3keys module

No initial value is given for every key. An `__initialize` action will set the storage with proper initial values.

```
630 \keys_define:nn { CDR@Export } {
```

🔴 **file**=*<name>* the output file name, must be provided otherwise an error is raised.

```
631   file .tl_set:N = \l_CDR_file_tl,
632   file .value_required:n = true,
```

🔴 **tags**=*<tags comma list>* the list of tags. No exportation when this list is void. Initially empty.

```
633   tags .code:n = {
634     \clist_set:Nn \l_CDR_tags_clist { #1 }
635     \clist_remove_duplicates:N \l_CDR_tags_clist
636     \prop_put:NVV \l_CDR_prop \l_keys_key_str \l_CDR_tags_clist
637   },
638   tags .value_required:n = true,
```

🔴 **lang** one of the languages pygments is aware of. Initially `tex`.

```
639   lang .code:n = {
640     \prop_put:NVn \l_CDR_prop \l_keys_key_str { #1 }
641   },
642   lang .value_required:n = true,
```

🔴 **preamble** the added preamble. Initially empty.

```
643   preamble .code:n = {
644     \prop_put:NVn \l_CDR_prop \l_keys_key_str { #1 }
645   },
646   preamble .value_required:n = true,
```

🔴 **postamble** the added postamble. Initially empty.

```
647   postamble .code:n = {
648     \prop_put:NVn \l_CDR_prop \l_keys_key_str { #1 }
649   },
650   postamble .value_required:n = true,
```

🔴 **raw**[`=true|false`] true to remove any additional material, false otherwise. Initially false.

```
651   raw .choices:nn = { false, true, {} } {
652     \prop_put:NVx \l_CDR_prop \l_keys_key_str {
653       \int_compare:nNnTF
654         \l_keys_choice_int = 1 { false } { true }
655     }
656   },
```

✅ **\_\_initialize** Meta key to properly initialize all the variables.

```

657 __initialize .meta:n = {
658   __initialize_prop = #1,
659   file=,
660   tags=,
661   lang=tex,
662   preamble=,
663   postamble=,
664   raw=false,
665 },
666 __initialize .default:n = \l_CDR_prop,

```

✓ **\_\_initialize\_prop** Goody: properly initialize the local property storage.

```

667 __initialize_prop .code:n = \prop_clear:N #1,
668 __initialize_prop .default:n = \l_CDR_prop,
669 }

```

### 11.3 Implementation

```

670 \NewDocumentCommand \CDRExport { m } {
671   \keys_set:nn { CDR@Export } { __initialize }
672   \keys_set:nn { CDR@Export } { #1 }
673   \tl_if_empty:NTF \l_CDR_file_tl {
674     \PackageWarning
675       { coder }
676       { Missing~key~‘file’ }
677   } {
678     \CDR_export_set:VcV \l_CDR_file_tl { file } \l_CDR_file_tl
679     \prop_map_inline:Nn \l_CDR_prop {
680       \CDR_export_set:Vcn \l_CDR_file_tl { ##1 } { ##2 }
681     }

```

If a `lang` is given, forwards the declaration to all the tagged chunks.

```

682   \prop_get:NnNT \l_CDR_prop { tags } \l_CDR_tags_clist {
683     \exp_args:NV
684     \CDR_export_get:ccNT \l_CDR_file_tl { lang } \l_CDR_tl {
685       \clist_map_inline:Nn \l_CDR_tags_clist {
686         \CDR_tag_set:ccV { ##1 } { lang } \l_CDR_tl
687       }
688     }
689   }
690 }
691 }

```

Files are created at the end of the typesetting process.

```

692 \AddToHook { enddocument / end } {
693   \prop_map_inline:Nn \g_CDR_export_prop {
694     \tl_set:Nn \l_CDR_prop { #2 }
695     \str_set:Nx \l_CDR_str {
696       \prop_item:Nn \l_CDR_prop { file }
697     }
698     \lua_now:n { CDR:export_file('l_CDR_str') }

```

```

699   \clist_map_inline:nn {
700     tags, raw, preamble, postamble
701   } {
702     \str_set:Nx \l_CDR_str {
703       \prop_item:Nn \l_CDR_prop { ##1 }
704     }
705     \lua_now:n {
706       CDR:export_file_info('##1', '\l_CDR_str')
707     }
708   }
709   \lua_now:n { CDR:export_file_complete() }
710 }
711 }

```

## 12 Style

pygments, through `coder-tool.py`, creates style commands, but the storage is managed on the L<sup>A</sup>T<sub>E</sub>X side by `coder.sty`.

### 12.1 Storage

`\g_CDR_style_prop` Storage for styles, the keys are style names as understood by pygments.

```

712 \prop_new:N \l_CDR_style_prop

```

(End definition for `\g_CDR_style_prop`. This variable is documented on page ??.)

### 12.2 Managements

---

**`\CDR@StyleDefine`** `\CDR@StyleDefine {<style name>} {<style commands>}`  
 Store the `<style commands>` under `<style name>`.

```

713 \cs_new:Npn \CDR@StyleDefine {
714   \prop_put:Nnn \l_CDR_style_prop
715 }

```

## 13 Creating display engines

### 13.1 Utilities

---

<code>\CDR_code_engine:c</code>	★	<code>\CDR_code_engine:c {&lt;engine name&gt;}</code>
<code>\CDR_code_engine:V</code>	★	<code>\CDR_block_engine:c {&lt;engine name&gt;}</code>
<code>\CDR_block_engine:c</code>	★	<code>\CDR_code_engine:c</code> builds a command sequence name based on <code>&lt;engine name&gt;</code> .
<code>\CDR_block_engine:V</code>	★	<code>\CDR_block_engine:c</code> builds an environment name based on <code>&lt;engine name&gt;</code> .

---

```

716 \cs_new:Npn \CDR_code_engine:c #1 {
717   CDR@colored/code/#1:n
718 }
719 \cs_new:Npn \CDR_block_engine:c #1 {

```

```

720   CDR@colored/block/#1
721 }
722 \cs_new:Npn \CDR_code_engine:V {
723   \exp_args:NV \CDR_code_engine:c
724 }
725 \cs_new:Npn \CDR_block_engine:V {
726   \exp_args:NV \CDR_block_engine:c
727 }

```

`\l_CDR_engine_tl` Storage for an engine name.

```

728 \tl_new:N \l_CDR_engine_tl

```

(End definition for `\l_CDR_engine_tl`. This variable is documented on page ??.)

---

`\CDRGetOption` `\CDRGetOption {<relative key path>}`

---

Returns the value given to `\CDRCode` command or `CDRBlock` environment for the `<relative key path>`. This function is only available during `\CDRCode` execution and inside `CDRBlock` environment.

