

`coder` — code inlined in a \LaTeX document^{*}

Jérôme LAURENS[†]

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.pdf` gives different examples. Here is the implementation of the package.
This \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 and efficient syntax highlighting.

The `pygmentex` and `minted` packages were somehow a source of inspiration.

3 Known bugs and limitations

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

^{*}This file describes version 2022/02/07, last revised 2022/02/07.

[†]E-mail: jerome.laurens@u-bourgogne.fr

4 Namespace and conventions

L^AT_EX 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`.

The `c` argument specifier is used here in a more general acception. Normaly , it means that the argument is turned to a command sequence name. Here, it means that the argument is 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 L^AT_EX 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, block 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`, L^AT_EX 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 L^AT_EX coloring commands. These are stored in a `*.pyg.tex` file named after the md5 digest of the original code chunk, a `*.pyg.sty` L^AT_EX style file is recorded as well. On return, `coder-tool.py` gives to `coder-util.lua` some L^AT_EX 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 L^AT_EX 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^AT_EX 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 component, the *tag*, which is used for inheritance. All tags starting with two leading underscore characters are reserved by the package. Other tags are at the user disposal.

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**= \langle *font size* \rangle 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**= \langle *font shape* \rangle 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**= \langle *integer* \rangle number of spaces given by a tab character, Initially 2 (8 for **fancyvrb**).
- **defineactive**= \langle *macro* \rangle to define the effect of active characters. This allows to do some devious tricks, see the **fancyvrb** package. Initially empty.
- ✓ **relabel**= \langle *label* \rangle define a label to be used with **\pageref**. Initially empty.
- **commentchar**= \langle *character* \rangle lines starting with this character are ignored. Initially empty.
- **gobble**= \langle *integer* \rangle 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 \LaTeX **\fboxsep** macro is added between the left vertical line and the text. Initially **none**: no frame.
- **label**={ [**top string**] \langle *string* \rangle } 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 \langle *top string* \rangle is given between square brackets, it will be used for the top line and \langle *string* \rangle for the bottom line. Otherwise, \langle *string* \rangle 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**= \langle *dimension* \rangle gap between numbers and verbatim lines. Initially 12pt.

- **firstnumber=auto|last| $\langle integer \rangle$** 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= $\langle integer \rangle$** 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= $\langle integer \rangle$** first line to print. Initially empty: all lines from the first are printed.
- **lastline= $\langle integer \rangle$** last line to print. Initially empty: all lines until the last one are printed.
- **baselinestretch=auto| $\langle dimension \rangle$** value to give to the usual `\baselinestretch` L^AT_EX parameter. Initially **auto**: its current value just before the verbatim command.
- ⊘ **commandchars= $\langle three\ characters \rangle$** 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= $\langle dimension \rangle$** indentation to add at the start of each line. Initially **0pt**: no left margin.
- **xrightmargin= $\langle dimension \rangle$** 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= $\langle dimension \rangle$** value to give to the T_EX `\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= $\langle name \rangle$** 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, $\$ \dots \$$ 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** \LaTeX 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 \LaTeX 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 = kpse.find_file('coder-tool.py')
    (End definition for CDR_PY_PATH. This variable is documented on page ??.)
```

PYTHON_PATH Location of the python utility, defaults to 'python'.


```
11 local PYTHON_PATH = io.popen([[which python]]):read('a'):match("^%s*(.-%s*$")
    (End definition for PYTHON_PATH. This variable is documented on page ??.)
```

set_python_path CDR:set_python_path(*path var*)

 Set manually the path of the python utility with the contents of the *path var*. If the given path does not point to a file or a link then an error is raised.

```
12 local function set_python_path(self, path_var)
13   local path = assert(token.get_macro(assert(path_var)))
14   if #path>0 then
15     local mode,_,_ = lfs.attributes(self.PYTHON_PATH,'mode')
16     assert(mode == 'file' or mode == 'link')
17   else
18     path = io.popen([[which python]]):read('a'):match("^%s*(.-%s*$")
19   end
20   self.PYTHON_PATH = path
21 end
```

escape *variable* = CDR.escape(*string*)

 Escape the given string to be used by the shell.

```
22 local function escape(s)
23   s = s:gsub(' ','\\ ')
24   s = s:gsub('\\','\\\\')
25   s = s:gsub('\\r','\\r')
26   s = s:gsub('\\n','\\n')
27   s = s:gsub('"','\\"')
28   s = s:gsub("'",'"')
29   return s
30 end
```

make_directory *variable* = CDR.make_directory(*string path*)

Make a directory at the given path.

```
31 local function make_directory(path)
32   local mode,_,_ = lfs.attributes(path,"mode")
33   if mode == "directory" then
34     return true
35   elseif mode ~= nil then
36     return nil,path.." exist and is not a directory",1
```



```

37 end
38 if os["type"] == "windows" then
39     path = path:gsub("/", "\\")
40     _,_,__ = os.execute(
41         "if not exist " .. path .. "\\nul " .. "mkdir " .. path
42     )
43 else
44     _,_,__ = os.execute("mkdir -p " .. path)
45 end
46 mode = lfs.attributes(path,"mode")
47 if mode == "directory" then
48     return true
49 end
50 return nil,path.." exist and is not a directory",1
51 end

```

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

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

```

52 local dir_p, json_p
53 local jobname = tex.jobname
54 dir_p = './'..jobname..'pygd/'
55 if make_directory(dir_p) == nil then
56     dir_p = './'
57     json_p = dir_p..jobname..'pyg.json'
58 else
59     json_p = dir_p..'input.pyg.json'
60 end

```

print_file_content CDR.print_file_content($\langle \text{macro name} \rangle$)

The command named $\langle \text{macro name} \rangle$ contains the path to a file. Read the content of that file and print the result to the T_EX stream.

```

61 local function print_file_content(name)
62     local p = token.get_macro(name)
63     local fh = assert(io.open(p, 'r'))
64     s = fh:read('a')
65     fh:close()
66     tex.print(s)
67 end

```

load_exec CDR.load_exec($\langle \text{lua code chunk} \rangle$)

Class method. Loads the given $\langle \text{lua code chunk} \rangle$ and execute it. On error, messages are printed.

```

68 local function load_exec(chunk)
69   local func, err = load(chunk)
70   if func then
71     local ok, err = pcall(func)
72     if not ok then
73       print("coder-util.lua Execution error:", err)
74       print('chunk:', chunk)
75     end
76   else
77     print("coder-util.lua Compilation error:", err)
78     print('chunk:', chunk)
79   end
80 end

```

safe_equals $\langle \text{variable} \rangle = \text{safe_equals}(\langle \text{string} \rangle)$

Class method. Returns an $\langle =...= \rangle$ string as $\langle \text{ans} \rangle$ exactly composed of sufficiently many = signs such that $\langle \text{string} \rangle$ contains neither sequence $[\langle \text{ans} \rangle [\text{ nor }] \langle \text{ans} \rangle]$.

```

81 local eq_pattern = P({ Cp() * P('=')^1 * Cp() + P(1) * V(1) })
82 local function safe_equals(s)
83   local i, j = 0, 0
84   local max = 0
85   while true do
86     i, j = eq_pattern:match(s, j)
87     if i == nil then
88       return rep('=', max + 1)
89     end
90     i = j - i
91     if i > max then
92       max = i
93     end
94   end
95 end

```

load_exec_output $\text{CDR:load_exec_output}(\langle \text{lua code chunk} \rangle)$

Instance method to parse the $\langle \text{lua code chunk} \rangle$ string for commands and execute them. The patterns being searched are enclosed within opening <<<<< and closing >>>>>, each containing 5 characters,

?TEX: $\langle \text{TeX instructions} \rangle$ the $\langle \text{TeX instructions} \rangle$ are executed asynchronously once the control comes back to T_EX.

!LUA: $\langle \text{!Lua instructions} \rangle$ the $\langle \text{!Lua instructions} \rangle$ are executed synchronously. When not properly designed, these instructions may cause a forever loop on execution, for example, they must not use `CDR:if_code_engine`.

?LUA: $\langle \text{?Lua instructions} \rangle$ these $\langle \text{?Lua instructions} \rangle$ are executed asynchronously once the control comes back to T_EX through a call to `\directlua`, which means that they will wait until any previous asynchronous $\langle \text{?TeX instructions} \rangle$ or $\langle \text{?Lua instructions} \rangle$ completes.

```

96 local parse_pattern
97 do
98   local tag = P('!'') + '*' + '?'
99   local stp = '>>>>'
100  local cmd = (P(1) - stp)^0
101  parse_pattern = P({
102    P('<<<<') * Cg(tag) * 'LUA:' * Cg(cmd) * stp * Cp() + 1 * V(1)
103  })
104 end
105 local function load_exec_output(self, s)
106   local i, tag, cmd
107   i = 1
108   while true do
109     tag, cmd, i = parse_pattern:match(s, i)
110     if tag == '!' then
111       self.load_exec(cmd)
112     elseif tag == '*' then
113       local eqs = safe_equals(cmd)
114       cmd = '[' .. eqs .. '[' .. cmd .. ']' .. eqs .. ']'
115       tex.print([[
116 \directlua{CDR:load_exec[]..cmd..[]}%
117 ]])
118     elseif tag == '?' then
119       print('\nDEBUG/coder: ' .. cmd)
120     else
121       return
122     end
123   end
124 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:.

```

125 local function options_reset(self)
126   self['.options'] = {}
127 end

```

option_add CDR:option_add(<key>, <value name>)

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

```

128 local function option_add(self, key, value_name)
129   local p = self['.options']
130   p[key] = token.get_macro(assert(value_name))
131 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.

```
132 local function highlight_code_prepare(self)
133   self['.arguments'] = {
134     __cls__ = 'Arguments',
135     code = '',
136     cache = false,
137     debug = false,
138     pygopts = {
139       __cls__ = 'PygOpts',
140       lang = 'tex',
141       style = 'default',
142     },
143     texopts = {
144       __cls__ = 'TeXOpts',
145       already_style = false
146     }
147   }
148 end
149
150 local function highlight_set(self, key, value)
151   local args = self['.arguments']
152   local t = args
153   if t[key] == nil then
154     t = args.pygopts
155     if t[key] == nil then
156       t = args.texopts
157       assert(t[key] ~= nil)
158     end
159   end
160   t[key] = value
161 end
162
163 local function highlight_set_var(self, key, var)
164   self:highlight_set(key, assert(token.get_macro(var or 'l_CDR_tl')))
165 end
166
167 local function highlight_code(self)
168   local args = self['.arguments']
169   local json_p = self.json_p
170   local f = assert(io.open(json_p, 'w'))
171   local ok, err = f:write(json.tostring(args, true))
172   f:close()
173   if ok == nil then
174     print('File error('..json_p..'): '..err)
175   end
176 end
```

```

176 local cmd = ('%s %s %q'):format(
177     self.PYTHON_PATH,
178     self.CDR_PY_PATH,
179     json_p
180 )
181 local o = io.popen(cmd):read('a')
182 self:load_exec_output(o)
183 end

```

5.2 Block

highlight_block_prepare CDR:highlight_block_prepare(*<tags clist>*)

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

```

184 local function highlight_block_prepare(self, tags_clist)
185     local t = {}
186     for tag in string.gmatch(tags_clist, '([^\,]+)') do
187         t[#t+1]=tag
188     end
189     self['block tags'] = tags_clist
190     self['.lines'] = {}
191 end

```

process_line CDR:process_line(*<line variable name>*)

Store the content of the given named variable.

```

192 local function process_line(self, line_variable_name)
193     local line = assert(token.get_macro(assert(line_variable_name)))
194     local ll = self['.lines']
195     ll[#ll+1] = line
196     local lt = self['lines by tag'] or {}
197     self['lines by tag'] = lt
198     for tag in self['block tags']:gmatch('([^\,]+)') do
199         ll = lt[tag] or {}
200         lt[tag] = ll
201         ll[#ll+1] = line
202     end
203 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.

```

204 local function highlight_block(self, block_name)
205 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.

<u>export_file</u>	CDR:export_file(<i><file name var></i>)
--------------------	---

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

```
206 local function export_file(self, file_name)
207   self['.name'] = assert(token.get_macro(assert(file_name)))
208   self['.export'] = {}
209 end
```

<u>export_file_info</u>	CDR:export_file_info(<i><key></i> , <i><value name var></i>)
-------------------------	--

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

```
210 local function export_file_info(self, key, value)
211   local export = self['.export']
212   value = assert(token.get_macro(assert(value)))
213   export[key] = value
214 end
```

<u>export_complete</u>	CDR:export_complete()
------------------------	-----------------------

This is called at export time.

