

`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 Lua \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. Normally, 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. As such, there is no need to explicitly expand such an argument.

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 `pygments`.

`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 `\CDRCodeEngineNew` or `\CDRBlockEngineNew`.

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_source`,
2. `coder-util.lua` reads the content of some command, and stores 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.

The package `coder.sty` only exchanges with `coder-util.lua` using `\directlua` and `tex.print`. `coder-tool.py` in turn only exchanges with `coder-util.lua`: we put in `coder-tool.py` as few L^AT_EX logic as possible. It receives instructions from `coder.sty` as command line arguments, L^AT_EX 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 `\CDRCodeEngineNew` and `\CDRBlockEngineNew`.

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 L^AT_EX **\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, `$...$` 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.
- **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.

All the line templates below are L^AT_EX 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 md5    = _ENV.md5
6 local kpse   = _ENV.kpse
7 local rep    = string.rep
8 local lpeg   = require("lpeg")
9 local P, Cg, Cp, V = lpeg.P, lpeg.Cg, lpeg.Cp, lpeg.V
10 local json  = require('lualibs-util-jsn')
```

3 General purpose material

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

```
11 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'`.

```
12 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.


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

is_truthy if CDR.is_truthy(*<string>*) then
<true code>
else
<false code>
end

Execute *<false code>* if *<string>* is the string “false”, *<true code>* otherwise.

```
23 local function is_truthy(s)
24   return s ~= 'false'
25 end
```

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

 Escape the given string to be used by the shell.

```
26 local function escape(s)
27   s = s:gsub(' ','\\ ')
28   s = s:gsub('\\','\\\\')
29   s = s:gsub('\\r','\\r')
30   s = s:gsub('\\n','\\n')
31   s = s:gsub('"','\\"')
32   s = s:gsub("'",'\\\'')
33   return s
34 end
```

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

Make a directory at the given path.


```

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

```

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

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

```

56 local dir_p, json_p
57 local jobname = tex.jobname
58 dir_p = './..jobname..'pygd/'
59 if make_directory(dir_p) == nil then
60   dir_p = './'
61   json_p = dir_p..jobname..'pyg.json'
62 else
63   json_p = dir_p..'input.pyg.json'
64 end

```

print_file_content CDR.print_file_content($\langle macro name \rangle$)

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

```

65 local function print_file_content(name)
66   local p = token.get_macro(name)
67   local fh = assert(io.open(p, 'r'))
68   local s = fh:read('a')
69   fh:close()
70   tex.print(s)
71 end

```

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

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

```
72 local eq_pattern = P({ Cp() * P('=')^1 * Cp() + P(1) * V(1) })
73 local function safe_equals(s)
74   local i, j = 0, 0
75   local max = 0
76   while true do
77     i, j = eq_pattern:match(s, j)
78     if i == nil then
79       return rep('=', max + 1)
80     end
81     i = j - i
82     if i > max then
83       max = i
84     end
85   end
86 end
```

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

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

```
87 local function load_exec(self, chunk)
88   local env = setmetatable({ self = self, tex = tex }, _ENV)
89   local func, err = load(chunk, 'coder-tool', 't', env)
90   if func then
91     local ok
92     ok, err = pcall(func)
93     if not ok then
94       print("coder-util.lua Execution error:", err)
95       print('chunk:', chunk)
96     end
97   else
98     print("coder-util.lua Compilation error:", err)
99     print('chunk:', chunk)
100   end
101 end
```

load_exec_output

CDR:load_exec_output(*lua code chunk*)

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

?TEX:*TeX instructions* the *TeX instructions* are executed asynchronously once the control comes back to T_EX.

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

?LUA:*?Lua instructions* these *?Lua instructions* 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 *?TeX instructions* or *?Lua instructions* completes.

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

4 Properties

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

5 Hiligting

5.1 Common

highlight_set CDR:highlight_set(...)

Highlight the currently entered block. 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.