## 13.2 Implementation

---

<code>\CDRNewCodeEngine</code>	<code>\CDRNewCodeEngine {&lt;engine name&gt;}{&lt;engine body&gt;}</code>
<code>\CDRRenewCodeEngine</code>	<code>\CDRRenewCodeEngine{&lt;engine name&gt;}{&lt;engine body&gt;}</code>

---

`<engine name>` is a non void string, once expanded. The `<engine body>` is a list of instructions which may refer to the first argument as `#1`, which is the value given for key `<engine name>` engine options, and the second argument as `#2`, which is the colored code.

```

729 \NewDocumentCommand \CDRNewCodeEngine { mm } {
730   \exp_args:Nx
731   \tl_if_empty:nTF { #1 } {
732     \PackageWarning
733       { coder }
734       { The~engine~cannot~be~void. }
735   } {
736     \cs_new:cpn { \CDR_code_engine:c {#1} } ##1 {
737       \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
738       #2
739     }
740     \ignorespaces
741   }
742 }

743 \NewDocumentCommand \CDRRenewCodeEngine { mm } {
744   \exp_args:Nx
745   \tl_if_empty:nTF { #1 } {
746     \PackageWarning
747       { coder }
748       { The~engine~cannot~be~void. }
749     \use_none:n
750   } {

```



```

751 \cs_if_exist:cTF { \CDR_code_engine:c { #1 } } {
752   \cs_set:cpn { \CDR_code_engine:c { #1 } } ##1 {
753     \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
754     #2
755   }
756 } {
757   \PackageWarning
758   { coder }
759   { No~code~engine~#1.}
760 }
761 \ignorespaces
762 }
763 }

```

---

`\CDR_apply_code_engine:n`    `\CDR_apply_code_engine:n {<verbatim code>}`

Get the code engine and apply. When the code engine is not recognized, an error is raised.

```

764 \cs_set:Npn \CDR_apply_code_engine:n {
765   \tl_set:Nx \l_CDR_tl { \CDR_tag_get:c { engine } }
766   \CDR_if_code_engine:VTF \l_CDR_tl {
767     \use:c { \CDR_code_engine:V \l_CDR_tl }
768   } {
769     \PackageError
770     { coder }
771     { \l_CDR_tl\space code~engine~unknown,~replaced-by~'default' }
772     {See~\CDRNewCodeEngine~in~the~coder~manual}
773     \use:c { \CDR_code_engine:c { default } }
774   }
775 }

```

---

`\CDRNewBlockEngine`    `\CDRNewBlockEngine {<engine name>} {<begin instructions>} {<end instructions>}`  
`\CDRRenewBlockEngine`    `\CDRRenewBlockEngine {<engine name>} {<begin instructions>} {<end instructions>}`

Create a L<sup>A</sup>T<sub>E</sub>X environment uniquely named after `<engine name>`, which must be a non void string once expanded. The `<begin instructions>` and `<end instructions>` are list of instructions which may refer to the unique argument as `#1`, which is the value given to `CDRBlock` environment for key `<engine name>` engine options. Various options are available with the `\CDRGetOption` function. *Implementation detail:* the third argument is parsed by `\NewDocumentEnvironment`.

```

776 \NewDocumentCommand \CDRNewBlockEngine { mm } {
777   \NewDocumentEnvironment { \CDR_block_engine:c { #1 } } { m } {
778     \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
779     #2
780   }
781 }

782 \NewDocumentCommand \CDRRenewBlockEngine { mm } {
783   \tl_if_empty:nTF { #1 } {
784     \PackageWarning
785     { coder }

```

```

786     { The~engine~cannot~be~void. }
787     \use_none:n
788   } {
789     \RenewDocumentEnvironment { \CDR_block_engine:c { #1 } } { m } {
790       \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
791       #2
792     }
793   }
794 }

```

### 13.3 Conditionals

---

\CDR\_if\_code\_engine:cTF ★ \CDR\_if\_code\_engine:cTF {<engine name>} {<true code>} {<false code>}

---

If there exists a code engine with the given <engine name>, execute <true code>. Otherwise, execute <false code>.

```

795 \prg_new_conditional:Nnn \CDR_if_code_engine:c { p, T, F, TF } {
796   \cs_if_exist:cTF { \CDR_code_engine:c { #1 } } {
797     \prg_return_true:
798   } {
799     \prg_return_false:
800   }
801 }
802 \prg_new_conditional:Nnn \CDR_if_code_engine:V { p, T, F, TF } {
803   \cs_if_exist:cTF { \CDR_code_engine:V #1 } {
804     \prg_return_true:
805   } {
806     \prg_return_false:
807   }
808 }

```

---

\CDR\_has\_block\_engine:cTF ★ \CDR\_has\_block\_engine:c {<engine name>} {<true code>} {<false code>}

---

If there exists a block engine with the given <engine name>, execute <true code>, otherwise, execute <false code>.

```

809 \prg_new_conditional:Nnn \CDR_has_block_engine:c { p, T, F, TF } {
810   \cs_if_exist:cTF { \CDR_block_engine:c { #1 } } {
811     \prg_return_true:
812   } {
813     \prg_return_false:
814   }
815 }
816 \prg_new_conditional:Nnn \CDR_has_block_engine:V { p, T, F, TF } {
817   \cs_if_exist:cTF { \CDR_block_engine:V #1 } {
818     \prg_return_true:
819   } {
820     \prg_return_false:
821   }
822 }

```

## 13.4 Default code engine

The default code engine does nothing special and forwards its argument as is.

```
823 \CDRNewCodeEngine { default } { #1 }
```

## 13.5 Default block engine

The default block engine does nothing.

```
824 \CDRNewBlockEngine { default } { } { }
```

# 14 \CDRCode function

## 14.1 Storage

`\l_CDR_tag_tl` To store the tag given.


```
825 \tl_new:N \l_CDR_tag_tl
```

*(End definition for \l\_CDR\_tag\_tl. This variable is documented on page ??.)*


## 14.2 `\_CDR_tag / __code` **l3keys** module

This is the module used to parse the user interface of the `\CDRCode` command.

```
826 \CDR_tag_keys_define:nn { __code } {
```

 **tag=**`<name>` to use the settings of the already existing named tag to display.

```
827   tag .tl_set:N = \l_CDR_tag_tl,
828   tag .value_required:n = true,
```

 **\_\_initialize** initialize

```
829   __initialize .meta:n = {
830     tag = default,
831   },
832   __initialize .value_forbidden:n = true,
833 }
```

## 14.3 Implementation

---

`\CDRCode` `\CDRCode{<key[=value]>}<delimiter><code><same delimiter>`

---

```
834 \cs_new:Npn \CDR_tl_put_right_braced:Nn #1 #2 {
835   \tl_put_right:Nn #1 { { #2 } }
836 }
837 \cs_new:Npn \CDR_tl_put_left_braced:Nn #1 #2 {
838   \tl_put_left:Nn #1 { { #2 } }
839 }
840 \cs_new:Npn \CDR_brace_if_contains_comma:n #1 {
```

```

841 \tl_if_in:nnTF { #1 } { , } { { #1 } } { #1 }
842 }
843 \cs_generate_variant:Nn \CDR_brace_if_contains_comma:n { V }
844 \cs_new:Npn \CDR_feed_options_clist:N #1 {
845   \clist_map_inline:nn {
846     formatcom, fontfamily, fontsize, fontshape,
847     tabsize, defineactive, reflabel
848   } {
849     \CDR_tag_get:cN { ##1 } \l_CDR_tl
850     \tl_if_empty:NF \l_CDR_tl {
851       \tl_put_left:Nn #1 {
852         ##1 = \CDR_brace_if_contains_comma:V \l_CDR_tl,
853       }
854     }
855   }
856   \clist_map_inline:nn { showspaces, showtabs, obeytabs } {
857     \tl_put_left:Nx #1 { ##1 = \CDR_tag_get:cN { ##1 }, }
858   }
859 }
860 \cs_new:Npn \CDR_code:n #1 {
861   \CDR_tag_inherit:cx { __local } {
862     \tl_if_empty:NF \l_CDR_tag_tl { \l_CDR_tag_tl, }
863     __code, default.code, default, __pygments, __fancyvrb,
864   }
865   \clist_clear:N \l_CDR_options_clist
866   \CDR_feed_options_clist:N \l_CDR_options_clist
867   \CDR_if_truthy:eTF { \CDR_tag_get:c {pygments} } {
868     \PackageWarning
869       { coder }
870       { pygments~unsuported }
871     \DefineShortVerb { #1 }
872     \SaveVerb [
873       aftersave = {
874         \UndefineShortVerb { #1 }
875         \lua_now:n {CDR:highlight_code('FV@SV@CDR@Code')}
876         \group_end:
877       }
878     ] { CDR@Code }
879   } {
880     \DefineShortVerb { #1 }
881     \SaveVerb [
882       aftersave = {
883         \UndefineShortVerb { #1 }
884         \CDR_code_fvset:
885         \CDR_apply_code_engine:n { \UseVerb { CDR@Code } }
886         \group_end:
887       }
888     ] { CDR@Code } #1
889   }
890 }