```
215 local function export_complete(self)
216   local name    = self['.name']
217   local export  = self['.export']
218   local records = self['.records']
219   local tt = {}
220   local s = export.preamble
221   if s then
222     tt[#tt+1] = s
223   end
224   for _,tag in ipairs(export.tags) do
225     s = records[tag]:concat('\n')
226     tt[#tt+1] = s
227     records[tag] = { [1] = s }
228   end
229   s = export.postamble
230   if s then
231     tt[#tt+1] = s
232   end
233   if #tt>0 then
234     local fh = assert(io.open(name,'w'))
235     fh:write(tt:concat('\n'))
236     fh:close()
```

```

237 end
238 self['.file'] = nil
239 self['.exportation'] = nil
240 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^AT_EX run and possibly between different L^AT_EX 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><style name.pyg.sty></i>, <i><digest.pyg.tex></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.

```

241 local function cache_clean_all(self)
242   local to_remove = {}
243   for f in lfs.dir(dir_p) do
244     to_remove[f] = true
245   end
246   for k,_ in pairs(to_remove) do
247     os.remove(dir_p .. k)
248   end
249 end
250 local function cache_record(self, style, colored)
251   self['.style_set'][style] = true
252   self['.colored_set'][colored] = true
253 end
254 local function cache_clean_unused(self)
255   local to_remove = {}
256   for f in lfs.dir(dir_p) do
257     if not self['.style_set'][f] and not self['.colored_set'][f] then
258       to_remove[f] = true
259     end
260   end
261   for k,_ in pairs(to_remove) do
262     os.remove(dir_p .. k)
263   end
264 end

```

`_DESCRIPTION` Short text description of the module.

```

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

```

8 Return the module

```
266 return {  
  
    Known fields are  
  
267     _DESCRIPTION      = _DESCRIPTION,  
  
    _VERSION to store <version string>,  
  
268     _VERSION          = token.get_macro('fileversion'),  
  
    date to store <date string>,  
  
269     date              = token.get_macro('filedate'),  
  
    Various paths ,  
  
270     CDR_PY_PATH       = CDR_PY_PATH,  
271     PYTHON_PATH       = PYTHON_PATH,  
272     set_python_path   = set_python_path,  
  
    escape  
  
273     escape            = escape,  
  
    make__directory  
  
274     make_directory    = make_directory,  
  
    load__exec  
  
275     load_exec         = load_exec,  
  
276     load_exec_output  = load_exec_output,  
  
    record__line  
  
277     record_line       = function(self,line) end,  
  
    highlight__code  
  
278     highlight_code_prepare = highlight_code_prepare,  
279     highlight_set         = highlight_set,  
280     highlight_set_var     = highlight_set_var,  
281     highlight_code        = highlight_code,  
  
    highlight__block_prepare, highlight__block  
  
282     highlight_block_prepare = highlight_block_prepare,  
283     highlight_block         = highlight_block,
```



```

cache__clean__all

284 cache_clean_all    = cache_clean_all,

cache__record

285 cache_record      = cache_record,

cache__clean__unused

286 cache_clean_unused = cache_clean_unused,

287 options_reset     = options_reset,

288 option_add         = option_add,

```

Internals

```

289 ['.style_set']    = {},
290 ['.colored_set']  = {},
291 ['.options']       = {},
292 ['.export']        = {},
293 ['.name']          = nil,

```

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

```

294 already            = false,

```

Other

```

295 json_p             = json_p,

296 }

297 %</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 import sys
11 import os
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 BaseOpts(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 __init__(self, d={}):
30         for k, v in d.items():
31             if type(v) == str:
32                 if v.lower() == 'true':
33                     setattr(self, k, True)
34                 elif v.lower() == 'false':
35                     setattr(self, k, False)
36                 continue
37             setattr(self, k, v)
```

3.1 TeXOptsclass

```
38 class TeXOpts(BaseOpts):
39     tags = ''
40     inline = True
41     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

```

42 sty_template=r'''% !TeX root=...
43 \makeatletter
44 \CDR@StyleDefine{<placeholder:style_name>}{%
45   <placeholder:style_defs>}%
46 \makeatother'''
47 code_template =r'''% !TeX root=...
48 \makeatletter
49 \CDR@StyleUse{<placeholder:style_name>}%
50 \CDR@CodeEngineApply{<placeholder:highlighted>}%
51 \makeatother'''
52
53 single_line_template='<placeholder:number><placeholder:line>'
54 first_line_template='<placeholder:number><placeholder:line>'
55 second_line_template='<placeholder:number><placeholder:line>'
56 white_line_template='<placeholder:number><placeholder:line>'
57 black_line_template='<placeholder:number><placeholder:line>'
58 block_template='<placeholder:count><placeholder:highlighted>'
59 def __init__(self, *args, **kwargs):
60     super().__init__(*args, **kwargs)
61     self.inline = self.ensure_bool(self.inline)

```

3.2 PygOptsclass

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.

```

62 class PygOpts(BaseOpts):
63     style = 'default'
64     nobackground = False
65     linenos = False
66     linenostart = 1
67     linenostep = 1
68     commandprefix = 'Py'
69     texcomments = False
70     mathescape = False
71     escapeinside = ""
72     envname = 'Verbatim'
73     lang = 'tex'
74     def __init__(self, *args, **kwargs):
75         super().__init__(*args, **kwargs)
76         self.linenos = self.ensure_bool(self.linenos)
77         self.linenostart = abs(int(self.linenostart))
78         self.linenostep = abs(int(self.linenostep))
79         self.texcomments = self.ensure_bool(self.texcomments)
80         self.mathescape = self.ensure_bool(self.mathescape)

```

3.3 FVclass

```

81 class FVOpts(BaseOpts):
82     gobble = 0
83     tabsize = 4
84     linenosep = '0pt'
85     commentchar = ''

```

```

86     frame = 'none'
87     label = ''
88     labelposition = 'none'
89     numbers = 'left'
90     numbersep = r'\hspace{1ex}'
91     firstnumber = 'auto'
92     stepnumber = 1
93     numberblanklines = True
94     firstline = ''
95     lastline = ''
96     baselinestretch = 'auto'
97     resetmargins = True
98     xleftmargin = 'Opt'
99     xrightmargin = 'Opt'
100    hfuzz = '2pt'
101    samepage = False
102    def __init__(self, *args, **kwargs):
103        super().__init__(*args, **kwargs)
104        self.gobble = abs(int(self.gobble))
105        self.tabsize = abs(int(self.tabsize))
106        if self.firstnumber != 'auto':
107            self.firstnumber = abs(int(self.firstnumber))
108        self.stepnumber = abs(int(self.stepnumber))
109        self.numberblanklines = self.ensure_bool(self.numberblanklines)
110        self.resetmargins = self.ensure_bool(self.resetmargins)
111        self.samepage = self.ensure_bool(self.samepage)

```

3.4 Argumentsclass

```

112 class Arguments(BaseOpts):
113     cache = False
114     debug = False
115     code = ""
116     style = "default"
117     json = ""
118     directory = "."
119     texopts = TeXOpts()
120     pygopts = PygOpts()
121     fv_opts = FVOpts()
122     directory = ""

```

4 Controller main class

```

123 class Controller:

```

4.1 Static methods

object_hook Helper for json parsing.

```

124     @staticmethod
125     def object_hook(d):

```

```

126     __cls__ = d.get('__cls__', 'Arguments')
127     print('HOOK __cls__', __cls__, d.get('code', 'FAILED'))
128     if __cls__ == 'PygOpts':
129         return PygOpts(d)
130     elif __cls__ == 'FV0pts':
131         return FV0pts(d)
132     elif __cls__ == 'TeX0pts':
133         return TeX0pts(d)
134     else:
135         return Arguments(d)

```

lua_command lua_command_now lua_debug	self.lua_command(<i>(asynchronous lua command)</i>) self.lua_command_now(<i>(synchronous lua command)</i>) Wraps the given command between markers. It will be in the output of the <code>coder-tool.py</code> , further captured by <code>coder-util.lua</code> and either forwarded to \TeX or executed synchronously.
--	---

```

136     @staticmethod
137     def lua_command(cmd):
138         print(f'<<<<*LUA:{cmd}>>>>')
139     @staticmethod
140     def lua_command_now(cmd):
141         print(f'<<<<!LUA:{cmd}>>>>')
142     @staticmethod
143     def lua_debug(msg):
144         print(f'<<<<?LUA:{msg}>>>>')

```

lua_text_escape	self.lua_text_escape(<i>(text)</i>) Wraps the given command between [=...= and]=...=] with as many equal signs as necessary to ensure a correct lua syntax.
------------------------	---

```

145     @staticmethod
146     def lua_text_escape(s):
147         k = 0
148         for m in re.findall('+=', s):
149             if len(m) > k: k = len(m)
150         k = (k + 1) * "="
151         return f'[{k}][{s}]{k}']

```

4.2 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 ??.)

```

152     _json_p = None
153     @property
154     def json_p(self):
155         p = self._json_p
156         if p:
157             return p
158         else:

```

```

159     p = self.arguments.json
160     if p:
161         p = Path(p).resolve()
162     self._json_p = p
163     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 ??.)

```

164     _pygd_p = None
165     @property
166     def pygd_p(self):
167         p = self._pygd_p
168         if p:
169             return p
170         p = self.arguments.directory
171         if p:
172             p = Path(p)
173         else:
174             p = self.json_p
175             if p:
176                 p = p.parent
177             else:
178                 p = Path('SHARED')
179         if p:
180             p = p.resolve().with_suffix(".pygd")
181             p.mkdir(exist_ok=True)
182         self._pygd_p = p
183         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 ??.)

```

184     @property
185     def pyg_sty_p(self):
186         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 ??.)

```

187     @property
188     def parser(self):
189         parser = argparse.ArgumentParser(
190             prog=sys.argv[0],
191             description='''
192 Writes to the output file a set of LaTeX macros describing
193 the syntax highlighting of the input file as given by pygments.
194 ''')
195     )
196     parser.add_argument(

```

```

197     "-v", "--version",
198     help="Print the version and exit",
199     action='version',
200     version=f'coder-tool version {__version__},
201     ' (c) {__YEAR__} by Jérôme LAURENS.'
202 )
203 parser.add_argument(
204     "--debug",
205     action='store_true',
206     default=None,
207     help="display informations useful for debugging"
208 )
209 parser.add_argument(
210     "json",
211     metavar="<json data file>",
212     help=""
213     file name with extension, contains processing information
214     ""
215 )
216 return parser
217

```

4.3 Methods

4.3.1 __init__

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

```

218 def __init__(self, argv = sys.argv):
219     argv = argv[1:] if re.match(".*coder\-\tool\.py$", argv[0]) else argv
220     ns = self.parser.parse_args(
221         argv if len(argv) else ['-h']
222     )
223     with open(ns.json, 'r') as f:
224         self.arguments = json.load(
225             f,
226             object_hook = Controller.object_hook
227         )
228     args = self.arguments
229     args.json = ns.json
230     texopts = self.texopts = args.texopts
231     pygopts = self.pygopts = args.pygopts
232     fv_opts = self.fv_opts = args.fv_opts
233     formatter = self.formatter = LatexFormatter(
234         style = pygopts.style,
235         nobackground = pygopts.nobackground,
236         commandprefix = pygopts.commandprefix,
237         texcomments = pygopts.texcomments,
238         mathescape = pygopts.mathescape,
239         escapeinside = pygopts.escapeinside,
240         envname = 'CDR@Pyg@Verbatim',
241     )

```

```

242
243     try:
244         lexer = self.lexer = get_lexer_by_name(pygopts.lang)
245     except ClassNotFound as err:
246         sys.stderr.write('Error: ')
247         sys.stderr.write(str(err))
248
249     escapeinside = pygopts.escapeinside
250     # When using the LaTeX formatter and the option 'escapeinside' is
251     # specified, we need a special lexer which collects escaped text
252     # before running the chosen language lexer.
253     if len(escapeinside) == 2:
254         left = escapeinside[0]
255         right = escapeinside[1]
256         lexer = self.lexer = LatexEmbeddedLexer(left, right, lexer)
257
258     gobble = fv_opts.gobble
259     if gobble:
260         lexer.add_filter('gobble', n=gobble)
261     tabsize = fv_opts.tabsize
262     if tabsize:
263         lexer.tabsize = tabsize
264     lexer.encoding = ''
265

```

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.

```

266     def get_pyg_tex_p(self, digest):
267         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.

```

268     def create_style(self):
269         pyg_sty_p = self.pyg_sty_p
270         if self.arguments.cache and pyg_sty_p.exists():
271             if self.arguments.debug:
272                 self.lua_debug(f'Style already available: {os.path.relpath(pyg_sty_p)}')
273             return
274         texopts = self.texopts
275         style = self.pygopts.style
276         if texopts.already_style:
277             if self.arguments.debug:
278                 self.lua_debug(f'Style already available: {style}')
279             return

```



```

280     formatter = self.formatter
281     style_defs = formatter.get_style_defs() \
282         .replace(r'\makeatletter', '') \
283         .replace(r'\makeatother', '') \
284         .replace('\n', '%\n')
285     sty = self.texopts.sty_template.replace(
286         '<placeholder:style_name>',
287         style,
288     ).replace(
289         '<placeholder:style_defs>',
290         style_defs,
291     ).replace(
292         '{}%',
293         '%[\n]%',
294     ).replace(
295         '[]%',
296         '%[\n]%',
297     ).replace(
298         '{}%',
299         '%[\n]%',
300     )
301     with pyg_sty_p.open(mode='w', encoding='utf-8') as f:
302         f.write(sty)
303     cmd = rf'\input{{.{os.path.relpath(pyg_sty_p)}}}%'
304     self.lua_command_now(
305         rf'tex.print({self.lua_text_escape(cmd)})'
306     )

```

4.3.4 pygmentize

self.pygmentize `<code variable> = self.pygmentize(<code>[, inline=<yorn>])`

Where the `<code>` is highlighted by pygments.