```
131 local function highlight_set(self, key, value)
132   local args = self['.arguments']
133   local t = args
134   if t[key] == nil then
135     t = args.pygopts
136     if t[key] == nil then
137       t = args.texopts
138       assert(t[key] ~= nil)
139     end
140   end
141   t[key] = value
142 end
143
144 local function highlight_set_var(self, key, var)
145   self:highlight_set(key, assert(token.get_macro(var or 'l_CDR_t1')))
146 end
```

highlight_source CDR:highlight_source(<src>, <sty>)

Highlight the currently entered block if <src> is true, build the style definitions if <sty> is true. 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.

```
147 local function highlight_source(self, sty, src)
148   local args = self['.arguments']
149   local texopts = args.texopts
150   local pygopts = args.pygopts
151   local inline = texopts.is_inline
152   local use_cache = self.is_truthy(args.cache)
153   local use_py = false
154   local cmd = self.PYTHON_PATH..' '..self.CDR_PY_PATH
155   local debug = args.debug
156   local pyg_sty_p
157   if sty then
158     pyg_sty_p = dir_p..pygopts.style..'pyg.sty'
159     texopts.pyg_sty_p = pyg_sty_p
160     local mode,_,_ = lfs.attributes(pyg_sty_p, 'mode')
161     if not mode or not use_cache then
162       use_py = true
163       if debug then
164         print('PYTHON STYLE:')
165       end
166       cmd = cmd..' --create_style'
167     end
```

```

168     self:cache_record(pyg_sty_p)
169 end
170 local pyg_tex_p
171 if src then
172     local source
173     if inline then
174         source = args.source
175     else
176         local ll = self['.lines']
177         source = table.concat(ll, '\n')
178     end
179     local base = dir_p..md5.sumhexa( ('s:s:s'
180     ):format(
181         source,
182         inline and 'code' or 'block',
183         pygopts.style
184     )
185 )
186 pyg_tex_p = base..'pyg.tex'
187 local mode,_,_ = lfs.attributes(pyg_tex_p,'mode')
188 if not mode or not use_cache then
189     use_py = true
190     if debug then
191         print('PYTHON SOURCE:', inline)
192     end
193     if not inline then
194         local tex_p = base..'tex'
195         local f = assert(io.open(tex_p, 'w'))
196         local ok, err = f:write(source)
197         f:close()
198         if not ok then
199             print('File error('..tex_p..'): '..err)
200         end
201         if debug then
202             print('OUTPUT: '..tex_p)
203         end
204     end
205     cmd = cmd..(' --base=%q'):format(base)
206 end
207 end
208 if use_py then
209     local json_p = self.json_p
210     local f = assert(io.open(json_p, 'w'))
211     local ok, err = f:write(json.tostring(args, true))
212     f:close()
213     if not ok then
214         print('File error('..json_p..'): '..err)
215     end
216     cmd = cmd..(' %q'):format(json_p)
217     if debug then
218         print('CDR>'..cmd)
219     end
220     local o = io.popen(cmd):read('a')
221     if debug then

```

```

222     print('PYTHON', o)
223     end
224 end
225 self:cache_record(
226     sty and pyg_sty_p or nil,
227     src and pyg_tex_p or nil
228 )
229 cmd = [= ''
230 if sty then
231     cmd = [[\CDR@StyleInput{}}..pyg_sty_p..[{}]]
232 end
233 if src then
234     cmd = cmd..[[\CDR@SourceInput{}}..pyg_tex_p..[{}]]
235 end
236 if #cmd > 0 then
237     cmd = [[\makeatletter}}..cmd..[\makeatother]]
238     tex.print(cmd)
239 end
240 ]=]
241 if sty then
242     cmd = [{}}..pyg_sty_p..[{}]]
243 else
244     cmd = '{} '
245 end
246 if src then
247     cmd = cmd..[{}}..pyg_tex_p..[{}]]
248 else
249     cmd = cmd..'{} '
250 end
251 if #cmd > 4 then
252     cmd = [[\makeatletter\CDR@Callback}}..cmd..[\makeatother]]
253     tex.print(cmd)
254 end
255 if debug then
256     print('CDR<'..cmd)
257 end
258 end

```

5.2 Code

highlight_code_prepare CDR:highlight_code_prepare()

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.