```

---

`\CDR_to_lua:` `\CDR_to_lua:`

Retrieve info from the tree storage and forwards to lua.

```
891 \cs_new:Npn \CDR_to_lua: {  
892   \lua_now:n { CDR:options_reset() }  
893   \seq_map_inline:Nn \g_CDR_tag_path_seq {  
894     \CDR_tag_get:cNT { ##1 } \l_CDR_tl {  
895       \str_set:Nx \l_CDR_str { \l_CDR_tl }  
896       \lua_now:n { CDR:option_add('##1','l_CDR_str') }  
897     }  
898   }  
899 }
```

## 15 CDRBlock environment

`CDRBlock` `\begin{CDRBlock}{<key[=value] list>} ... \end{CDRBlock}`

### 15.1 Storage

`\l_CDR_block_prop`


```
900 \prop_new:N \l_CDR_block_prop
```

*(End definition for `\l_CDR_block_prop`. This variable is documented on page ??.)*


### 15.2 `__block l3keys` module

This module is used to parse the user interface of the `CDRBlock` environment.


```
901 \CDR_tag_keys_define:nn { __block } {
```

 **ignore**[=true|false] to ignore this code chunk.


```
902   ignore .code:n = \CDR_tag_boolean_set:x { #1 },  
903   ignore .default:n = true,
```

 **test**[=true|false] whether the chunk is a test,

```
904   test .code:n = \CDR_tag_boolean_set:x { #1 },  
905   test .default:n = true,
```

 **engine options**=<engine options> exact options forwarded to the engine. Normally, options are appended to the default ones, assuming a key-value interface.

```
906   engine-options .code:n = \CDR_tag_set:,  
907   engine options .default:n = true,
```

 **\_\_initialize** initialize

```

908 __initialize .meta:n = {
909     tags = ,
910     ignore = false,
911     test= false,
912 },
913 __initialize .value_forbidden:n = true,
914 }

```

## 15.3 Context

Inside the CDRBlock environments, some local variables are available:

● \l\_CDR\_tags\_clist

## 15.4 Implementation

We start by saving some fancyvrb macros that we further want to extend. The unique mandatory argument of these macros will eventually be recorded to be saved later on.

```

915 \clist_map_inline:nn { i, ii, iii, iv } {
916     \cs_set_eq:cc { CDR@ListProcessLine@ #1 } { FV@ListProcessLine@ #1 }
917 }
918 \cs_new:Npn \CDR_process_line:n #1 {
919     \str_set:Nn \l_CDR_str { #1 }
920     \lua_now:n {CDR:process_line('l_CDR_str')}
921 }

922 \cs_new:Npn \CDR_keys_inherit__:nnn #1 #2 #3 {
923     \keys_define:nn { #1 } { #2 .inherit:n = { #3 } }
924 }
925 \cs_new:Npn \CDR_keys_inherit:nnn #1 #2 #3 {
926     \tl_if_empty:nTF { #1 } {
927         \CDR_keys_inherit__:nnn { } { #2 } { #3 }
928     } {
929         \clist_set:Nn \l_CDR_clist { #3 }
930         \exp_args:Nnnx
931         \CDR_keys_inherit__:nnn { #1 } { #2 } {
932             #1 / \clist_use:Nn \l_CDR_clist { ,#1/ }
933         }
934     }
935 }
936 \cs_generate_variant:Nn \CDR_keys_inherit:nnn { VnV, Vnn }
937 \def\FVB@CDRBlock #1 {
938     \@bsphack
939     \group_begin:
940     \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
941         \prg_return_true:
942     }
943     \clist_set:Nn \l_tmpa_clist {
944         __block, default.block, default, __fancyvrb.block, __fancyvrb,
945     }
946     \CDR_keys_inherit:VnV \c_CDR_tag { __local } \l_tmpa_clist
947     \clist_map_inline:Nn \l_tmpa_clist {

```

```

948 \CDR_tag_keys_set:nn { ##1 } { __initialize }
949 }
950 \CDR_tag_keys_set_known:nnN { __local } { #1 } \l_CDR_tl

```

Get the list of tags and setup coder-util.lua for recording or highlighting.

```

951 \clist_if_empty:NT \l_CDR_tags_clist {
952 \CDR_tag_get:ccN { default.block } { tags } \l_CDR_tags_clist
953 \clist_if_empty:NT \l_CDR_tags_clist {
954 \PackageWarning
955 { coder }
956 { No~(default)~tags~provided. }
957 }
958 }
959 \lua_now:n { CDR:process_block_new('l_CDR_tags_clist') }

```

\l\_CDR\_bool is true iff one of the tags needs pygments.

```

960 \bool_set_false:N \l_CDR_bool
961 \clist_map_inline:Nn \l_CDR_tags_clist {
962 \CDR_if_truthy:eT { \CDR_tag_get:cc { ##1 } { pygments } } {
963 \clist_map_break:n { \bool_set_true:N \l_CDR_bool }
964 }
965 }
966 \bool_if:NF \l_CDR_bool {
967 \CDR_keys_inherit:Vnx \c_CDR_tag { __local } {
968 \c_CDR_tag / __fancyvrb.all
969 }
970 \CDR_tag_keys_set_known:nVN { __local } \l_CDR_tl \l_CDR_tl
971 }
972 \CDR_check_unknown:N \l_CDR_tl
973 \clist_set:Nx \l_CDR_clist {
974 __block, default.block, default, __fancyvrb.block, __fancyvrb
975 }
976 \bool_if:NF \l_CDR_bool {
977 \clist_put_right:Nx \l_CDR_clist { __fancyvrb.all }
978 }
979 \CDR_keys_inherit:VnV \c_CDR_tag_get { __local } \l_CDR_clist
980
981 \CDR_tag_get:cN {reflabel} \l_CDR_tl
982 \exp_args:NV \label \l_CDR_tl
983 ERROR \bool_if:nF { \clist_if_empty_p:n } {}
984 \clist_if_empty:NF \l_CDR_tags_clist {
985 \cs_map_inline:nn { i, ii, iii, iv } {
986 \cs_set:cpn { FV@ListProcessLine@ #####1 } ##1 {
987 \CDR_process_line:n { ##1 }
988 \use:c { CDR@ListProcessLine@ #####1 } { ##1 }
989 }
990 }
991 }
992 \CDR_tag_get:cNF { engine } \l_CDR_engine_tl {
993 \tl_set:Nn \l_CDR_engine_tl { default }
994 }
995 \CDR_tag_get:xNF { \l_CDR_engine_tl~engine~options } \l_CDR_tl {
996 \tl_clear:N \l_CDR_tl