```

307     def pygmentize(self, code):
308         code = highlight(code, self.lexer, self.formatter)
309         m = re.match(
310             r'\\begin{CDR@Pyg@Verbatim}.*?\n(.*)\n\\end{CDR@Pyg@Verbatim}\s*\Z',
311             code,
312             flags=re.S
313         )
314         assert(m)
315         highlighted = m.group(1)
316         texopts = self.texopts
317         if texopts.inline:
318             return texopts.code_template.replace(
319                 '<placeholder:highlighted>', highlighted
320             ).replace(
321                 '<placeholder:style_name>', self.pygopts.style
322             )
323         fv_opts = self.fv_opts
324         lines = highlighted.split('\n')
325         number = fv_opts.firstnumber
326         stepnumber = fv_opts.stepnumber

```

```

327     no = ''
328     numbering = fv_opts.numbers != 'none'
329     ans_code = []
330     def more(template):
331         ans_code.append(template.replace(
332             '<placeholder:number>', f'{number}',
333             ).replace(
334                 '<placeholder:line>', line,
335             ))
336         number += 1
337     if len(lines) == 1:
338         line = lines.pop(0)
339         more(texopts.single_line_template)
340     elif len(lines):
341         line = lines.pop(0)
342         more(texopts.first_line_template)
343         line = lines.pop(0)
344         more(texopts.second_line_template)
345         if stepnumber < 2:
346             def template():
347                 return texopts.black_line_template
348         elif stepnumber % 5 == 0:
349             def template():
350                 return texopts.black_line_template if number %\
351                     stepnumber == 0 else texopts.white_line_template
352         else:
353             def template():
354                 return texopts.black_line_template if (number - firstnumber) %\
355                     stepnumber == 0 else texopts.white_line_template
356
357     for line in lines:
358         more(template())
359
360     hilighted = '\n'.join(ans_code)
361     return texopts.block_template.replace(
362         '<placeholder:count>', f'{number-firstnumber}'
363     ).replace(
364         '<placeholder:hilighted>', hilighted
365     )
366     %%%
367     %%%     ans_code.append(fr'''%
368 %%%\begin{{CDR@Block/engine/{pygopts.style}}}
369 %%%\CDRBlock@linenos@used:n {{{','.join(numbers)}}}%
370 %%%{m.group(1)}{'\n'.join(lines)}{m.group(3)}%
371 %%%\end{{CDR@Block/engine/{pygopts.style}}}
372 %%%''' )
373     %%%     ans_code = "".join(ans_code)
374     %%%     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.

```

375 def create_pygmented(self):
376     arguments = self.arguments
377     code = arguments.code
378     if not code:
379         return False
380     inline = self.texopts.inline
381     h = hashlib.md5(f'{str(code)}:{inline}'.encode('utf-8'))
382     pyg_tex_p = self.get_pyg_tex_p(h.hexdigest())
383     cmd = rf'\input{{.{os.path.relpath(pyg_tex_p)}}}%
384     if self.arguments.cache and pyg_tex_p.exists():
385         print("Already available:", pyg_tex_p)
386         self.lua_command_now(
387             rf'tex.print({self.lua_text_escape(cmd)})'
388         )
389         return True
390     code = self.pygmentize(code)
391     with pyg_tex_p.open(mode='w',encoding='utf-8') as f:
392         f.write(code)
393     self.lua_command_now(
394         rf'tex.print({self.lua_text_escape(cmd)})'
395     )
396     # \CDR_remove:n {{colored:}}%
397     # \input {{ \tl_to_str:n {{}} }}%
398     # \CDR:n {{colored:}}%
399     pyg_sty_p = self.pyg_sty_p
400     if pyg_sty_p.parent.stem != 'SHARED':
401         self.lua_command_now( f'''CDR:cache_record(
402     {self.lua_text_escape(pyg_sty_p.name)},
403     {self.lua_text_escape(pyg_tex_p.name)}
404 )''' )
405     print("PREMATURE EXIT")
406     exit(1)

```

4.4 Main entry

```

407 if __name__ == '__main__':
408     try:
409         ctrl = Controller()
410         x = ctrl.create_style() or ctrl.create_pygmented()
411         print(f'{sys.argv[0]}: done')
412         sys.exit(x)
413     except KeyboardInterrupt:
414         sys.exit(1)
415 %</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 {<tag name>} {<relative key path>} {<value>}</code> Store <i><value></i> , which is further retrieved with the instruction <code>\CDR_tag_get:cc {<tag name>} {<relative key path>}</code> . Only <i><tag name></i> and <i><relative key path></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 { /([~/*])/(.*)$ } \use_none:n { $ }
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 {<value>}</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 `<dir>`, `<relative key path>` and `<value>` are provided. The latter is guessed from `\l_keys_value_tl`, and `CDR_tag_set:n` is called. This is meant to be call from `\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 {{key path}} {{value}}`

When the last component of `\l_keys_path_str` should not be used to store the `<value>`, but `<key path>` should be used instead. This last component is replaced and `\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:cn
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 `\l_keys_path_str` is set properly. This is where a syntax like `\keys_set:nn {...} { choice/a }` is managed.

```
96 \regex_const:Nn \c_CDR_root_regex { ^(.*)/.*$ } \use_none:n { $ }
97 \cs_new:Npn \CDR_tag_choices: {
98   \exp_args:NVV
99   \str_if_eq:nnT \l_keys_key_tl \l_keys_choice_tl {
100     \exp_args:NnV
101     \regex_extract_once:NnNT \c_CDR_root_regex
102       \l_keys_path_str \l_CDR_seq {
103       \str_set:Nx \l_keys_path_str {
104         \seq_item:Nn \l_CDR_seq 2
105       }
106     }
107   }
108 }
```

`\CDR_tag_choices_set:` `\CDR_tag_choices_set:`

Calls `\CDR_tag_set:n` with the content of `\l_keys_choice_tl` as value. Before, ensure that the `\l_keys_path_str` is set properly.

```

109 \cs_new:Npn \CDR_tag_choices_set: {
110   \CDR_tag_choices:
111   \exp_args:NV
112   \CDR_tag_set:n \l_keys_choice_tl
113 }

```

`\CDR_if_truthy:nTF` `\CDR_if_truthy:nTF {<token list>} {<true code>} {<false code>}`
`\CDR_if_truthy:eTF`

Execute `<true code>` when `<token list>` is a truthy value, `<false code>` otherwise. A truthy value is a text which leading character, if any, is none of “fFnN”.

```

114 \prg_new_conditional:Nnn \CDR_if_truthy:n { p, T, F, TF } {
115   \exp_args:Nf
116   \str_compare:nNnTF { \str_lowercase:n { #1 } } = { false } {
117     \prg_return_false:
118   } {
119     \prg_return_true:
120   }
121 }
122 \prg_generate_conditional_variant:Nnn \CDR_if_truthy:n { e } { p, T, F, TF }

```

`\CDR_tag_boolean_set:n` `\CDR_tag_boolean_set:n {<choice>}`

Calls `\CDR_tag_set:n` with true if the argument is truthy, false otherwise.

```

123 \cs_new_protected:Npn \CDR_tag_boolean_set:n #1 {
124   \CDR_if_truthy:nTF { #1 } {
125     \CDR_tag_set:n { true }
126   } {
127     \CDR_tag_set:n { false }
128   }
129 }
130 \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 `\c_CDR_tag_get/<tag name>/<relative key path>`. When typesetting some code with either the `\CDRCode` command or the `CDRBlock` environment, all properties defined locally are collected under the reserved `\c_CDR_tag_get/__local/<relative path>` full key paths. The `l3keys` module `\c_CDR_tag_get/__local` is modified in \TeX groups only. For running text code chunks, this module inherits from

1. `\c_CDR_tag_get/<tag name>` for the provided `<tag name>`,
2. `\c_CDR_tag_get/default.code`
3. `\c_CDR_tag_get/default`

4. \c_CDR_tag_get/__pygments
5. \c_CDR_tag_get/__fancyvrb
6. \c_CDR_tag_get/__fancyvrb.all when no using pygments

For text block code chunks, this module inherits from

1. \c_CDR_tag_get/⟨name₁⟩, ..., \c_CDR_tag_get/⟨name_n⟩ 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.

```
131 \prg_new_conditional:Nnn \CDR_tag_if_exist_here:cc { T, F, TF } {
132   \cs_if_exist:cTF { \CDR_tag_get_path:cc { #1 } { #2 } } {
133     \prg_return_true:
134   } {
135     \prg_return_false:
136   }
137 }
```

```
\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.

```
138 \prg_new_conditional:Nnn \CDR_tag_if_exist:cc { T, F, TF } {
139   \cs_if_exist:cTF { \CDR_tag_get_path:cc { #1 } { #2 } } {
140     \prg_return_true:
141   } {
142     \seq_if_exist:cTF { \CDR_tag_parent_seq:c { #1 } } {
143       \seq_map_tokens:cn
144         { \CDR_tag_parent_seq:c { #1 } }
145         { \CDR_tag_if_exist_f:cn { #2 } }
146     } {
```

```

147     \prg_return_false:
148   }
149 }
150 }
151 \cs_new:Npn \CDR_tag_if_exist_f:cn #1 #2 {
152   \quark_if_no_value:nTF { #2 } {
153     \seq_map_break:n {
154       \prg_return_false:
155     }
156   } {
157     \CDR_tag_if_exist:ccT { #2 } { #1 } {
158       \seq_map_break:n {
159         \prg_return_true:
160       }
161     }
162   }
163 }

```

\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.

```

164 \cs_new:Npn \CDR_tag_get:cc #1 #2 {
165   \CDR_tag_if_exist_here:ccTF { #1 } { #2 } {
166     \use:c { \CDR_tag_get_path:cc { #1 } { #2 } }
167   } {
168     \seq_if_exist:cT { \CDR_tag_parent_seq:c { #1 } } {
169       \seq_map_tokens:cn
170       { \CDR_tag_parent_seq:c { #1 } }
171       { \CDR_tag_get_f:cn { #2 } }
172     }
173   }
174 }
175 \cs_new:Npn \CDR_tag_get_f:cn #1 #2 {
176   \quark_if_no_value:nF { #2 } {
177     \CDR_tag_if_exist_here:ccT { #2 } { #1 } {
178       \seq_map_break:n {
179         \use:c { \CDR_tag_get_path:cc { #2 } { #1 } }
180       }
181     }
182   }
183 }

```

\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.

```

184 \cs_new:Npn \CDR_tag_get:c {
185   \CDR_tag_get:cc { __local }
186 }

```

\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>.

```
187 \cs_new:Npn \CDR_tag_get:cN #1 #2 {
188   \tl_set:Nx #2 { \CDR_tag_get:c { #1 } }
189 }
```

\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>.

```
190 \prg_new_conditional:Nnn \CDR_tag_get:ccN { T, F, TF } {
191   \CDR_tag_if_exist:nnTF { #1 } { #2 } {
192     \tl_set:Nx #3 \CDR_tag_get:cc { #1 } { #2 }
193     \prg_return_true:
194   } {
195     \prg_return_false:
196   }
197 }
```

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.

```
198 \cs_new:Npn \CDR_tag_parent_seq:c #1 {
199   g_CDR:parent.tag @ #1 _seq
200 }
```

\CDR_tag_inherit:cn \CDR_tag_inherit:cn {<child name>} {<parent names comma list>}

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

```
201 \cs_new:Npn \CDR_tag_inherit:cn #1 #2 {
202   \seq_set_from_clist:cn { \CDR_tag_parent_seq:c { #1 } } { #2 }
203   \seq_remove_duplicates:c \l_CDR_tl
204   \seq_remove_all:cn \l_CDR_tl {}
205   \seq_put_right:cn \l_CDR_tl { \q_no_value }
206 }
207 \cs_new:Npn \CDR_tag_inherit:cx {
208   \exp_args:Nnx \CDR_tag_inherit:cn
209 }
210 \cs_new:Npn \CDR_tag_inherit:cV {
211   \exp_args:NnV \CDR_tag_inherit:cn
212 }
```

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.

```
213 \AddToHook { begindocument/before } {
214   \IfFileExists {./\jobname.aux} {} {
215     \lua_now:n {CDR:cache_clean_all()}
216   }
217 }
```

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

```
218 \AddToHook { enddocument/end } {
219   \lua_now:n {CDR:cache_clean_unused()}
220 }
```

8 Utilities

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

```
221 \cs_new:Npn \CDR_clist_map_inline:Nnn #1 #2 {
222   \clist_if_empty:NTF #1 {
223     #2
224     \use_none:n
225   } {
226     \clist_map_inline:Nn #1
227   }
228 }
```

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

```
229 \prg_new_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
230   \PackageError
231     { coder }
232     { Conditional-not-available }
233 }
```

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

```
234 \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`.

```
235 \cs_generate_variant:Nn \keys_define:nn { Vn, xn }
236 \cs_new:Npn \CDR_tag_keys_define:nn #1 {
237   \keys_define:xn { \c_CDR_tag / \exp_not:n { #1 } }
238 }
239 \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`.

```
240 \cs_new:Npn \CDR_tag_keys_set:nn #1 {
241   \exp_args:Nx
242   \keys_set:nn { \c_CDR_tag / \exp_not:n { #1 } }
243 }
```

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>`.

```
244 \cs_new:Npn \CDR_keys_set_known:nnN #1 #2 {
245   \keys_set_known:nnnN { #1 } { #2 } { #1 }
246 }
247 \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.

```

248 \cs_generate_variant:Nn \keys_set_known:nnnN { VVV, nVx }
249 \cs_new:Npn \CDR_tag_keys_set_known:nnN #1 {
250   \CDR_keys_set_known:xnN { \c_CDR_tag / \exp_not:n { #1 } }
251 }
252 \cs_generate_variant:Nn \CDR_tag_keys_set_known:nnN { nV }

```

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

```

253 \tl_set:Nn \l_CDR_tl { /([~/*])(?:/(.))*? $ } \use_none:n { $ }
254 \tl_put_left:NV \l_CDR_tl \c_CDR_tag
255 \tl_put_left:Nn \l_CDR_tl { ^ }
256 \exp_args:NNV
257 \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 `'/`.