```

259 local function highlight_code_prepare(self)
260     self['.arguments'] = {
261         __cls__ = 'Arguments',
262         source = '',
263         cache = true,
264         debug = false,
265         pygopts = {

```

```

266     __cls__ = 'PygOpts',
267     lang    = 'tex',
268     style   = 'default',
269 },
270 texopts = {
271     __cls__ = 'TeXOpts',
272     tags    = '',
273     is_inline = true,
274     pyg_sty_p = '',
275 },
276 fv_opts = {
277     __cls__ = 'FVOpts',
278 }
279 }
280 self.highlight_json_written = false
281 end
282

```

5.3 Block

highlight_block_prepare CDR:highlight_block_prepare(*(tags_clist var)*)

Records the contents of the *(tags_clist var)* L^AT_EX variable to prepare block highlighting.

```

283 local function highlight_block_prepare(self, tags_clist_var)
284   local tags_clist = assert(token.get_macro(assert(tags_clist_var)))
285   local t = {}
286   for tag in string.gmatch(tags_clist, '([^\,]+)') do
287     t[#t+1]=tag
288   end
289   self['.tags_clist'] = tags_clist
290   self['.block_tags'] = t
291   self['.lines'] = {}
292   self['.arguments'] = {
293     __cls__ = 'Arguments',
294     cache   = false,
295     debug   = false,
296     source  = nil,
297     pygopts = {
298       __cls__ = 'PygOpts',
299       lang    = 'tex',
300       style   = 'default',
301     },
302     texopts = {
303       __cls__ = 'TeXOpts',
304       tags    = tags_clist,
305       is_inline = false,
306       pyg_sty_p = '',
307     },
308     fv_opts = {
309       __cls__ = 'FVOpts',
310     }
311   }

```

```

312 self.hilight_json_written = false
313 end
314

```

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

Store the content of the given named variable.

```

315 local function record_line(self, line_variable_name)
316   local line = assert(token.get_macro(assert(line_variable_name)))
317   local ll = assert(self['.lines'])
318   ll[#ll+1] = line
319   local lt = self['lines by tag'] or {}
320   self['lines by tag'] = lt
321   for _,tag in ipairs(self['.block tags']) do
322     ll = lt[tag] or {}
323     lt[tag] = ll
324     ll[#ll+1] = line
325   end
326 end

```

hilight_advance CDR:hilight_advance(*<count>*)

<count> is the number of line hilighted.

```

327 local function hilight_advance(self, count)
328 end

```

6 Exportation

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

export_file CDR:export_file(*<file name var>*)

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

```

329 local function export_file(self, file_name)
330   self['.name'] = assert(token.get_macro(assert(file_name)))
331   self['.export'] = {}
332 end

```

export_file_info CDR:export_file_info(*<key>*, *<value name var>*)

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

```

333 local function export_file_info(self, key, value)
334   local export = self['.export']
335   value = assert(token.get_macro(assert(value)))
336   export[key] = value
337 end

```

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

This is called at export time.

```

338 local function export_complete(self)
339     local name      = self['.name']
340     local export    = self['.export']
341     local records   = self['.records']
342     local tt = {}
343     local s = export.preamble
344     if s then
345         tt[#tt+1] = s
346     end
347     for _,tag in ipairs(export.tags) do
348         s = records[tag]:concat('\n')
349         tt[#tt+1] = s
350         records[tag] = { [1] = s }
351     end
352     s = export.postamble
353     if s then
354         tt[#tt+1] = s
355     end
356     if #tt>0 then
357         local fh = assert(io.open(name,'w'))
358         fh:write(tt:concat('\n'))
359         fh:close()
360     end
361     self['.file'] = nil
362     self['.exportation'] = nil
363 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.

```

364 local function cache_clean_all(self)
365     local to_remove = {}