```

```

997 }
998 \exp_args:NnV
999 \begin { \CDR_block_engine:V \l_CDR_engine_tl } \l_CDR_tl
1000 \FV@VerbatimBegin
1001 \FV@Scan
1002 }
1003 \def\FVE@CDRBlock{
1004 \FV@VerbatimEnd
1005 \end { \CDR_block_engine:V \l_CDR_engine_tl }
1006 \group_end:
1007 \@esphack
1008 }
1009 \DefineVerbatimEnvironment{CDRBlock}{CDRBlock}{}
1010

```

## 16 The CDR@Pyg@Verbatim environment

This is the environment wrapping the `pygments` generated code when in block mode. It is the sole content of the various `*.pyg.tex` files.

```

1011 \def\FVB@CDR@Pyg@Verbatim #1 {
1012 \group_begin:
1013 \FV@VerbatimBegin
1014 \FV@Scan
1015 }
1016 \def\FVE@CDR@Pyg@Verbatim{
1017 \FV@VerbatimEnd
1018 \group_end:
1019 }
1020 \DefineVerbatimEnvironment{CDR@Pyg@Verbatim}{CDR@Pyg@Verbatim}{}
1021

```

## 17 More

---

```

\CDR_if_record:TF ★ \CDR_if_record:TF {\true code} {\false code}

```

---

Execute `\true code` when code should be recorded, `\false code` otherwise. The code should be recorded for the `CDRBlock` environment when there is a non empty list of tags and `pygment` is used. *Implementation details*: we assume that if `\l_CDR_tags_clist` is not empty then we are in a `CDRBlock` environment.

```

1022 \prg_new_conditional:Nnn \CDR_if_record: { T, F, TF } {
1023 \clist_if_empty:NTF \l_CDR_tags_clist {
1024 \prg_return_false:
1025 } {
1026 \CDR_if_use_pygments:TF {
1027 \prg_return_true:
1028 } {
1029 \prg_return_false:
1030 }
1031 }
1032 }

```



```

1033 \cs_new:Npn \CDR_process_recordNO: {
1034   \tl_put_right:Nx \l_CDR_recorded_tl { \the\verbatim@line \iow_newline: }
1035   \group_begin:
1036   \tl_set:Nx \l_tmpa_tl { \the\verbatim@line }
1037   \lua_now:e {CDR.records.append([==[\l_tmpa_tl]==])}
1038   \group_end:
1039 }

```

CDR            \begin{<CDR>} ... \end{<CDR>}  
Private environment.

```

1040 \newenvironment{CDR}{
1041   \def \verbatim@processline {
1042     \group_begin:
1043     \CDR_processline_code_append:
1044     \group_end:
1045   }
1046   % \CDR_if_show_code:T {
1047   %   \CDR_if_use_minted:TF {
1048   %     \Needspace* { 2\baselineskip }
1049   %   } {
1050   %     \frenchspacing\@vobeyspaces
1051   %   }
1052   % }
1053 } {
1054   \CDR:nNTF { lang } \l_tmpa_tl {
1055     \tl_if_empty:NT \l_tmpa_tl {
1056       \clist_map_inline:Nn \l_CDR_clist {
1057         \CDR:nnNT { ##1 } { lang } \l_tmpa_tl {
1058           \tl_if_empty:NF \l_tmpa_tl {
1059             \clist_map_break:
1060           }
1061         }
1062       }
1063       \tl_if_empty:NT \l_tmpa_tl {
1064         \tl_set:Nn \l_tmpa_tl { tex }
1065       }
1066     }
1067   } {
1068     \tl_set:Nn \l_tmpa_tl { tex }
1069   }
1070 % NO WAY
1071   \clist_map_inline:Nn \l_CDR_clist {
1072     \CDR_gput:nnV { ##1 } { lang } \l_tmpa_tl
1073   }
1074 }

```

CDR.M            \begin{<CDR.M>} ... \end{<CDR.N>}  
Private environment when minted.

```

1075 \newenvironment{CDR_M}{
1076   \setkeys { FV } { firstnumber=last, }
1077   \clist_if_empty:NTF \l_CDR_clist {
1078     \exp_args:Nnx \setkeys { FV } {

```

```

1079     firstnumber=\CDR_int_use:n { },
1080 } } {
1081   \clist_map_inline:Nn \l_CDR_clist {
1082     \exp_args:Nnx \setkeys { FV } {
1083       firstnumber=\CDR_int_use:n { ##1 },
1084     }
1085     \clist_map_break:
1086   } }
1087   \iow_open:Nn \minted@code { \jobname.pyg }
1088   \tl_set:Nn \l_CDR_line_tl {
1089     \tl_set:Nx \l_tmpa_tl { \the\verbatim@line }
1090     \exp_args:NNV \iow_now:Nn \minted@code \l_tmpa_tl
1091   }
1092 } {
1093   \CDR_if_show_code:T {
1094     \CDR_if_use_minted:TF {
1095       \iow_close:N \minted@code
1096       \vspace* { \dimexpr -\topsep-\parskip }
1097       \tl_if_empty:NF \l_CDR_info_tl {
1098         \tl_use:N \l_CDR_info_tl
1099         \vspace* { \dimexpr -\topsep-\parskip-\baselineskip }
1100         \par\noindent
1101       }
1102       \exp_args:NV \minted@pygmentize \l_tmpa_tl
1103       \DeleteFile { \jobname.pyg }
1104       \vspace* { \dimexpr -\topsep -\partopsep }
1105     } {
1106       \@esphack
1107     }
1108   }
1109 }

```

CDR.P      \begin{<CDR.P> ... \end{<CDR.P>}  
Private pseudo environment. This is just a practical way of declaring balanced actions.

```

1110 \newenvironment{CDR_P}{
1111   \if_mode_vertical:
1112     \noindent
1113   \else
1114     \vspace*{ \topsep }
1115     \par\noindent
1116   \fi
1117   \CDR_gset_chunks:
1118   \tl_if_empty:NTF \g_CDR_chunks_tl {
1119     \CDR_if:nTF {show_lineno} {
1120       \CDR_if_use_margin:TF {

```

No chunk name, line numbers in the margin

```

1121     \tl_set:Nn \l_CDR_info_tl {
1122       \hbox_overlap_left:n {
1123         \CDR:n { format/code }
1124         {
1125           \CDR:n { format/name }

```

```

1126         \CDR:n { format/lineno }
1127         \clist_if_empty:NTF \l_CDR_clist {
1128             \CDR_int_use:n { }
1129         } {
1130             \clist_map_inline:Nn \l_CDR_clist {
1131                 \CDR_int_use:n { ##1 }
1132                 \clist_map_break:
1133             }
1134         }
1135     }
1136     \hspace*{1ex}
1137 }
1138 }
1139 } {

```

No chunk name, line numbers not in the margin

```

1140     \tl_set:Nn \l_CDR_info_tl {
1141     {
1142         \CDR:n { format/code }
1143         {
1144             \CDR:n { format/name }
1145             \CDR:n { format/lineno }
1146             \hspace*{3ex}
1147             \hbox_overlap_left:n {
1148                 \clist_if_empty:NTF \l_CDR_clist {
1149                     \CDR_int_use:n { }
1150                 } {
1151                     \clist_map_inline:Nn \l_CDR_clist {
1152                         \CDR_int_use:n { ##1 }
1153                         \clist_map_break:
1154                     }
1155                 }
1156             }
1157             \hspace*{1ex}
1158         }
1159     }
1160 }
1161 }
1162 } {