```

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

```



```

288         \l_CDR_tl .value_required:n = true,
289     }
290 }
291 }
292 }
293 } {
294     \regex_match:NnT \c_CDR_engine_regex { #1 } {
295         \CDR_tag_keys_define:nn { default } {
296             #1 .code:n = \CDR_tag_set:n { ##1 },
297             #1 .value_required:n = true,
298         }
299     }
300 }
301 }
302 \cs_new:Npn \CDR_tag_provide_from_clist:nn #1 #2 {
303     \CDR_tag_provide_from_clist:n { #1 }
304 }
305 \cs_new:Npn \CDR_tag_provide_from_keyval:n {
306     \keyval_parse:nnn {
307         \CDR_tag_provide_from_clist:n
308     } {
309         \CDR_tag_provide_from_clist:nn
310     }
311 }
312 \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 {⟨true code⟩} {⟨false code⟩}</code> Execute <code>⟨true code⟩</code> when <code>pygments</code> is available, <code>⟨false code⟩</code> otherwise. <i>Implementation detail:</i> we define the conditionals and set them afterwards.
---	--

```

313 \sys_get_shell:nnN {which-pygmentize} {} \l_CDR_tl
314 \prg_new_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } { }
315 \tl_if_in:NnTF \l_CDR_tl { pygmentize } {
316     \prg_set_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } {
317         \prg_return_true:
318     }
319 } {
320     \prg_set_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } {
321         \prg_return_false:
322     }
323 }

```

9.2.2 `__pygment` `l3keys` module

```

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

325   lang .code:n = \CDR_tag_set:,
326   lang .value_required:n = true,

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

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

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

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

  commandprefix=<text> The LATEX commands used to produce colored output are con-
    structed using this prefix and some letters. Initially PY.

330   commandprefix .code:n = \CDR_tag_set:,
331   commandprefix .value_required:n = true,

  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.

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

  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.

334   escapeinside .code:n = \CDR_tag_set:,
335   escapeinside .value_required:n = true,

  __initialize Initializer.

336   __initialize .meta:n = {
337     lang = tex,
338     pygments = \CDR_has_pygments:TF { true } { false },
339     style=default,
340     commandprefix=PY,
341     mathescape=false,
342     escapeinside=,
343   },
344   __initialize .value_forbidden:n = true,

345 }
346 \AtBeginDocument{
347   \CDR_tag_keys_set:nn { __pygments } { __initialize }
348 }

```

9.2.3 \c_CDR_tag / __pygments.block l3keys module

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

- **texcomments**[=*true*|*false*] If set to *true*, enables L^AT_EX comment lines. That is, L^AT_EX markup in comment tokens is not escaped so that L^AT_EX can render it. Initially *false*.

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

- **__initialize** Initializer.

```
352 __initialize .meta:n = {
353   texcomments=false,
354 },
355 __initialize .value_forbidden:n = true,

356 }
357 \AtBeginDocument{
358   \CDR_tag_keys_set:nn { __pygments.block } { __initialize }
359 }
```

9.3 Specific to **coder**

9.3.1 default l3keys module

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

Keys are:

- **cache** Set to *true* if `coder-tool.py` should use already existing files instead of creating new ones.

```
361 cache .code:n = \CDR_tag_boolean_set:x { #1 },
```

- **debug** Set to *true* if various debugging messages should be printed to the console .

```
362 debug .code:n = \CDR_tag_boolean_set:x { #1 },
```

- **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.

```
363 post~processor .code:n = \CDR_tag_set:,
364 post~processor .value_required:n = true,
```

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

```
365 parskip .code:n = \CDR_tag_set:,
366 parskip .value_required:n = true,
```

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

```

367 engine .code:n = \CDR_tag_set:,
368 engine .value_required:n = true,

```

● **default engine options**=*(default engine options)* to specify the corresponding options,

```

369 default~engine~options .code:n = \CDR_tag_set:,
370 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.

```

371 __initialize .meta:n = {
372   cache = false,
373   debug = false,
374   post-processor = ,
375   parskip = \the\parskip,
376   engine = default,
377   default~engine~options = ,
378 },
379 __initialize .value_forbidden:n = true,
380 }
381 \AtBeginDocument{
382   \CDR_tag_keys_set:nn { default } { __initialize }
383 }

```

9.3.2 default.code l3keys module

Void for the moment.

```

384 \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.

```

385 __initialize .meta:n = {
386 },
387 __initialize .value_forbidden:n = true,
388 }
389 \AtBeginDocument{
390   \CDR_tag_keys_set:nn { default.code } { __initialize }
391 }

```

9.3.3 default.block l3keys module

```
392 \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.

```
393 tags .code:n = {  
394   \clist_set:Nn \l_CDR_tags_clist { #1 }  
395   \clist_remove_duplicates:N \l_CDR_tags_clist  
396   \exp_args:NV  
397   \CDR_tag_set:n \l_CDR_tags_clist  
398 },
```

```
399 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.

```
400 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,

```
401 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),

```
402 tags~format .code:n = \CDR_tag_set:,  
403 tags~format .value_required:n = true,
```

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

```
404 blockskip .code:n = \CDR_tag_set:,  
405 blockskip .value_required:n = true,
```

● **__initialize** the separation with the surrounding text. Initially *\topsep*.

```
406 __initialize .meta:n = {  
407   tags = ,  
408   show~tags = true,  
409   only~top = true,  
410   use~margin = true,  
411   tags~format = {  
412     \sffamily  
413     \scriptsize  
414     \color{gray}  
415   },  
416   blockskip = \topsep,  
417 },  
418 __initialize .value_forbidden:n = true,
```

```

419 }
420 \AtBeginDocument{
421   \CDR_tag_keys_set:nn { default.block } { __initialize }
422 }

```

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

```

423 \CDR_tag_keys_define:nn { __fancyvrb } {

```

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

```

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

```

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

```

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

```

- **fontsize**=** 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.

```

428   fontsize .code:n = \CDR_tag_set:,
429   fontsize .value_required:n = true,

```

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

```

430   fontshape .code:n = \CDR_tag_set:,
431   fontshape .value_required:n = true,

```

- **fontseries**=*<series name>* L^AT_EX font series to use. Initially `auto`: the same as the current font.

```

432   fontseries .code:n = \CDR_tag_set:,
433   fontseries .value_required:n = true,

```

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

```

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

```

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

```
435 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.

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

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

```
437 tabsize .code:n = \CDR_tag_set:,
438 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.

```
439 defineactive .code:n = \CDR_tag_set:,
440 defineactive .value_required:n = true,
```

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

```
441 relabel .code:n = \CDR_tag_set:,
442 relabel .value_required:n = true,
```

✅ **__initialize** Initialization.

```
443 __initialize .meta:n = {
444   formatcom = ,
445   fontfamily = tt,
446   fontsize = auto,
447   fontseries = auto,
448   fontshape = auto,
449   showspaces = false,
450   showtabs = false,
451   obeytabs = false,
452   tabsize = 2,
453   defineactive = ,
454   relabel = ,
455 },
456 __initialize .value_forbidden:n = true,

457 }
458 \AtBeginDocument{
459   \CDR_tag_keys_set:nn { __fancyvrb } { __initialize }
460 }
```

9.4.2 `__fancyvrb.block` `l3keys` module

Block specific options.

```
461 \regex_const:Nn \c_CDR_integer_regex { ^(+|-)?\d+$ } \use_none:n { $ }
462 \CDR_tag_keys_define:nn { __fancyvrb.block } {
```

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

```
463   commentchar .code:n = \CDR_tag_set:,
464   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.

```
465   gobble .choices:nn = {
466     0,1,2,3,4,5,6,7,8,9
467   } {
468     \CDR_tag_choices_set:
469   },
```

- **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 `LATEX \fboxsep` macro is added between the left vertical line and the text. Initially **none**: no frame.

```
470   frame .choices:nn =
471     { none, leftline, topline, bottomline, lines, single }
472     { \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.

```
473   label .code:n = \CDR_tag_set:,
474   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.

```
475   labelposition .choices:nn =
476     { none, topline, bottomline, all }
477     { \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.


```

478 numbers .choices:nn =
479   { none, left, right }
480   { \CDR_tag_choices_set: },

```

● **numbersep**= $\langle dimension \rangle$ gap between numbers and verbatim lines. Initially 12pt.

```

481 numbersep .code:n = \CDR_tag_set:,
482 numbersep .value_required:n = true,

```

● **firstnumber**=**auto**|**last**| $\langle integer \rangle$ 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.

```

483 firstnumber .code:n = {
484   \regex_match:NnTF \c_CDR_integer_regex { #1 } {
485     \CDR_tag_set:
486   } {
487     \str_case:nnF { #1 } {
488       { auto } { \CDR_tag_set: }
489       { last } { \CDR_tag_set: }
490     } {
491       \PackageWarning
492         { CDR }
493         { Value~'#1'~not~in~auto,~last. }
494     }
495   }
496 },
497 firstnumber .value_required:n = true,

```

● **stepnumber**= $\langle integer \rangle$ interval at which line numbers are printed. Initially 1: all lines are numbered.

```

498 stepnumber .code:n = \CDR_tag_set:,
499 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.

```

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

```

● **firstline**= $\langle integer \rangle$ first line to print. Initially empty: all lines from the first are printed.

```

501 firstline .code:n = \CDR_tag_set:,
502 firstline .value_required:n = true,

```

● **lastline**= $\langle integer \rangle$ last line to print. Initially empty: all lines until the last one are printed.

```

503 lastline .code:n = \CDR_tag_set:,
504 lastline .value_required:n = true,

```

🔴 **baselinestretch=auto**⟨*dimension*⟩ value to give to the usual `\baselinestretch` L^AT_EX parameter. Initially `auto`: its current value just before the verbatim command.

```
505 baselinestretch .code:n = \CDR_tag_set:,
506 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.

```
507 xleftmargin .code:n = \CDR_tag_set:,
508 xleftmargin .value_required:n = true,
```

🔴 **xrightmargin=⟨*dimension*⟩** right margin to add after each line. Initially `Opt`: no right margin.

```
509 xrightmargin .code:n = \CDR_tag_set:,
510 xrightmargin .value_required:n = true,
```

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

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

🔴 **hfuzz=⟨*dimension*⟩** value to give to the T_EX `\hfuzz` dimension for text to format. This can be used to avoid seeing some unimportant overfull box messages. Initially `2pt`.

```
512 hfuzz .code:n = \CDR_tag_set:,
513 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`.

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

✅ **__initialize** Initialization.

```
515 __initialize .meta:n = {
516   commentchar = ,
517   gobble = 0,
518   frame = none,
519   label = ,
520   labelposition = none,% auto?
521   numbers = left,
522   numbersep = \hspace{1ex},
523   firstnumber = auto,
```

```

524     stepnumber = 1,
525     numberblanklines = true,
526     firstline = ,
527     lastline = ,
528     baselinestretch = auto,
529     resetmargins = true,
530     xleftmargin = 0pt,
531     xrightmargin = 0pt,
532     hfuzz = 2pt,
533     samepage = false,
534 },
535 __initialize .value_forbidden:n = true,

536 }
537 \AtBeginDocument{
538   \CDR_tag_keys_set:nn { __fancyvrb.block } { __initialize }
539 }

```

9.4.3 __fancyvrb.all l3keys module

Options available when pygments is not used.

```

540 \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.

```

541   commandchars .code:n = \CDR_tag_set:,
542   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.

```

543   codes .code:n = \CDR_tag_set:,
544   codes .value_required:n = true,

```

- ✓ **__initialize** Initialization.

```

545   __initialize .meta:n = {
546     commandchars = ,
547     codes = ,
548   },
549   __initialize .value_forbidden:n = true,

550 }
551 \AtBeginDocument{
552   \CDR_tag_keys_set:nn { __fancyvrb.all } { __initialize }
553 }

```


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

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

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

```
555   only~description .choices:nn = { false, true, {} } {
556     \int_compare:nNnTF \l_keys_choice_int = 1 {
557       \prg_set_conditional:Nnn \CDR_if_only_description: { p, T, F, TF } { \prg_return_true: }
558     } {
559       \prg_set_conditional:Nnn \CDR_if_only_description: { p, T, F, TF } { \prg_return_false: }
560     }
561   },
562   only~description .initial:n = false,
```

-  **python path** if automatic processing is not available, manually setting the path to the python utility is required. Giving a void path forces an automatic guess using which.

```
563   python-path .code:n = {
564     \str_set:Nn \l_CDR_str { #1 }
565     \lua_now:n { CDR:set_python_path('l_CDR_str') }
566   },
567 }
```

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.

```

568 \exp_args_generate:n { xV, nnV }
569 \cs_new:Npn \CDR_check_unknown:N #1 {
570   \tl_if_empty:NF #1 {
571     \cs_set:Npn \CDR_check_unknown:n ##1 {
572       \PackageWarning
573         { coder }
574         { Unknow~key~'##1' }
575     }
576     \cs_set:Npn \CDR_check_unknown:nn ##1 ##2 {
577       \CDR_check_unknown:n { ##1 }
578     }
579     \exp_args:NnnV
580     \keyval_parse:nnn {
581       \CDR_check_unknown:n
582     } {
583       \CDR_check_unknown:nn
584     } #1
585   }
586 }

587 \NewDocumentCommand \CDRSet { m } {
588   \CDR_keys_set_known:nnN { CDR@Set } { #1 } \l_CDR_keyval_tl
589   \clist_map_inline:nn {
590     __pygments, __pygments.block,
591     default.block, default.code, default,
592     __fancyvrb, __fancyvrb.block, __fancyvrb.all
593   } {
594     \CDR_tag_keys_set_known:nVN { ##1 } \l_CDR_keyval_tl \l_CDR_keyval_tl
595   }
596   \CDR_keys_set_known:VVN \c_CDR_Tags \l_CDR_keyval_tl \l_CDR_keyval_tl
597   \CDR_tag_provide_from_keyval:V \l_CDR_keyval_tl
598   \CDR_tag_keys_set_known:nVN { default } \l_CDR_keyval_tl \l_CDR_keyval_tl
599   \CDR_keys_set_known:VVN \c_CDR_Tags \l_CDR_keyval_tl \l_CDR_keyval_tl
600   \CDR_check_unknown:N \l_CDR_keyval_tl
601 }

```

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

```
602 \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.

```

603 \cs_new:Npn \CDR_export_get_path:cc #1 #2 {
604   \c_CDR_export_get @ #1 / #2 :
605 }

```

\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_EX group level.