```

No chunk name, no line numbers

```

1163     \tl_clear:N \l_CDR_info_tl
1164 }
1165 } {
1166     \CDR_if:nTF {show_lineno} {

```

Chunk names, line numbers, in the margin

```

1167     \tl_set:Nn \l_CDR_info_tl {
1168         \hbox_overlap_left:n {
1169             \CDR:n { format/code }
1170             {
1171                 \CDR:n { format/name }
1172                 \g_CDR_chunks_tl :

```

```

1173         \hspace*{1ex}
1174         \CDR:n { format/lineno }
1175         \clist_map_inline:Nn \l_CDR_clist {
1176             \CDR_int_use:n { ####1 }
1177             \clist_map_break:
1178         }
1179     }
1180     \hspace*{1ex}
1181 }
1182 \tl_set:Nn \l_CDR_info_tl {
1183     \hbox_overlap_left:n {
1184         \CDR:n { format/code }
1185         {
1186             \CDR:n { format/name }
1187             \CDR:n { format/lineno }
1188             \clist_map_inline:Nn \l_CDR_clist {
1189                 \CDR_int_use:n { ####1 }
1190                 \clist_map_break:
1191             }
1192         }
1193     } \hspace*{1ex}
1194 }
1195 }
1196 }
1197 } {

```

Chunk names, no line numbers, in the margin

```

1198     \tl_set:Nn \l_CDR_info_tl {
1199         \hbox_overlap_left:n {
1200             \CDR:n { format/code }
1201             {
1202                 \CDR:n { format/name }
1203                 \g_CDR_chunks_tl :
1204             }
1205         } \hspace*{1ex}
1206     }
1207     \tl_clear:N \l_CDR_info_tl
1208 }
1209 }
1210 }
1211 \CDR_if_use_minted:F {
1212     \tl_set:Nn \l_CDR_line_tl {
1213         \noindent
1214         \hbox_to_wd:nn { \textwidth } {
1215             \tl_use:N \l_CDR_info_tl
1216             \CDR:n { format/code }
1217             \the\verbatim@line
1218             \hfill
1219         }
1220     } \par
1221 }
1222 \@bsphack
1223 }
1224 } {

```

```

1225 \vspace*{ \topsep }
1226 \par
1227 \@esphack
1228 }

```

## 18 Management

`\g_CDR_in_impl_bool` Whether we are currently in the implementation section.

```

1229 \bool_new:N \g_CDR_in_impl_bool
      (End definition for \g_CDR_in_impl_bool. This variable is documented on page ??.)

```

---

`\CDR_if_show_code:TF` `\CDR_if_show_code:TF`  $\{\langle true\ code\rangle\}$   $\{\langle false\ code\rangle\}$   
 Execute  $\langle true\ code\rangle$  when code should be printed,  $\langle false\ code\rangle$  otherwise.

```

1230 \prg_new_conditional:Nnn \CDR_if_show_code: { T, F, TF } {
1231   \bool_if:nTF {
1232     \g_CDR_in_impl_bool && !\g_CDR_with_impl_bool
1233   } {
1234     \prg_return_false:
1235   } {
1236     \prg_return_true:
1237   }
1238 }

```

`\g_CDR_with_impl_bool`

```

1239 \bool_new:N \g_CDR_with_impl_bool
      (End definition for \g_CDR_with_impl_bool. This variable is documented on page ??.)

```

## 19 minted and pygments

`\g_CDR_minted_on_bool` Whether minted is available, initially set to `false`.

```

1240 \bool_new:N \g_CDR_minted_on_bool
      (End definition for \g_CDR_minted_on_bool. This variable is documented on page ??.)

```

`\g_CDR_use_minted_bool` Whether minted is used, initially set to `false`.

```

1241 \bool_new:N \g_CDR_use_minted_bool
      (End definition for \g_CDR_use_minted_bool. This variable is documented on page ??.)

```

---

`\CDR_if_use_minted:TF` `\CDR_if_use_minted:TF`  $\{\langle true\ code\rangle\}$   $\{\langle false\ code\rangle\}$   
 Execute  $\langle true\ code\rangle$  when using minted,  $\langle false\ code\rangle$  otherwise.

```

1242 \prg_new_conditional:Nnn \CDR_if_use_minted: { T, F, TF } {
1243   \bool_if:NTF \g_CDR_use_minted_bool
1244     { \prg_return_true: }
1245     { \prg_return_false: }
1246 }

```

---

\\_CDR\_minted\_on:    \\_CDR\_minted\_on:

Private function. During the preamble, loads minted, sets \g\_CDR\_minted\_on\_bool to true and prepares pygments processing.

```

1247 \cs_set:Npn \_CDR_minted_on: {
1248   \bool_gset_true:N \g_CDR_minted_on_bool
1249   \RequirePackage{minted}
1250   \setkeys{ minted@opt@g } { linenos=false }
1251   \minted@def@opt{post-processor}
1252   \minted@def@opt{post-processor-args}
1253   \pretocmd\minted@inputpyg{
1254     \CDR@postprocesspyg {\minted@outputdir\minted@infile}
1255   }{\fail}

```

In the execution context of \minted@inputpyg,

#1 is the name of the python script, e.g., “process.py”

#2 is the input “.pygtex” file “\minted@outputdir\minted@infile”

#3 are more args passed to the python script, possibly empty

```

1256   \newcommand{\CDR@postprocesspyg}[1]{%
1257     \group_begin:
1258     \tl_set:Nx \l_tmpa_tl {\CDR:n { post_processor } }
1259     \tl_if_empty:NF \l_tmpa_tl {

```

Execute ‘python3 <script.py> <file.pygtex> <more\_args>’

```

1260       \tl_set:Nx \l_tmpb_tl {\CDR:n { post_processor_args } }
1261       \exp_args:Nx
1262       \sys_shell_now:n {
1263         python3\space
1264         \l_tmpa_tl\space
1265         ##1\space
1266         \l_tmpb_tl
1267       }
1268     }
1269     \group_end:
1270   }
1271 }

1272 %\AddToHook { begindocument / end } {
1273 %   \cs_set_eq:NN \_CDR_minted_on: \prg_do_nothing:
1274 %}

```

Utilities to setup pygment post processing. The pygment post processor marks some code with \CDREmph.

```

1275 \ProvideDocumentCommand{\CDREmph}{m}{\textcolor{red}{#1}}

```

---

\CDRPreamble    \CDRPreamble {<variable>} {<file name>}

Store the content of <file name> into the variable <variable>.

```

1276 \DeclareDocumentCommand \CDRPreamble { m m } {
1277   \msg_info:nnn
1278   { coder }
1279   { :n }
1280   { Reading-preamble-from-file-"#2". }
1281   \group_begin:
1282   \tl_set:Nn \l_tmpa_tl { #2 }
1283   \exp_args:NNNx
1284   \group_end:
1285   \tl_set:Nx #1 { \directlua{CDR.print_file_content('l_tmpa_tl')}} }
1286 }

```

## 20 Section separators

<hr/>	<hr/>
<code>\CDRImplementation</code>	<code>\CDRImplementation</code>
<code>\CDRFinale</code>	<code>\CDRFinale</code>
<hr/>	<hr/>
	<code>\CDRImplementation</code> start an implementation part where all the sectioning commands do nothing, whereas <code>\CDRFinale</code> stop an implementation part.

## 21 Finale

```

1287 \newcounter{CDR@impl@page}
1288 \DeclareDocumentCommand \CDRImplementation {} {
1289   \bool_if:NF \g_CDR_with_impl_bool {
1290     \clearpage
1291     \bool_gset_true:N \g_CDR_in_impl_bool
1292     \let\CDR@old@part\part
1293     \DeclareDocumentCommand\part{som}{}
1294     \let\CDR@old@section\section
1295     \DeclareDocumentCommand\section{som}{}
1296     \let\CDR@old@subsection\subsection
1297     \DeclareDocumentCommand\subsection{som}{}
1298     \let\CDR@old@subsubsection\subsubsection
1299     \DeclareDocumentCommand\subsubsection{som}{}
1300     \let\CDR@old@paragraph\paragraph
1301     \DeclareDocumentCommand\paragraph{som}{}
1302     \let\CDR@old@subparagraph\subparagraph
1303     \DeclareDocumentCommand\subparagraph{som}{}
1304     \cs_if_exist:NT \refsection{ \refsection }
1305     \setcounter{ CDR@impl@page }{ \value{page} }
1306   }
1307 }
1308 \DeclareDocumentCommand\CDRFinale {} {
1309   \bool_if:NF \g_CDR_with_impl_bool {
1310     \clearpage
1311     \bool_gset_false:N \g_CDR_in_impl_bool
1312     \let\part\CDR@old@part
1313     \let\section\CDR@old@section
1314     \let\subsection\CDR@old@subsection
1315     \let\subsubsection\CDR@old@subsubsection
1316     \let\paragraph\CDR@old@paragraph

```

```

1317 \let\subparagraph\CDR@old@subparagraph
1318 \setcounter { page } { \value{ CDR@impl@page } }
1319 }
1320 }
1321 \cs_set_eq:NN \CDR_line_number: \prg_do_nothing:

```

## 22 Finale

```

1322 \AddToHook { cmd/FancyVerbFormatLine/before } {
1323 \CDR_line_number:
1324 }
1325 \AddToHook { shipout/before } {
1326 \tl_gclear:N \g_CDR_chunks_tl
1327 }

1328 % =====
1329 % Auxiliary:
1330 % finding the widest string in a comma
1331 % separated list of strings delimited by parenthesis
1332 % =====
1333
1334 % arguments:
1335 % #1) text: a comma separated list of strings
1336 % #2) formatter: a macro to format each string
1337 % #3) dimension: will hold the result
1338
1339 \cs_new:Npn \CDRWidest (#1) #2 #3 {
1340 \group_begin:
1341 \dim_set:Nn #3 { Opt }
1342 \clist_map_inline:nn { #1 } {
1343 \hbox_set:Nn \l_tmpa_box { #2{##1} }
1344 \dim_set:Nn \l_tmpa_dim { \dim_eval:n { \box_wd:N \l_tmpa_box } }
1345 \dim_compare:nNnT { #3 } < { \l_tmpa_dim } {
1346 \dim_set_eq:NN #3 \l_tmpa_dim
1347 }
1348 }
1349 \exp_args:NNNV
1350 \group_end:
1351 \dim_set:Nn #3 #3
1352 }
1353 \ExplSyntaxOff
1354

```

## 23 pygmentex implementation

```

1355 % =====
1356 % fancyvrb new commands to append to a file
1357 % =====
1358
1359 % See http://tex.stackexchange.com/questions/47462/inputenc-error-with-unicode-chars-and-verbatim
1360
1361 \ExplSyntaxOn

```



```

1362
1363 \seq_new:N \l_CDR_records_seq
1364
1365 \long\def\unexpanded@write#1#2{\write#1{\unexpanded{#2}}}
1366
1367 \def\CDRAppend{\FV@Environment{}}{CDRAppend}}
1368
1369 \def\FVB@CDRAppend#1{%
1370   \@bsphack
1371   \begin{group}
1372     \seq_clear:N \l_CDR_records_seq
1373     \FV@UseKeyValues
1374     \FV@DefineWhiteSpace
1375     \def\FV@Space{\space}%
1376     \FV@DefineTabOut
1377     \def\FV@ProcessLine{%##1
1378       \seq_put_right:Nn \l_CDR_records_seq { ##1 }%
1379       \immediate\unexpanded@write#1%{##1}
1380     }%
1381     \let\FV@FontScanPrep\relax
1382     \let\@noligs\relax
1383     \FV@Scan
1384   }
1385 \def\FVE@CDRAppend{
1386   \seq_use:Nn \l_CDR_records_seq /
1387   \endgroup
1388   \@esphack
1389 }
1390 \DefineVerbatimEnvironment{CDRAppend}{CDRAppend}{}
1391
1392 \DeclareDocumentEnvironment { Inline } { m } {
1393   \clist_clear:N \l_CDR_clist
1394   \keys_set:nn { CDR_code } { #1 }
1395   \clist_map_inline:Nn \l_CDR_clist {
1396     \CDR_int_if_exist:nF { ##1 } {
1397       \CDR_int_new:nn { ##1 } { 1 }
1398       \seq_new:c { g/CDR/chunks/##1 }
1399     }
1400   }
1401   \CDR_if:nT {reset} {
1402     \CDR_clist_map_inline:Nnn \l_CDR_clist {
1403       \CDR_int_gset:nn { } 1
1404     } {
1405       \CDR_int_gset:nn { ##1 } 1
1406     }
1407   }
1408   \tl_clear:N \l_CDR_code_name_tl
1409   \clist_map_inline:Nn \l_CDR_clist {
1410     \prop_concat:ccc
1411       {g/CDR/Code/}
1412       {g/CDR/Code/##1/}
1413       {g/CDR/Code/}
1414     \tl_set:Nn \l_CDR_code_name_tl { ##1 }
1415     \clist_map_break:

```

```

1416 }
1417 \int_gset:Nn \g_CDR_int
1418 { \CDR_int_use:n { \l_CDR_code_name_tl } }
1419 \tl_clear:N \l_CDR_info_tl
1420 \tl_clear:N \l_CDR_name_tl
1421 \tl_clear:N \l_CDR_recorded_tl
1422 \tl_clear:N \l_CDR_chunks_tl
1423 \cs_set:Npn \verbatim@processline {
1424   \CDR_process_record:
1425 }
1426 \CDR_if_show_code:TF {
1427   \exp_args:NNx
1428   \skip_set:Nn \parskip { \CDR:n { parskip } }
1429   \clist_if_empty:NTF \l_CDR_clist {
1430     \tl_gclear:N \g_CDR_chunks_tl
1431   } {
1432     \clist_set_eq:NN \l_tmpa_clist \l_CDR_clist
1433     \clist_sort:Nn \l_tmpa_clist {
1434       \str_compare:nNnTF { ##1 } > { ##2 } {
1435         \sort_return_swapped:
1436       } {
1437         \sort_return_same:
1438       }
1439     }
1440     \tl_set:Nx \l_tmpa_tl { \clist_use:Nn \l_tmpa_clist , }
1441     \CDR_if:nT {show_name} {
1442       \CDR_if:nT {use_margin} {
1443         \CDR_if:nT {only_top} {
1444           \tl_if_eq:NNT \l_tmpa_tl \g_CDR_chunks_tl {
1445             \tl_gset_eq:NN \g_CDR_chunks_tl \l_tmpa_tl
1446             \tl_clear:N \l_tmpa_tl
1447           }
1448         }
1449         \tl_if_empty:NF \l_tmpa_tl {
1450           \tl_set:Nx \l_CDR_chunks_tl {
1451             \clist_use:Nn \l_CDR_clist ,
1452           }
1453           \tl_set:Nn \l_CDR_name_tl {
1454             {
1455               \CDR:n { format/name }
1456               \l_CDR_chunks_tl :
1457               \hspace*{1ex}
1458             }
1459           }
1460         }
1461       }
1462       \tl_if_empty:NF \l_tmpa_tl {
1463         \tl_gset_eq:NN \g_CDR_chunks_tl \l_tmpa_tl
1464       }
1465     }
1466   }
1467   \if_mode_vertical:
1468   \else:
1469   \par