\CDR_export_set:VcV

```

606 \cs_new_protected:Npn \CDR_export_set:ccn #1 #2 #3 {
607   \cs_set:cpn { \CDR_export_get_path:cc { #1 } { #2 } } { \exp_not:n { #3 } }
608 }
609 \cs_new_protected:Npn \CDR_export_set:Vcn #1 {
610   \exp_args:NV
611   \CDR_export_set:ccn { #1 }
612 }
613 \cs_new_protected:Npn \CDR_export_set:VcV #1 #2 #3 {
614   \exp_args:NVnV
615   \CDR_export_set:ccn #1 { #2 } #3
616 }

```

\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.

```

617 \prg_new_conditional:Nnn \CDR_export_if_exist:cc { p, T, F, TF } {
618   \cs_if_exist:cTF { \CDR_export_get_path:cc { #1 } { #2 } } {
619     \prg_return_true:
620   } {
621     \prg_return_false:
622   }
623 }

```

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

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

```

624 \cs_new:Npn \CDR_export_get:cc #1 #2 {
625   \CDR_export_if_exist:ccT { #1 } { #2 } {
626     \use:c { \CDR_export_get_path:cc { #1 } { #2 } }
627   }
628 }

```

\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.

```

629 \prg_new_protected_conditional:Nnn \CDR_export_get:ccN { T, F, TF } {
630   \CDR_export_if_exist:ccTF { #1 } { #2 } {
631     \tl_set:Nx #3 { \CDR_export_get:cc { #1 } { #2 } }
632     \prg_return_true:
633   } {
634     \prg_return_false:
635   }
636 }

```

`\g_CDR_export_prop` Global storage for $\langle \text{file name} \rangle = \langle \text{file export info} \rangle$

```

637 \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.

```

638 \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.

```

639 \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.

```

640 \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.

```

641 \keys_define:nn { CDR@Export } {


```

 **file**= $\langle \text{name} \rangle$ the output file name, must be provided otherwise an error is raised.

```

642   file .tl_set:N = \l_CDR_file_tl,
643   file .value_required:n = true,

```

 **tags**= $\langle \text{tags comma list} \rangle$ the list of tags. No exportation when this list is void. Initially empty.

```

644   tags .code:n = {
645     \clist_set:Nn \l_CDR_tags_clist { #1 }
646     \clist_remove_duplicates:N \l_CDR_tags_clist
647     \prop_put:NVV \l_CDR_prop \l_keys_key_str \l_CDR_tags_clist
648   },
649   tags .value_required:n = true,

```

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

```
650 lang .code:n = {
651     \prop_put:NVn \l_CDR_prop \l_keys_key_str { #1 }
652 },
653 lang .value_required:n = true,
```

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

```
654 preamble .code:n = {
655     \prop_put:NVn \l_CDR_prop \l_keys_key_str { #1 }
656 },
657 preamble .value_required:n = true,
```

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

```
658 postamble .code:n = {
659     \prop_put:NVn \l_CDR_prop \l_keys_key_str { #1 }
660 },
661 postamble .value_required:n = true,
```

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

```
662 raw .choices:nn = { false, true, {} } {
663     \prop_put:NVx \l_CDR_prop \l_keys_key_str {
664         \int_compare:nNnTF
665             \l_keys_choice_int = 1 { false } { true }
666     }
667 },
```

✅ **__initialize** Meta key to properly initialize all the variables.

```
668 __initialize .meta:n = {
669     __initialize_prop = #1,
670     file=,
671     tags=,
672     lang=tex,
673     preamble=,
674     postamble=,
675     raw=false,
676 },
677 __initialize .default:n = \l_CDR_prop,
```

✅ **__initialize_prop** Goody: properly initialize the local property storage.

```
678 __initialize_prop .code:n = \prop_clear:N #1,
679 __initialize_prop .default:n = \l_CDR_prop,
680 }
```


11.3 Implementation

```

681 \NewDocumentCommand \CDRExport { m } {
682   \keys_set:nn { CDR@Export } { __initialize }
683   \keys_set:nn { CDR@Export } { #1 }
684   \tl_if_empty:NTF \l_CDR_file_tl {
685     \PackageWarning
686       { coder }
687       { Missing~key~‘file’ }
688   } {
689     \CDR_export_set:VcV \l_CDR_file_tl { file } \l_CDR_file_tl
690     \prop_map_inline:Nn \l_CDR_prop {
691       \CDR_export_set:Vcn \l_CDR_file_tl { ##1 } { ##2 }
692     }

```

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

```

693   \prop_get:NnNT \l_CDR_prop { tags } \l_CDR_tags_clist {
694     \exp_args:NV
695     \CDR_export_get:ccNT \l_CDR_file_tl { lang } \l_CDR_tl {
696       \clist_map_inline:Nn \l_CDR_tags_clist {
697         \CDR_tag_set:ccV { ##1 } { lang } \l_CDR_tl
698       }
699     }
700   }
701 }
702 }

```

Files are created at the end of the typesetting process.

```

703 \AddToHook { enddocument / end } {
704   \prop_map_inline:Nn \g_CDR_export_prop {
705     \tl_set:Nn \l_CDR_prop { #2 }
706     \str_set:Nx \l_CDR_str {
707       \prop_item:Nn \l_CDR_prop { file }
708     }
709     \lua_now:n { CDR:export_file('l_CDR_str') }
710     \clist_map_inline:nn {
711       tags, raw, preamble, postamble
712     } {
713       \str_set:Nx \l_CDR_str {
714         \prop_item:Nn \l_CDR_prop { ##1 }
715       }
716       \lua_now:n {
717         CDR:export_file_info('##1', 'l_CDR_str')
718       }
719     }
720     \lua_now:n { CDR:export_file_complete() }
721   }
722 }

```

12 Style

`pygments`, through `coder-tool.py`, creates style commands, but the storage is managed on the \LaTeX side by `coder.sty`. This is a \LaTeX style API.

<code>\CDR@StyleDefine</code>	<code>\CDR@StyleDefine {<pygments style name>} {<definitions>}</code>
-------------------------------	---

Define the definitions for the given *<pygments style name>*.

```

723 \cs_set:Npn \CDR@StyleDefine #1 {
724   \tl_gset:cn { g_CDR@Style/#1 }
725 }

```

<code>\CDR@StyleUse</code>	<code>\CDR@StyleUse {<pygments style name>}</code>
----------------------------	--

Use the definitions for the given *<pygments style name>*. No safe check is made.

```

726 \cs_set:Npn \CDR@StyleUse #1 {
727   \tl_use:c { g_CDR@Style/#1 }
728 }

```

<code>\CDR@StyleExist</code>	<code>\CDR@StyleExist {<pygments style name>} {<true code>} {<false code>}</code>
------------------------------	---

Execute *<true code>* if a style exists with that given name, *<false code>* otherwise.

```

729 \prg_new_conditional:Nnn \CDR@StyleIfExist:c { TF } {
730   \tl_if_exist:cTF { g_CDR@Style/#1 } {
731     \prg_return_true:
732   } {
733     \prg_return_false:
734   }
735 }
736 \cs_set_eq:NN \CDR@StyleIfExist \CDR@StyleIfExist:cTF

```

13 Creating display engines

13.1 Utilities

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

```

737 \cs_new:Npn \CDR_code_engine:c #1 {
738   CDR@colored/code/#1:nn
739 }
740 \cs_new:Npn \CDR_block_engine:c #1 {
741   CDR@colored/block/#1
742 }
743 \cs_new:Npn \CDR_code_engine:V {
744   \exp_args:NV \CDR_code_engine:c
745 }
746 \cs_new:Npn \CDR_block_engine:V {
747   \exp_args:NV \CDR_block_engine:c
748 }

```

`\l_CDR_engine_tl` Storage for an engine name.

749 \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

\CDRNewCodeEngine \CDRNewCodeEngine {<engine name>}{<engine body>}
\CDRRenewCodeEngine \CDRRenewCodeEngine{<engine name>}{<engine body>}

<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.

```

750 \NewDocumentCommand \CDRNewCodeEngine { mm } {
751   \exp_args:Nx
752   \tl_if_empty:nTF { #1 } {
753     \PackageWarning
754       { coder }
755     { The~engine~cannot~be~void. }
756   } {
757     \cs_new:cpn { \CDR_code_engine:c {#1} } ##1 ##2 {
758       \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
759       #2
760     }
761     \ignorespaces
762   }
763 }

764 \NewDocumentCommand \CDRRenewCodeEngine { mm } {
765   \exp_args:Nx
766   \tl_if_empty:nTF { #1 } {
767     \PackageWarning
768       { coder }
769     { The~engine~cannot~be~void. }
770     \use_none:n
771   } {
772     \cs_if_exist:cTF { \CDR_code_engine:c { #1 } } {
773       \cs_set:cpn { \CDR_code_engine:c { #1 } } ##1 ##2 {
774         \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
775         #2
776       }
777     } {
778       \PackageWarning
779         { coder }
780       { No~code~engine~#1.}
781     }
782     \ignorespaces

```

```

783 }
784 }

```

\CDR@CodeEngineApply \CDR@CodeEngineApply {<verbatim code>}

Get the code engine and apply. When the code engine is not recognized, an error is raised. *Implementation detail:* the argument is parsed by the last macro.

```

785 \cs_new:Npn \CDR@CodeEngineApply {
786   \CDR_tag_get:cN { engine } \l_CDR_tl
787   \CDR_if_code_engine:VF \l_CDR_tl {
788     \PackageError
789       { coder }
790       { \l_CDR_tl\space code-engine-unknown,~replaced-by-'default' }
791       { See~\CDRNewCodeEngine-in-the-coder-manual }
792     \tl_set:Nn \l_CDR_tl { default }
793   }
794   \exp_args:Nnx
795   \use:c { \CDR_code_engine:V \l_CDR_tl }
796   { \CDR_tag_get:c { \l_CDR_tl-engine-options } }
797 }

```

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

Create a L^AT_EX 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.

```

798 \NewDocumentCommand \CDRNewBlockEngine { mm } {
799   \NewDocumentEnvironment { \CDR_block_engine:c { #1 } } { m } {
800     \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
801     #2
802   }
803 }

804 \NewDocumentCommand \CDRRenewBlockEngine { mm } {
805   \tl_if_empty:nTF { #1 } {
806     \PackageWarning
807       { coder }
808       { The~engine~cannot~be~void. }
809     \use_none:n
810   } {
811     \RenewDocumentEnvironment { \CDR_block_engine:c { #1 } } { m } {
812       \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
813       #2
814     }
815   }
816 }

```

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⟩*.

```
817 \prg_new_conditional:Nnn \CDR_if_code_engine:c { p, T, F, TF } {
818   \cs_if_exist:cTF { \CDR_code_engine:c { #1 } } {
819     \prg_return_true:
820   } {
821     \prg_return_false:
822   }
823 }
824 \prg_new_conditional:Nnn \CDR_if_code_engine:V { p, T, F, TF } {
825   \cs_if_exist:cTF { \CDR_code_engine:V #1 } {
826     \prg_return_true:
827   } {
828     \prg_return_false:
829   }
830 }
```

`\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⟩*.

```
831 \prg_new_conditional:Nnn \CDR_has_block_engine:c { p, T, F, TF } {
832   \cs_if_exist:cTF { \CDR_block_engine:c { #1 } } {
833     \prg_return_true:
834   } {
835     \prg_return_false:
836   }
837 }
838 \prg_new_conditional:Nnn \CDR_has_block_engine:V { p, T, F, TF } {
839   \cs_if_exist:cTF { \CDR_block_engine:V #1 } {
840     \prg_return_true:
841   } {
842     \prg_return_false:
843   }
844 }
```

13.4 Default code engine

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

```
845 \CDRNewCodeEngine { default } { #2 }
```

13.5 Default block engine

The default block engine does nothing.

```
846 \CDRNewBlockEngine { default } { } { }
```

13.6 tcolorbox related engine

If the tcolorbox is loaded, related code and block engines are available.

14 \CDRCode function

14.1 Storage

`\l_CDR_tag_tl` To store the tag given.


```
847 \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.

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

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

```
849   tag .tl_set:N = \l_CDR_tag_tl,
850   tag .value_required:n = true,
```

 **__initialize** initialize

```
851   __initialize .meta:n = {
852     tag = default,
853   },
854   __initialize .value_forbidden:n = true,
855 }
```

14.3 Implementation

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

```
856 \cs_new:Npn \CDR_code_fvset_braced:nn #1 #2 {
857   \fvset { #1 = { #2 } }
858 }
859 \cs_set:Npn \CDR_code_fvset: {
860   \tl_clear:N \l_CDR_options_tl
861   \clist_map_inline:nn {
862     formatcom,
863     fontfamily,
864     fontsize,
865     fontshape,
866     showspaces,
867     showtabs,
868     obeytabs,
869     tabsize,
```

```

870 %   defineactive,
871 %   rellabel,
872 } {
873   \tl_set:Nx \l_CDR_tl { \CDR_tag_get:c { ##1 } }
874   \tl_if_in:NnTF \l_CDR_tl { , } {
875     \exp_args:NnV
876     \CDR_code_fvset_braced:nn { ##1 } \l_CDR_tl
877   } {
878     \tl_put_left:Nn \l_CDR_tl { ##1 = }
879     \exp_args:NV
880     \fvset \l_CDR_tl
881   }
882 }
883 }
884
885 \cs_new:Npn \CDR_brace_if_contains_comma:n #1 {
886   \tl_if_in:nnTF { #1 } { , } { { #1 } } { #1 }
887 }
888 \cs_generate_variant:Nn \CDR_brace_if_contains_comma:n { V }
889 \cs_new:Npn \CDR_feed_options_clist:N #1 {
890   \clist_map_inline:nn {
891     formatcom, fontfamily, fontsize, fontshape,
892     tabsize, defineactive, rellabel
893   } {
894     \CDR_tag_get:cN { ##1 } \l_CDR_tl
895     \tl_if_empty:NF \l_CDR_tl {
896       \tl_put_left:Nn #1 {
897         ##1 = \CDR_brace_if_contains_comma:V \l_CDR_tl,
898       }
899     }
900   }
901   \clist_map_inline:nn { showspaces, showtabs, obeytabs } {
902     \tl_put_left:Nx #1 { ##1 = \CDR_tag_get:cN { ##1 }, }
903   }
904 }
905 \cs_new:Npn \CDR_code_format: {
906   \frenchspacing
907   \CDR_tag_get:cN { baselinestretch } \l_CDR_tl
908   \tl_if_eq:NnF \l_CDR_tl { auto } {
909     \exp_args:NNV
910     \def \baselinestretch \l_CDR_tl
911   }
912   \CDR_tag_get:cN { fontfamily } \l_CDR_tl
913   \tl_if_eq:NnT \l_CDR_tl { tt } { \tl_set:Nn \l_CDR_tl { lmtt } }
914   \exp_args:NV
915   \fontfamily \l_CDR_tl
916   \clist_map_inline:nn { series, shape } {
917     \CDR_tag_get:cN { font##1 } \l_CDR_tl
918     \tl_if_eq:NnF \l_CDR_tl { auto } {
919       \exp_args:NnV
920       \use:c { font##1 } \l_CDR_tl
921     }
922   }
923   \CDR_tag_get:cN { fontsize } \l_CDR_tl

```

```

924 \tl_if_eq:NnF \l_CDR_tl { auto } {
925   \tl_use:N \l_CDR_tl
926 }
927 \selectfont
928 % \@noligs ?? this is in fancyvrb but does not work here as is
929 }
930 \cs_new:Npn \CDR_code:n #1 {
931   \CDR_tag_inherit:cx { __local } {
932     \tl_if_empty:NF \l_CDR_tag_tl { \l_CDR_tag_tl, }
933     __code, default.code, default, __pygments, __fancyvrb,
934   }
935   \clist_clear:N \l_CDR_options_clist
936   \CDR_feed_options_clist:N \l_CDR_options_clist
937   \CDR_if_truthy:eTF { \CDR_tag_get:c {pygments} } {
938     \DefineShortVerb { #1 }
939     \SaveVerb [
940       aftersave = {
941         \UndefineShortVerb { #1 }
942         \lua_now:n { CDR:highlight_code_prepare() }
943         \CDR@StyleIfExist { \l_CDR_tl } {
944           \lua_now:n { CDR:highlight_set('already_style', 'true') }
945         } { }
946         \CDR_tag_get:cN {cache} \l_CDR_tl
947         \lua_now:n { CDR:highlight_set_var('cache') }
948         \CDR_tag_get:cN {debug} \l_CDR_tl
949         \lua_now:n { CDR:highlight_set_var('debug') }
950         \CDR_tag_get:cN {style} \l_CDR_tl
951         \lua_now:n { CDR:highlight_set_var('style') }
952         \CDR_tag_get:cN {lang} \l_CDR_tl
953         \lua_now:n { CDR:highlight_set_var('lang') }
954         \lua_now:n { CDR:highlight_set_var('code', 'FV@SV@CDR@Code') }
955         \CDR_code_format:
956         \lua_now:n { CDR:highlight_code() }
957         \group_end:
958       }
959     ] { CDR@Code } #1
960   } {
961     \DefineShortVerb { #1 }
962     \SaveVerb [
963       aftersave = {
964         \UndefineShortVerb { #1 }
965         \CDR_code_fvset:
966         \CDR@CodeEngineApply { \UseVerb { CDR@Code } }
967         \group_end:
968       }
969     ] { CDR@Code } #1
970   }
971 }
972 \NewDocumentCommand \CDRCode { O{} } {
973   \group_begin:
974   \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
975     \prg_return_false:
976   }
977   \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {

```



```

978     __code, __pygments, default.code, default, __fancyvrb, __fancyvrb.all
979 }
980 \CDR_tag_keys_set_known:nnN { __local } { #1 } \l_CDR_keyval_tl
981 \CDR_tag_provide_from_keyval:V \l_CDR_keyval_tl
982 \exp_args:NnV
983 \CDR_tag_keys_set:nn { __local } \l_CDR_keyval_tl
984 \CDR_code:n
985 }

```

\CDR_to_lua: \CDR_to_lua:

Retrieve info from the tree storage and forwards to lua.

```

986 \cs_new:Npn \CDR_to_lua: {
987   \lua_now:n { CDR:options_reset() }
988   \seq_map_inline:Nn \g_CDR_tag_path_seq {
989     \CDR_tag_get:cNT { ##1 } \l_CDR_tl {
990       \str_set:Nx \l_CDR_str { \l_CDR_tl }
991       \lua_now:n { CDR:option_add('##1','l_CDR_str') }
992     }
993   }
994 }

```

15 CDRBlock environment

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

15.1 Storage

\l_CDR_block_prop

```

995 \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.

```

996 \CDR_tag_keys_define:nn { __block } {


```

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

```

997   ignore .code:n = \CDR_tag_boolean_set:x { #1 },
998   ignore .default:n = true,

```

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

```

999   test .code:n = \CDR_tag_boolean_set:x { #1 },
1000   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.

```
1001 engine-options .code:n = \CDR_tag_set:,
1002 engine options .default:n = true,
```

🔴 **__initialize** initialize

```
1003 __initialize .meta:n = {
1004     tags = ,
1005     ignore = false,
1006     test= false,
1007 },
1008 __initialize .value_forbidden:n = true,
1009 }
```

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.

```
1010 \clist_map_inline:nn { i, ii, iii, iv } {
1011     \cs_set_eq:cc { CDR@ListProcessLine@ #1 } { FV@ListProcessLine@ #1 }
1012 }
1013 \cs_new:Npn \CDR_process_line:n #1 {
1014     \str_set:Nn \l_CDR_str { #1 }
1015     \lua_now:n {CDR:process_line('l_CDR_str')}
1016 }

1017 \cs_new:Npn \CDR_keys_inherit__:nnn #1 #2 #3 {
1018     \keys_define:nn { #1 } { #2 .inherit:n = { #3 } }
1019 }
1020 \cs_new:Npn \CDR_keys_inherit:nnn #1 #2 #3 {
1021     \tl_if_empty:nTF { #1 } {
1022         \CDR_keys_inherit__:nnn { } { #2 } { #3 }
1023     } {
1024         \clist_set:Nn \l_CDR_clist { #3 }
1025         \exp_args:Nnnx
1026         \CDR_keys_inherit__:nnn { #1 } { #2 } {
1027             #1 / \clist_use:Nn \l_CDR_clist { ,#1/ }
1028         }
1029     }
1030 }
1031 \cs_generate_variant:Nn \CDR_keys_inherit:nnn { VnV, Vnn }
1032 \def\FVB@CDRBlock #1 {
1033     \@bsphack
1034     \group_begin:
```

```

1035 \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
1036   \prg_return_true:
1037 }
1038 \clist_set:Nn \l_tmpa_clist {
1039   __block, default.block, default, __fancyvrb.block, __fancyvrb,
1040 }
1041 \CDR_keys_inherit:VnV \c_CDR_tag { __local } \l_tmpa_clist
1042 \clist_map_inline:Nn \l_tmpa_clist {
1043   \CDR_tag_keys_set:nn { ##1 } { __initialize }
1044 }
1045 \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.

```

1046 \clist_if_empty:NT \l_CDR_tags_clist {
1047   \CDR_tag_get:ccN { default.block } { tags } \l_CDR_tags_clist
1048   \clist_if_empty:NT \l_CDR_tags_clist {
1049     \PackageWarning
1050       { coder }
1051       { No~(default)~tags~provided. }
1052   }
1053 }
1054 \lua_now:n { CDR:highlight_block_prepare('l_CDR_tags_clist') }

```

\l_CDR_bool is true iff one of the tags needs pygments.

```

1055 \bool_set_false:N \l_CDR_bool
1056 \clist_map_inline:Nn \l_CDR_tags_clist {
1057   \CDR_if_truthy:eT { \CDR_tag_get:cc { ##1 } { pygments } } {
1058     \clist_map_break:n { \bool_set_true:N \l_CDR_bool }
1059   }
1060 }
1061 \bool_if:NF \l_CDR_bool {
1062   \CDR_keys_inherit:Vnx \c_CDR_tag { __local } {
1063     \c_CDR_tag / __fancyvrb.all
1064   }
1065   \CDR_tag_keys_set_known:nVN { __local } \l_CDR_tl \l_CDR_tl
1066 }
1067 \CDR_check_unknown:N \l_CDR_tl
1068 \clist_set:Nx \l_CDR_clist {
1069   __block, default.block, default, __fancyvrb.block, __fancyvrb
1070 }
1071 \bool_if:NF \l_CDR_bool {
1072   \clist_put_right:Nx \l_CDR_clist { __fancyvrb.all }
1073 }
1074 \CDR_keys_inherit:VnV \c_CDR_tag_get { __local } \l_CDR_clist
1075
1076 \CDR_tag_get:cN {reflabel} \l_CDR_tl
1077 \exp_args:NV \label \l_CDR_tl
1078 ERROR \bool_if:nF { \clist_if_empty_p:n } {}
1079 \clist_if_empty:NF \l_CDR_tags_clist {
1080   \cs_map_inline:nn { i, ii, iii, iv } {
1081     \cs_set:cpn { FV@ListProcessLine@ #####1 } ##1 {
1082       \CDR_process_line:n { ##1 }
1083       \use:c { CDR@ListProcessLine@ #####1 } { ##1 }

```

```

1084     }
1085   }
1086 }
1087 \CDR_tag_get:cNF { engine } \l_CDR_engine_tl {
1088   \tl_set:Nn \l_CDR_engine_tl { default }
1089 }
1090 \CDR_tag_get:xNF { \l_CDR_engine_tl~engine~options } \l_CDR_tl {
1091   \tl_clear:N \l_CDR_tl
1092 }
1093 \exp_args:NnV
1094 \begin { \CDR_block_engine:V \l_CDR_engine_tl } \l_CDR_tl
1095 \FV@VerbatimBegin
1096 \FV@Scan
1097 }
1098 \def\FVE@CDRBlock{
1099   \FV@VerbatimEnd
1100   \end { \CDR_block_engine:V \l_CDR_engine_tl }
1101   \group_end:
1102   \@esphack
1103 }
1104 \DefineVerbatimEnvironment{CDRBlock}{CDRBlock}{}
1105

```

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.

```

1106 \def\FVB@CDR@Pyg@Verbatim #1 {
1107   \group_begin:
1108   \FV@VerbatimBegin
1109   \FV@Scan
1110 }
1111 \def\FVE@CDR@Pyg@Verbatim{
1112   \FV@VerbatimEnd
1113   \group_end:
1114 }
1115 \DefineVerbatimEnvironment{CDR@Pyg@Verbatim}{CDR@Pyg@Verbatim}{}
1116

```

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.

```

1117 \prg_new_conditional:Nnn \CDR_if_record: { T, F, TF } {
1118   \clist_if_empty:NTF \l_CDR_tags_clist {
1119     \prg_return_false:
1120   } {
1121     \CDR_if_use_pygments:TF {
1122       \prg_return_true:
1123     } {
1124       \prg_return_false:
1125     }
1126   }
1127 }

1128 \cs_new:Npn \CDR_process_recordNO: {
1129   \tl_put_right:Nx \l_CDR_recorded_tl { \the\verbatim@line \iow_newline: }
1130   \group_begin:
1131   \tl_set:Nx \l_tmpa_tl { \the\verbatim@line }
1132   \lua_now:e {CDR.records.append([==[\l_tmpa_tl]==])}
1133   \group_end:
1134 }
```

CDR `\begin{⟨CDR⟩} ... \end{⟨CDR⟩}`
Private environment.

```

1135 \newenvironment{CDR}{
1136   \def \verbatim@processline {
1137     \group_begin:
1138     \CDR_process_line_code_append:
1139     \group_end:
1140   }
1141   % \CDR_if_show_code:T {
1142   %   \CDR_if_use_minted:TF {
1143   %     \Needspace* { 2\baselineskip }
1144   %   } {
1145   %     \frenchspacing\@vobeyspaces
1146   %   }
1147   % }
1148 } {
1149   \CDR:nNTF { lang } \l_tmpa_tl {
1150     \tl_if_empty:NT \l_tmpa_tl {
1151       \clist_map_inline:Nn \l_CDR_clist {
1152         \CDR:nnNT { ##1 } { lang } \l_tmpa_tl {
1153           \tl_if_empty:NF \l_tmpa_tl {
1154             \clist_map_break:
1155           }

```

```

1156     }
1157   }
1158   \tl_if_empty:NT \l_tmpa_tl {
1159     \tl_set:Nn \l_tmpa_tl { tex }
1160   }
1161 }
1162 } {
1163   \tl_set:Nn \l_tmpa_tl { tex }
1164 }
1165 % NO WAY
1166 \clist_map_inline:Nn \l_CDR_clist {
1167   \CDR_gput:nnV { ##1 } { lang } \l_tmpa_tl
1168 }
1169 }

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

1170 \newenvironment{CDR_M}{
1171   \setkeys { FV } { firstnumber=last, }
1172   \clist_if_empty:NTF \l_CDR_clist {
1173     \exp_args:Nnx \setkeys { FV } {
1174       firstnumber=\CDR_int_use:n { },
1175     } } {
1176     \clist_map_inline:Nn \l_CDR_clist {
1177       \exp_args:Nnx \setkeys { FV } {
1178         firstnumber=\CDR_int_use:n { ##1 },
1179       }
1180     \clist_map_break:
1181   } }
1182   \iow_open:Nn \minted@code { \jobname.pyg }
1183   \tl_set:Nn \l_CDR_line_tl {
1184     \tl_set:Nx \l_tmpa_tl { \the\verbatim@line }
1185     \exp_args:NNV \iow_now:Nn \minted@code \l_tmpa_tl
1186   }
1187 } {
1188   \CDR_if_show_code:T {
1189     \CDR_if_use_minted:TF {
1190       \iow_close:N \minted@code
1191       \vspace* { \dimexpr -\topsep-\parskip }
1192       \tl_if_empty:NF \l_CDR_info_tl {
1193         \tl_use:N \l_CDR_info_tl
1194         \vspace* { \dimexpr -\topsep-\parskip-\baselineskip }
1195         \par\noindent
1196       }
1197       \exp_args:NV \minted@pygmentize \l_tmpa_tl
1198       \DeleteFile { \jobname.pyg }
1199       \vspace* { \dimexpr -\topsep -\partopsep }
1200     } {
1201       \@esphack
1202     }
1203   }
1204 }

CDR.P      \begin{<CDR.P>} ... \end{<CDR.P>}