```

```

1470 \fi:
1471 \vspace{ \CDR:n { sep } }
1472 \noindent
1473 \frenchspacing
1474 \@vobeyspaces
1475 \normalfont\ttfamily
1476 \CDR:n { format/code }
1477 \hyphenchar\font\m@ne
1478 \@noligs
1479 \CDR_if_record:F {
1480   \cs_set_eq:NN \CDR_process_record: \prg_do_nothing:
1481 }
1482 \CDR_if_use_minted:F {
1483   \CDR_if:nT {show_lineno} {
1484     \CDR_if:nTF {use_margin} {
1485       \tl_set:Nn \l_CDR_info_tl {
1486         \hbox_overlap_left:n {
1487           {
1488             \l_CDR_name_tl
1489             \CDR:n { format/name }
1490             \CDR:n { format/lineno }
1491             \int_use:N \g_CDR_int
1492             \int_gincr:N \g_CDR_int
1493           }
1494           \hspace*{1ex}
1495         }
1496       }
1497     } {
1498       \tl_set:Nn \l_CDR_info_tl {
1499         {
1500           \CDR:n { format/name }
1501           \CDR:n { format/lineno }
1502           \hspace*{3ex}
1503           \hbox_overlap_left:n {
1504             \int_use:N \g_CDR_int
1505             \int_gincr:N \g_CDR_int
1506           }
1507         }
1508         \hspace*{1ex}
1509       }
1510     }
1511   }
1512   \cs_set:Npn \verbatim@processline {
1513     \CDR_process_record:
1514     \hspace*{\dimexpr \linewidth-\columnwidth}%
1515     \hbox_to_wd:nn { \columnwidth } {
1516       \l_CDR_info_tl
1517       \the\verbatim@line
1518       \color{lightgray}\dotfill
1519     }
1520     \tl_clear:N \l_CDR_name_tl
1521   \par\noindent
1522 }
1523 }

```

```

1524 } {
1525   \@bsphack
1526 }
1527 \group_begin:
1528 \g_CDR_hook_tl
1529 \let \do \@makeother
1530 \dospecials \catcode '\^~M \active
1531 \verbatim@start
1532 } {
1533   \int_gsub:Nn \g_CDR_int {
1534     \CDR_int_use:n { \l_CDR_code_name_tl }
1535   }
1536   \int_compare:nNnT { \g_CDR_int } > { 0 } {
1537     \CDR_clist_map_inline:Nnn \l_CDR_clist {
1538       \CDR_int_gadd:nn { } { \g_CDR_int }
1539     } {
1540       \CDR_int_gadd:nn { ##1 } { \g_CDR_int }
1541     }
1542     \int_gincr:N \g_CDR_code_int
1543     \tl_set:Nx \l_tmpb_tl { \int_use:N \g_CDR_code_int }
1544     \clist_map_inline:Nn \l_CDR_clist {
1545       \seq_gput_right:cV { g/CDR/chunks/##1 } \l_tmpb_tl
1546     }
1547     \prop_gput:NVV \g_CDR_code_prop \l_tmpb_tl \l_CDR_recorded_tl
1548   }
1549   \group_end:
1550   \CDR_if_show_code:T {
1551   }
1552   \CDR_if_show_code:TF {
1553     \CDR_if_use_minted:TF {
1554       \tl_if_empty:NF \l_CDR_recorded_tl {
1555         \exp_args:Nnx \setkeys { FV } {
1556           firstnumber=\CDR_int_use:n { \l_CDR_code_name_tl },
1557         }
1558         \iow_open:Nn \minted@code { \jobname.pyg }
1559         \exp_args:NNV \iow_now:Nn \minted@code \l_CDR_recorded_tl
1560         \iow_close:N \minted@code
1561         \vspace* { \dimexpr -\topsep-\parskip }
1562         \tl_if_empty:NF \l_CDR_info_tl {
1563           \tl_use:N \l_CDR_info_tl
1564           \skip_vertical:n { \dimexpr -\topsep-\parskip-\baselineskip }
1565           \par\noindent
1566         }
1567         \exp_args:Nnx \minted@pygmentize { \jobname.pyg } { \CDR:n { lang } }
1568         %\DeleteFile { \jobname.pyg }
1569         \skip_vertical:n { -\topsep-\partopsep }
1570       }
1571     } {
1572       \exp_args:Nx \skip_vertical:n { \CDR:n { sep } }
1573       \noindent
1574     }
1575   } {
1576     \@esphack
1577   }

```

```

1578 }
1579 % =====
1580 % Main options
1581 % =====
1582
1583 \newif\ifCDR@left
1584 \newif\ifCDR@right
1585
1586

```

## 24 Display engines

Inserting code snippets follows one of two modes: run or block. The former is displayed as running text and used by the `\CDRCode` command whereas the latter is displayed as a separate block and used by the `CDRBlock` environment. Both have one single required argument, which is a *<key-value>* configuration list conforming to `CDR_code` `l3keys` module. The contents is then colored with the aid of `coder-tool.py` which will return some code enclosed within an environment created by one of `\CDRNewCodeEngine`, `\CDRRenewCodeEngine`, `\CDRNewBlockEngine`, `\CDRRenewBlockEngine` functions.

### 24.1 Run mode efbox engine

---

`CDRCallWithOptions` ★ `\CDRCallWithOptions<cs>`

---

Call *<cs>*, assuming it has a first optional argument. It will receive the arguments passed to `\CDRCode` with the `options` key.

```

1587 \cs_new:Npn \CDRCallWithOptions #1 {
1588   \exp_last_unbraced:NNx
1589   #1[\CDR:n { options }]
1590 }
1591 \CDRNewCodeEngine {efbox} {
1592   \CDRCallWithOptions\efbox{#1}%
1593 }

```

### 24.2 Block mode default engine

```

1594 \CDRNewBlockEngine {} {
1595 } {
1596 }

```

### 24.3 options key-value controls

We accept any value because we do not know in advance the real target. Everything is collected in `\l_CDR_options_clist`.

`\l_CDR_options_clist` All the *<key[=value] items>* passed as options are collected here. This should be cleared before arguments are parsed.