```

Private pseudo environment. This is just a practical way of declaring balanced actions.

```

1205 \newenvironment{CDR_P}{
1206   \if_mode_vertical:
1207     \noindent
1208   \else
1209     \vspace*{ \topsep }
1210     \par\noindent
1211   \fi
1212   \CDR_gset_chunks:
1213   \tl_if_empty:NTF \g_CDR_chunks_tl {
1214     \CDR_if:nTF {show_lineno} {
1215       \CDR_if_use_margin:TF {

```

No chunk name, line numbers in the margin

```

1216       \tl_set:Nn \l_CDR_info_tl {
1217         \hbox_overlap_left:n {
1218           \CDR:n { format/code }
1219           {
1220             \CDR:n { format/name }
1221             \CDR:n { format/lineno }
1222             \clist_if_empty:NTF \l_CDR_clist {
1223               \CDR_int_use:n { }
1224             } {
1225               \clist_map_inline:Nn \l_CDR_clist {
1226                 \CDR_int_use:n { ##1 }
1227                 \clist_map_break:
1228               }
1229             }
1230           }
1231           \hspace*{1ex}
1232         }
1233       }
1234     } {

```

No chunk name, line numbers not in the margin

```

1235       \tl_set:Nn \l_CDR_info_tl {
1236         {
1237           \CDR:n { format/code }
1238           {
1239             \CDR:n { format/name }
1240             \CDR:n { format/lineno }
1241             \hspace*{3ex}
1242             \hbox_overlap_left:n {
1243               \clist_if_empty:NTF \l_CDR_clist {
1244                 \CDR_int_use:n { }
1245               } {
1246                 \clist_map_inline:Nn \l_CDR_clist {
1247                   \CDR_int_use:n { ##1 }
1248                   \clist_map_break:
1249                 }
1250               }

```

```

1251         }
1252         \hspace*{1ex}
1253     }
1254 }
1255 }
1256 }
1257 } {

```

No chunk name, no line numbers

```

1258     \tl_clear:N \l_CDR_info_tl
1259 }
1260 } {
1261     \CDR_if:nTF {show_lineno} {

```

Chunk names, line numbers, in the margin

```

1262     \tl_set:Nn \l_CDR_info_tl {
1263         \hbox_overlap_left:n {
1264             \CDR:n { format/code }
1265             {
1266                 \CDR:n { format/name }
1267                 \g_CDR_chunks_tl :
1268                 \hspace*{1ex}
1269                 \CDR:n { format/lineno }
1270                 \clist_map_inline:Nn \l_CDR_clist {
1271                     \CDR_int_use:n { ####1 }
1272                     \clist_map_break:
1273                 }
1274             }
1275             \hspace*{1ex}
1276         }
1277     \tl_set:Nn \l_CDR_info_tl {
1278         \hbox_overlap_left:n {
1279             \CDR:n { format/code }
1280             {
1281                 \CDR:n { format/name }
1282                 \CDR:n { format/lineno }
1283                 \clist_map_inline:Nn \l_CDR_clist {
1284                     \CDR_int_use:n { ####1 }
1285                     \clist_map_break:
1286                 }
1287             }
1288             \hspace*{1ex}
1289         }
1290     }
1291 }
1292 } {

```

Chunk names, no line numbers, in the margin

```

1293     \tl_set:Nn \l_CDR_info_tl {
1294         \hbox_overlap_left:n {
1295             \CDR:n { format/code }
1296             {
1297                 \CDR:n { format/name }

```



```

1298         \g_CDR_chunks_tl :
1299     }
1300     \hspace*{1ex}
1301 }
1302 \tl_clear:N \l_CDR_info_tl
1303 }
1304 }
1305 }
1306 \CDR_if_use_minted:F {
1307     \tl_set:Nn \l_CDR_line_tl {
1308         \noindent
1309         \hbox_to_wd:nn { \textwidth } {
1310             \tl_use:N \l_CDR_info_tl
1311             \CDR:n { format/code }
1312             \the\verbatim@line
1313             \hfill
1314         }
1315     }
1316 }
1317 \@bsphack
1318 }
1319 } {
1320     \vspace*{ \topsep }
1321     \par
1322     \@esphack
1323 }

```

18 Management

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

```
1324 \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 {⟨true code⟩} {⟨false code⟩}`

Execute *⟨true code⟩* when code should be printed, *⟨false code⟩* otherwise.

```

1325 \prg_new_conditional:Nnn \CDR_if_show_code: { T, F, TF } {
1326     \bool_if:nTF {
1327         \g_CDR_in_impl_bool && !\g_CDR_with_impl_bool
1328     } {
1329         \prg_return_false:
1330     } {
1331         \prg_return_true:
1332     }
1333 }

```

`\g_CDR_with_impl_bool`

```
1334 \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`.

```
1335 \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`.

```
1336 \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` `{⟨true code⟩}` `{⟨false code⟩}`

Execute `⟨true code⟩` when using minted, `⟨false code⟩` otherwise.

```
1337 \prg_new_conditional:Nnn \CDR_if_use_minted: { T, F, TF } {
1338   \bool_if:NTF \g_CDR_use_minted_bool
1339     { \prg_return_true: }
1340     { \prg_return_false: }
1341 }
```

`_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.

```
1342 \cs_set:Npn \_CDR_minted_on: {
1343   \bool_gset_true:N \g_CDR_minted_on_bool
1344   \RequirePackage{minted}
1345   \setkeys{ minted@opt@g } { linenos=false }
1346   \minted@def@opt{post~processor}
1347   \minted@def@opt{post~processor~args}
1348   \pretocmd\minted@inputpyg{
1349     \CDR@postprocesspyg {\minted@outputdir\minted@infile}
1350   }{\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

```
1351 \newcommand{\CDR@postprocesspyg}[1]{%
1352   \group_begin:
1353   \tl_set:Nx \l_tmpa_tl {\CDR:n { post_processor } }
1354   \tl_if_empty:NF \l_tmpa_tl {
```

Execute ‘`python3 <script.py> <file.pygtex> <more_args>`’

```

1355     \tl_set:Nx \l_tmpb_tl {\CDR:n { post_processor_args } }
1356     \exp_args:Nx
1357     \sys_shell_now:n {
1358       python3\space
1359       \l_tmpa_tl\space
1360       ##1\space
1361       \l_tmpb_tl
1362     }
1363   }
1364   \group_end:
1365 }
1366 }

1367 %\AddToHook { begindocument / end } {
1368 %   \cs_set_eq:NN \_CDR_minted_on: \prg_do_nothing:
1369 %}

```

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

```

1370 \ProvideDocumentCommand{\CDREmph}{m}{\textcolor{red}{#1}}

```

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

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

```

1371 \DeclareDocumentCommand \CDRPreamble { m m } {
1372   \msg_info:nnn
1373   { coder }
1374   { :n }
1375   { Reading-preamble-from-file-"#2". }
1376   \group_begin:
1377   \tl_set:Nn \l_tmpa_tl { #2 }
1378   \exp_args:NNNx
1379   \group_end:
1380   \tl_set:Nx #1 { \lua_now:n {CDR.print_file_content('l_tmpa_tl')} }
1381 }

```

20 Section separators

\CDRImplementation	\CDRImplementation
\CDRFinale	\CDRFinale

\CDRImplementation start an implementation part where all the sectioning commands do nothing, whereas \CDRFinale stop an implementation part.

21 Finale

```

1382 \newcounter{CDR@impl@page}
1383 \DeclareDocumentCommand \CDRImplementation {} {
1384   \bool_if:NF \g_CDR_with_impl_bool {
1385     \clearpage

```

```

1386 \bool_gset_true:N \g_CDR_in_impl_bool
1387 \let\CDR@old@part\part
1388 \DeclareDocumentCommand\part{som}{-}
1389 \let\CDR@old@section\section
1390 \DeclareDocumentCommand\section{som}{-}
1391 \let\CDR@old@subsection\subsection
1392 \DeclareDocumentCommand\subsection{som}{-}
1393 \let\CDR@old@subsubsection\subsubsection
1394 \DeclareDocumentCommand\subsubsection{som}{-}
1395 \let\CDR@old@paragraph\paragraph
1396 \DeclareDocumentCommand\paragraph{som}{-}
1397 \let\CDR@old@subparagraph\subparagraph
1398 \DeclareDocumentCommand\subparagraph{som}{-}
1399 \cs_if_exist:NT \refsection{ \refsection }
1400 \setcounter{ CDR@impl@page }{ \value{page} }
1401 }
1402 }
1403 \DeclareDocumentCommand\CDRFinale {} {
1404 \bool_if:NF \g_CDR_with_impl_bool {
1405 \clearpage
1406 \bool_gset_false:N \g_CDR_in_impl_bool
1407 \let\part\CDR@old@part
1408 \let\section\CDR@old@section
1409 \let\subsection\CDR@old@subsection
1410 \let\subsubsection\CDR@old@subsubsection
1411 \let\paragraph\CDR@old@paragraph
1412 \let\subparagraph\CDR@old@subparagraph
1413 \setcounter { page } { \value{ CDR@impl@page } }
1414 }
1415 }
1416 \cs_set_eq:NN \CDR_line_number: \prg_do_nothing:

```

22 Finale

```

1417 \AddToHook { cmd/FancyVerbFormatLine/before } {
1418 \CDR_line_number:
1419 }
1420 \AddToHook { shipout/before } {
1421 \tl_gclear:N \g_CDR_chunks_tl
1422 }

1423 % =====
1424 % Auxiliary:
1425 % finding the widest string in a comma
1426 % separated list of strings delimited by parenthesis
1427 % =====
1428
1429 % arguments:
1430 % #1) text: a comma separated list of strings
1431 % #2) formatter: a macro to format each string
1432 % #3) dimension: will hold the result
1433
1434 \cs_new:Npn \CDRWidest (#1) #2 #3 {

```

```

1435 \group_begin:
1436 \dim_set:Nn #3 { Opt }
1437 \clist_map_inline:nn { #1 } {
1438   \hbox_set:Nn \l_tmpa_box { #2{##1} }
1439   \dim_set:Nn \l_tmpa_dim { \dim_eval:n { \box_wd:N \l_tmpa_box } }
1440   \dim_compare:nNnT { #3 } < { \l_tmpa_dim } {
1441     \dim_set_eq:NN #3 \l_tmpa_dim
1442   }
1443 }
1444 \exp_args:NNNV
1445 \group_end:
1446 \dim_set:Nn #3 #3
1447 }
1448 \ExplSyntaxOff
1449

```

23 pygmentex implementation

```

1450 % =====
1451 % fancyvrb new commands to append to a file
1452 % =====
1453
1454 % See http://tex.stackexchange.com/questions/47462/inputenc-error-with-unicode-chars-and-verbatim
1455
1456 \ExplSyntaxOn
1457
1458 \seq_new:N \l_CDR_records_seq
1459
1460 \long\def\unexpanded@write#1#2{\write#1{\unexpanded{#2}}}
1461
1462 \def\CDRAppend{\FV@Environment{}}{\CDRAppend}}
1463
1464 \def\FVB@CDRAppend#1{%
1465   \@bsphack
1466   \begingroup
1467     \seq_clear:N \l_CDR_records_seq
1468     \FV@UseKeyValues
1469     \FV@DefineWhiteSpace
1470     \def\FV@Space{\space}%
1471     \FV@DefineTabOut
1472     \def\FV@ProcessLine{%##1
1473       \seq_put_right:Nn \l_CDR_records_seq { ##1 }%
1474       \immediate\unexpanded@write#1{##1}
1475     }%
1476     \let\FV@FontScanPrep\relax
1477     \let\@noligs\relax
1478     \FV@Scan
1479   }
1480 \def\FVE@CDRAppend{
1481   \seq_use:Nn \l_CDR_records_seq /
1482   \endgroup
1483   \@esphack
1484 }

```

```

1485 \DefineVerbatimEnvironment{CDRAppend}{CDRAppend}{}
1486
1487 \DeclareDocumentEnvironment { Inline } { m } {
1488   \clist_clear:N \l_CDR_clist
1489   \keys_set:nn { CDR_code } { #1 }
1490   \clist_map_inline:Nn \l_CDR_clist {
1491     \CDR_int_if_exist:nF { ##1 } {
1492       \CDR_int_new:nn { ##1 } { 1 }
1493       \seq_new:c { g/CDR/chunks/##1 }
1494     }
1495   }
1496   \CDR_if:nT {reset} {
1497     \CDR_clist_map_inline:Nnn \l_CDR_clist {
1498       \CDR_int_gset:nn { } 1
1499     } {
1500       \CDR_int_gset:nn { ##1 } 1
1501     }
1502   }
1503   \tl_clear:N \l_CDR_code_name_tl
1504   \clist_map_inline:Nn \l_CDR_clist {
1505     \prop_concat:ccc
1506       {g/CDR/Code/}
1507       {g/CDR/Code/##1/}
1508       {g/CDR/Code/}
1509     \tl_set:Nn \l_CDR_code_name_tl { ##1 }
1510     \clist_map_break:
1511   }
1512   \int_gset:Nn \g_CDR_int
1513   { \CDR_int_use:n { \l_CDR_code_name_tl } }
1514   \tl_clear:N \l_CDR_info_tl
1515   \tl_clear:N \l_CDR_name_tl
1516   \tl_clear:N \l_CDR_recorded_tl
1517   \tl_clear:N \l_CDR_chunks_tl
1518   \cs_set:Npn \verbatim@processline {
1519     \CDR_process_record:
1520   }
1521   \CDR_if_show_code:TF {
1522     \exp_args:NNx
1523     \skip_set:Nn \parskip { \CDR:n { parskip } }
1524     \clist_if_empty:NTF \l_CDR_clist {
1525       \tl_gclear:N \g_CDR_chunks_tl
1526     } {
1527       \clist_set_eq:NN \l_tmpa_clist \l_CDR_clist
1528       \clist_sort:Nn \l_tmpa_clist {
1529         \str_compare:nNnTF { ##1 } > { ##2 } {
1530           \sort_return_swapped:
1531         } {
1532           \sort_return_same:
1533         }
1534       }
1535       \tl_set:Nx \l_tmpa_tl { \clist_use:Nn \l_tmpa_clist , }
1536       \CDR_if:nT {show_name} {
1537         \CDR_if:nT {use_margin} {
1538           \CDR_if:nT {only_top} {

```