*(End definition for \l\_CDR\_options\_clist. This variable is documented on page ??.)*  
 There are 2 ways to collect options:

## 25 Something else

```

1597
1598 % =====
1599 % pygmented commands and environments
1600 % =====
1601
1602
1603 \newcommand\inputpygmented[2][{}]{%
1604   \begingroup
1605     \CDR@process@options{#1}%
1606     \immediate\write\CDR@outfile{<@@CDR@input@the\CDR@counter}%
1607     \immediate\write\CDR@outfile{\exp_args:NV\detokenize\CDR@global@options,\detokenize{#1}}%
1608     \immediate\write\CDR@outfile{#2}%
1609     \immediate\write\CDR@outfile{>@@CDR@input@the\CDR@counter}%
1610     %
1611     \csname CDR@snippet@the\CDR@counter\endcsname
1612     \global\advance\CDR@counter by 1\relax
1613   \endgroup
1614 }
1615
1616 \cs_generate_variant:Nn \exp_last_unbraced:NnNo { NxNo }
1617
1618 \newcommand\CDR@snippet@run[1]{%
1619   \group_begin:
1620   \typeout{DEBUG~PY~STYLE:< \CDR:n { style } > }
1621   \use_c:n { PYstyle }
1622   \CDR_when:nT { style } {
1623     \use_c:n { PYstyle \CDR:n { style } }
1624   }
1625   \cs_if_exist:cTF {PY} {PYOK} {PYKO}
1626   \CDR:n {font}
1627   \CDR@process@more@options{ \CDR:n {engine} }%
1628   \exp_last_unbraced:NxNo
1629   \use_c: { \CDR:n {engine} } [ \CDRRemainingOptions ]{#1}%
1630   \group_end:
1631 }
1632
1633 % ERROR: JL undefined \CDR@alllinenos
1634
1635 \ProvideDocumentCommand\captionof{mm}{-}{
1636 \def\CDR@alllinenos{(0)}
1637
1638 \def\FormatLineNumber#1{{\rmfamily\tiny#1}}
1639
1640 \newdimen\CDR@leftmargin
1641 \newdimen\CDR@linenosep
1642
1643 \def\CDR@lineno@do#1{%
1644   \CDR@linenosep Opt%
1645   \use_c: { CDR@ \CDR:n {block_engine} @margin }
1646   \exp_args:NNx
1647   \advance \CDR@linenosep { \CDR:n {linenosep} }
1648   \hbox_overlap_left:n {%

```

```

1649 \FormatLineNumber{#1}%
1650 \hspace*{\CDR@linenosep}%
1651 }%
1652 }
1653
1654 \newcommand\CDR@tcbbox@more@options{%
1655 nobeforeafter,%
1656 tcbbox~raise~base,%
1657 left=0mm,%
1658 right=0mm,%
1659 top=0mm,%
1660 bottom=0mm,%
1661 boxsep=2pt,%
1662 arc=1pt,%
1663 boxrule=0pt,%
1664 \CDR_options_if_in:nT {colback} {
1665 colback=\CDR:n {colback}
1666 }
1667 }
1668
1669 \newcommand\CDR@mdframed@more@options{%
1670 leftmargin=\CDR@leftmargin,%
1671 frametitlerule=true,%
1672 \CDR_if_in:nT {colback} {
1673 backgroundcolor=\CDR:n {colback}
1674 }
1675 }
1676
1677 \newcommand\CDR@tcolorbox@more@options{%
1678 grow~to~left~by=-\CDR@leftmargin,%
1679 \CDR_if_in:nNT {colback} {
1680 colback=\CDR:n {colback}
1681 }
1682 }
1683
1684 \newcommand\CDR@boite@more@options{%
1685 leftmargin=\CDR@leftmargin,%
1686 \ifcsname CDR@opt@colback\endcsname
1687 colback=\CDR@opt@colback,%
1688 \fi
1689 }
1690
1691 \newcommand\CDR@mdframed@margin{%
1692 \advance \CDR@linenosep \mdflength{outerlinewidth}%
1693 \advance \CDR@linenosep \mdflength{middlelinewidth}%
1694 \advance \CDR@linenosep \mdflength{innerlinewidth}%
1695 \advance \CDR@linenosep \mdflength{innerleftmargin}%
1696 }
1697
1698 \newcommand\CDR@tcolorbox@margin{%
1699 \advance \CDR@linenosep \kvtcb@left@rule
1700 \advance \CDR@linenosep \kvtcb@leftupper
1701 \advance \CDR@linenosep \kvtcb@boxsep
1702 }

```

```

1703
1704 \newcommand\CDR@boite@margin{%
1705   \advance \CDR@linenosep \boite@leftrule
1706   \advance \CDR@linenosep \boite@boxsep
1707 }
1708
1709 \def\CDR@global@options{}
1710
1711 \newcommand\setpygmented[1]{%
1712   \def\CDR@global@options{/CDR.cd,#1}%
1713 }
1714

```

## 26 Counters

---

<code>\CDR_int_new:nn</code>	<code>\CDR_int_new:n {&lt;name&gt;} {&lt;value&gt;}</code>
------------------------------	--

---

Create an integer after *<name>* and set it globally to *<value>*. *<name>* is a code name.

```

1715 \cs_new:Npn \CDR_int_new:nn #1 #2 {
1716   \int_new:c {g/CDR/int/#1}
1717   \int_gset:cn {g/CDR/int/#1} { #2 }
1718 }

```

---

<code>\CDR_int_set:nn</code>	<code>\CDR_int_set:n {&lt;name&gt;} {&lt;value&gt;}</code>
------------------------------	--

---

`\CDR_int_gset:nn` Set the integer named after *<name>* to the *<value>*. `\CDR_int_gset:n` makes a global change. *<name>* is a code name.

```

1719 \cs_new:Npn \CDR_int_set:nn #1 #2 {
1720   \int_set:cn {g/CDR/int/#1} { #2 }
1721 }
1722 \cs_new:Npn \CDR_int_gset:nn #1 #2 {
1723   \int_gset:cn {g/CDR/int/#1} { #2 }
1724 }

```

---

<code>\CDR_int_add:nn</code>	<code>\CDR_int_add:n {&lt;name&gt;} {&lt;value&gt;}</code>
------------------------------	--

---

`\CDR_int_gadd:nn` Add the *<value>* to the integer named after *<name>*. `\CDR_int_gadd:n` makes a global change. *<name>* is a code name.

```

1725 \cs_new:Npn \CDR_int_add:nn #1 #2 {
1726   \int_add:cn {g/CDR/int/#1} { #2 }
1727 }
1728 \cs_new:Npn \CDR_int_gadd:nn #1 #2 {
1729   \int_gadd:cn {g/CDR/int/#1} { #2 }
1730 }

```



---

`\CDR_int_sub:nn`    `\CDR_int_sub:n {<name>} {<value>}`  
`\CDR_int_gsub:nn`    Subtract the *<value>* from the integer named after *<name>*. `\CDR_int_gsub:n` makes a global change. *<name>* is a code name.

```

1731 \cs_new:Npn \CDR_int_sub:nn #1 #2 {
1732   \int_sub:cn {g/CDR/int/#1} { #2 }
1733 }
1734 \cs_new:Npn \CDR_int_gsub:nn #1 #2 {
1735   \int_gsub:cn {g/CDR/int/#1} { #2 }
1736 }

```

---

`\CDR_int_if_exist:nTF`    `\CDR_int_if_exist:nTF {<name>} {<true code>} {<false code>}`  
 Execute *<true code>* when an integer named after *<name>* exist, *<false code>* otherwise.

```

1737 \prg_new_conditional:Nnn \CDR_int_if_exist:n { T, F, TF } {
1738   \int_if_exist:cTF {g/CDR/int/#1} {
1739     \prg_return_true:
1740   } {
1741     \prg_return_false:
1742   }
1743 }

```

`\g/CDR/int/`    Generic and named line number counter. `\l_CDR_code_name_t` is used as *<name>*.  
`\g/CDR/int/<name>`  
 1744 `\CDR_int_new:nn {} { 1 }`  
*(End definition for \g/CDR/int/ and \g/CDR/int/<name>. These variables are documented on page ??.)*

---

`\CDR_int_use:n *`    `\CDR_int_use:n {<name>}`  
*<name>* is a code name.

```

1745 \cs_new:Npn \CDR_int_use:n #1 {
1746   \int_use:c {g/CDR/int/#1}
1747 }

1748 \ExplSyntaxOff

1749 %</sty>

```