```

1539         \tl_if_eq:NNT \l_tmpa_tl \g_CDR_chunks_tl {
1540             \tl_gset_eq:NN \g_CDR_chunks_tl \l_tmpa_tl
1541             \tl_clear:N \l_tmpa_tl
1542         }
1543     }
1544     \tl_if_empty:NF \l_tmpa_tl {
1545         \tl_set:Nx \l_CDR_chunks_tl {
1546             \clist_use:Nn \l_CDR_clist ,
1547         }
1548         \tl_set:Nn \l_CDR_name_tl {
1549             {
1550                 \CDR:n { format/name }
1551                 \l_CDR_chunks_tl :
1552                 \hspace*{1ex}
1553             }
1554         }
1555     }
1556 }
1557 \tl_if_empty:NF \l_tmpa_tl {
1558     \tl_gset_eq:NN \g_CDR_chunks_tl \l_tmpa_tl
1559 }
1560 }
1561 }
1562 \if_mode_vertical:
1563 \else:
1564 \par
1565 \fi:
1566 \vspace{ \CDR:n { sep } }
1567 \noindent
1568 \frenchspacing
1569 \@vobeyspaces
1570 \normalfont\ttfamily
1571 \CDR:n { format/code }
1572 \hyphenchar\font\m@ne
1573 \@noligs
1574 \CDR_if_record:F {
1575     \cs_set_eq:NN \CDR_process_record: \prg_do_nothing:
1576 }
1577 \CDR_if_use_minted:F {
1578     \CDR_if:nT {show_lineno} {
1579         \CDR_if:nTF {use_margin} {
1580             \tl_set:Nn \l_CDR_info_tl {
1581                 \hbox_overlap_left:n {
1582                     {
1583                         \l_CDR_name_tl
1584                         \CDR:n { format/name }
1585                         \CDR:n { format/lineno }
1586                         \int_use:N \g_CDR_int
1587                         \int_gincr:N \g_CDR_int
1588                     }
1589                     \hspace*{1ex}
1590                 }
1591             }
1592         } {

```

```

1593     \tl_set:Nn \l_CDR_info_tl {
1594     {
1595         \CDR:n { format/name }
1596         \CDR:n { format/lineno }
1597         \hspace*{3ex}
1598         \hbox_overlap_left:n {
1599             \int_use:N \g_CDR_int
1600             \int_gincr:N \g_CDR_int
1601         }
1602     }
1603     \hspace*{1ex}
1604 }
1605 }
1606 }
1607 \cs_set:Npn \verbatim@processline {
1608     \CDR_process_record:
1609     \hspace*{\dimexpr \linewidth-\columnwidth}%
1610     \hbox_to_wd:nn { \columnwidth } {
1611         \l_CDR_info_tl
1612         \the\verbatim@line
1613         \color{lightgray}\dotfill
1614     }
1615     \tl_clear:N \l_CDR_name_tl
1616     \par\noindent
1617 }
1618 }
1619 } {
1620     \@bsphack
1621 }
1622 \group_begin:
1623 \g_CDR_hook_tl
1624 \let \do \@makeother
1625 \dospecials \catcode '\^M \active
1626 \verbatim@start
1627 } {
1628     \int_gsub:Nn \g_CDR_int {
1629         \CDR_int_use:n { \l_CDR_code_name_tl }
1630     }
1631     \int_compare:nNnT { \g_CDR_int } > { 0 } {
1632         \CDR_clist_map_inline:Nnn \l_CDR_clist {
1633             \CDR_int_gadd:nn { } { \g_CDR_int }
1634         } {
1635             \CDR_int_gadd:nn { ##1 } { \g_CDR_int }
1636         }
1637         \int_gincr:N \g_CDR_code_int
1638         \tl_set:Nx \l_tmpb_tl { \int_use:N \g_CDR_code_int }
1639         \clist_map_inline:Nn \l_CDR_clist {
1640             \seq_gput_right:cV { g/CDR/chunks/##1 } \l_tmpb_tl
1641         }
1642         \prop_gput:NVV \g_CDR_code_prop \l_tmpb_tl \l_CDR_recorded_tl
1643     }
1644 \group_end:
1645 \CDR_if_show_code:T {
1646 }

```



```

1647 \CDR_if_show_code:TF {
1648   \CDR_if_use_minted:TF {
1649     \tl_if_empty:NF \l_CDR_recorded_tl {
1650       \exp_args:Nnx \setkeys { FV } {
1651         firstnumber=\CDR_int_use:n { \l_CDR_code_name_tl },
1652       }
1653       \iow_open:Nn \minted@code { \jobname.pyg }
1654       \exp_args:NNV \iow_now:Nn \minted@code \l_CDR_recorded_tl
1655       \iow_close:N \minted@code
1656       \vspace* { \dimexpr -\topsep-\parskip }
1657       \tl_if_empty:NF \l_CDR_info_tl {
1658         \tl_use:N \l_CDR_info_tl
1659         \skip_vertical:n { \dimexpr -\topsep-\parskip-\baselineskip }
1660         \par\noindent
1661       }
1662       \exp_args:Nnx \minted@pygmentize { \jobname.pyg } { \CDR:n { lang } }
1663       %\DeleteFile { \jobname.pyg }
1664       \skip_vertical:n { -\topsep-\partopsep }
1665     }
1666   } {
1667     \exp_args:Nx \skip_vertical:n { \CDR:n { sep } }
1668     \noindent
1669   }
1670 } {
1671   \@esphack
1672 }
1673 }
1674 % =====
1675 % Main options
1676 % =====
1677
1678 \newif\ifCDR@left
1679 \newif\ifCDR@right
1680
1681

```

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 colorized 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.

```

1682 \cs_new:Npn \CDRCallWithOptions #1 {
1683   \exp_last_unbraced:NNx
1684   #1[\CDR:n { options }]
1685 }
1686 \CDRNewCodeEngine {efbox} {
1687   \CDRCallWithOptions\efbox{#1}%
1688 }

```

24.2 Block mode default engine

```

1689 \CDRNewBlockEngine {} {
1690 } {
1691 }

```

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

```

1692
1693 % =====
1694 % pygmented commands and environments
1695 % =====
1696
1697
1698 \newcommand\inputpygmented[2] [] {
1699   \begin{group}
1700     \CDR@process@options{#1}%
1701     \immediate\write\CDR@outfile{<@@CDR@input@the\CDR@counter}%
1702     \immediate\write\CDR@outfile{\exp_args:NV\detokenize\CDR@global@options,\detokenize{#1}}%
1703     \immediate\write\CDR@outfile{#2}%
1704     \immediate\write\CDR@outfile{>@@CDR@input@the\CDR@counter}%
1705     %
1706     \csname CDR@snippet@the\CDR@counter\endcsname
1707     \global\advance\CDR@counter by 1\relax
1708   \end{group}
1709 }
1710
1711 \cs_generate_variant:Nn \exp_last_unbraced:NnNo { NxNo }
1712
1713 \newcommand\CDR@snippet@run[1] {
1714   \group_begin:
1715   \typeout{DEBUG~PY~STYLE:< \CDR:n { style } > }
1716   \use_c:n { PYstyle }
1717   \CDR_when:nT { style } {

```

```

1718 \use_c:n { PYstyle \CDR:n { style } }
1719 }
1720 \cs_if_exist:cTF {PY} {PYOK} {PYKO}
1721 \CDR:n {font}
1722 \CDR@process@more@options{ \CDR:n {engine} }%
1723 \exp_last_unbraced:NxNo
1724 \use:c { \CDR:n {engine} } [ \CDRRemainingOptions ]{#1}%
1725 \group_end:
1726 }
1727
1728 % ERROR: JL undefined \CDR@alllinenos
1729
1730 \ProvideDocumentCommand\captionof{mm}{-}
1731 \def\CDR@alllinenos{(0)}
1732
1733 \def\FormatLineNumber#1{{\rmfamily\tiny#1}}
1734
1735 \newdimen\CDR@leftmargin
1736 \newdimen\CDR@linenosep
1737
1738 \def\CDR@lineno@do#1{%
1739 \CDR@linenosep 0pt%
1740 \use:c { CDR@ \CDR:n {block_engine} @margin }
1741 \exp_args:NNx
1742 \advance \CDR@linenosep { \CDR:n {linenosep} }
1743 \hbox_overlap_left:n {%
1744 \FormatLineNumber{#1}%
1745 \hspace*{\CDR@linenosep}%
1746 }%
1747 }
1748
1749 \newcommand\CDR@tcbox@more@options{%
1750 nobeforeafter,%
1751 tcbox-raise-base,%
1752 left=0mm,%
1753 right=0mm,%
1754 top=0mm,%
1755 bottom=0mm,%
1756 boxsep=2pt,%
1757 arc=1pt,%
1758 boxrule=0pt,%
1759 \CDR_options_if_in:nT {colback} {
1760 colback=\CDR:n {colback}
1761 }
1762 }
1763
1764 \newcommand\CDR@mdframed@more@options{%
1765 leftmargin=\CDR@leftmargin,%
1766 frametitleule=true,%
1767 \CDR_if_in:nT {colback} {
1768 backgroundcolor=\CDR:n {colback}
1769 }
1770 }
1771

```

```

1772 \newcommand\CDR@tcolorbox@more@options{%
1773   grow~to~left~by=-\CDR@leftmargin,%
1774   \CDR_if_in:nNT {colback} {
1775     colback=\CDR:n {colback}
1776   }
1777 }
1778
1779 \newcommand\CDR@boite@more@options{%
1780   leftmargin=\CDR@leftmargin,%
1781   \ifcsname CDR@opt@colback\endcsname
1782     colback=\CDR@opt@colback,%
1783   \fi
1784 }
1785
1786 \newcommand\CDR@mdframed@margin{%
1787   \advance \CDR@linenosep \mdflength{outerlinewidth}%
1788   \advance \CDR@linenosep \mdflength{middlelinewidth}%
1789   \advance \CDR@linenosep \mdflength{innerlinewidth}%
1790   \advance \CDR@linenosep \mdflength{innerleftmargin}%
1791 }
1792
1793 \newcommand\CDR@tcolorbox@margin{%
1794   \advance \CDR@linenosep \kvtcb@left@rule
1795   \advance \CDR@linenosep \kvtcb@left@upper
1796   \advance \CDR@linenosep \kvtcb@boxsep
1797 }
1798
1799 \newcommand\CDR@boite@margin{%
1800   \advance \CDR@linenosep \boite@leftrule
1801   \advance \CDR@linenosep \boite@boxsep
1802 }
1803
1804 \def\CDR@global@options{}
1805
1806 \newcommand\setpygmented[1]{%
1807   \def\CDR@global@options{/CDR.cd,#1}%
1808 }
1809

```

26 Counters

`\CDR_int_new:nn` `\CDR_int_new:n {<name>} {<value>}`

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

```

1810 \cs_new:Npn \CDR_int_new:nn #1 #2 {
1811   \int_new:c {g/CDR/int/#1}
1812   \int_gset:cn {g/CDR/int/#1} { #2 }
1813 }

```

<u>\CDR_int_set:nn</u>	\CDR_int_set:n {<name>} {<value>}
<u>\CDR_int_gset:nn</u>	Set the integer named after <name> to the <value>. \CDR_int_gset:n makes a global change. <name> is a code name.

```

1814 \cs_new:Npn \CDR_int_set:nn #1 #2 {
1815   \int_set:cn {g/CDR/int/#1} { #2 }
1816 }
1817 \cs_new:Npn \CDR_int_gset:nn #1 #2 {
1818   \int_gset:cn {g/CDR/int/#1} { #2 }
1819 }

```

<u>\CDR_int_add:nn</u>	\CDR_int_add:n {<name>} {<value>}
<u>\CDR_int_gadd:nn</u>	Add the <value> to the integer named after <name>. \CDR_int_gadd:n makes a global change. <name> is a code name.

```

1820 \cs_new:Npn \CDR_int_add:nn #1 #2 {
1821   \int_add:cn {g/CDR/int/#1} { #2 }
1822 }
1823 \cs_new:Npn \CDR_int_gadd:nn #1 #2 {
1824   \int_gadd:cn {g/CDR/int/#1} { #2 }
1825 }

```

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

```

1826 \cs_new:Npn \CDR_int_sub:nn #1 #2 {
1827   \int_sub:cn {g/CDR/int/#1} { #2 }
1828 }
1829 \cs_new:Npn \CDR_int_gsub:nn #1 #2 {
1830   \int_gsub:cn {g/CDR/int/#1} { #2 }
1831 }

```

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

```

1832 \prg_new_conditional:Nnn \CDR_int_if_exist:n { T, F, TF } {
1833   \int_if_exist:cTF {g/CDR/int/#1} {
1834     \prg_return_true:
1835   } {
1836     \prg_return_false:
1837   }
1838 }

```

\g/CDR/int/ Generic and named line number counter. \l_CDR_code_name_t is used as <name>.

```

\g/CDR/int/<name>
1839 \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.

```
1840 \cs_new:Npn \CDR_int_use:n #1 {  
1841   \int_use:c {g/CDR/int/#1}  
1842 }
```

```
1843 \ExplSyntaxOff
```

```
1844 %</sty>
```