```

```

366   for f in lfs.dir(dir_p) do
367       to_remove[f] = true
368   end
369   for k,_ in pairs(to_remove) do
370       os.remove(dir_p .. k)
371   end
372 end
373 local function cache_record(self, pyg_sty_p, pyg_tex_p)
374   if pyg_sty_p then
375       self['.style_set'] [pyg_sty_p] = true
376   end
377   if pyg_tex_p then
378       self['.colored_set'] [pyg_tex_p] = true
379   end
380 end
381 local function cache_clean_unused(self)
382   local to_remove = {}
383   for f in lfs.dir(dir_p) do
384       f = dir_p .. f
385       if not self['.style_set'][f] and not self['.colored_set'][f] then
386           to_remove[f] = true
387       end
388   end
389   for f,_ in pairs(to_remove) do
390       os.remove(f)
391   end
392 end

```

_DESCRIPTION Short text description of the module.

```

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

```

8 Return the module

```

394 return {
      Known fields are
395   _DESCRIPTION      = _DESCRIPTION,
      _VERSION to store ⟨version string⟩,
396   _VERSION          = token.get_macro('fileversion'),
      date to store ⟨date string⟩,
397   date              = token.get_macro('filedate'),
      Various paths ,

```

```

398 CDR_PY_PATH      = CDR_PY_PATH,
399 PYTHON_PATH      = PYTHON_PATH,
400 set_python_path   = set_python_path,

    is_truthy

401 is_truthy        = is_truthy,

    escape

402 escape           = escape,

    make_directory

403 make_directory   = make_directory,

    load_exec

404 load_exec         = load_exec,

405 load_exec_output = load_exec_output,

    record_line

406 record_line      = record_line,

    highlight common

407 highlight_set     = highlight_set,
408 highlight_set_var = highlight_set_var,
409 highlight_source   = highlight_source,
410 highlight_advance  = highlight_advance,

    highlight code

411 highlight_code_prepare = highlight_code_prepare,

    highlight_block_prepare

412 highlight_block_prepare = highlight_block_prepare,

    cache_clean_all

413 cache_clean_all    = cache_clean_all,

    cache_record

414 cache_record       = cache_record,

    cache_clean_unused

415 cache_clean_unused = cache_clean_unused,

```

Internals

```
416  ['.style_set']      = {},
417  ['.colored_set']    = {},
418  ['.options']        = {},
419  ['.export']         = {},
420  ['.name']           = nil,
```

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

```
421  already              = false,
```

Other

```
422  json_p               = json_p,
```

```
423 }
```

```
424 %</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 json
```

```

16 from pygments import highlight as hilight
17 from pygments.formatters.latex import LatexEmbeddedLexer, LatexFormatter
18 from pygments.lexers import get_lexer_by_name
19 from pygments.util import ClassNotFound

```

3 Options classes

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

```

20 class BaseOpts(object):
21     @staticmethod
22     def ensure_bool(x):
23         if x == True or x == False: return x
24         x = x[0:1]
25         return x == 'T' or x == 't'
26
27     def __init__(self, d={}):
28         for k, v in d.items():
29             if type(v) == str:
30                 if v.lower() == 'true':
31                     setattr(self, k, True)
32                     continue
33                 elif v.lower() == 'false':
34                     setattr(self, k, False)
35                     continue
36             setattr(self, k, v)

```

3.1 TeXOpts class

```

36 class TeXOpts(BaseOpts):
37     tags = ''
38     is_inline = True
39     pyg_sty_p = None

```

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

```

40 sty_template=r'''% !TeX root=...
41 \makeatletter
42 \CDR@StyleDefine{<placeholder:style_name>} {%
43   <placeholder:style_defs>}%
44 \makeatother'''
45 single_line_template = r'''\CDR@Line{Single}{<placeholder:number>}{<placeholder:line>}'
46 first_line_template = r'''\CDR@Line{First}{<placeholder:number>}{<placeholder:line>}'
47 second_line_template = r'''\CDR@Line{Second}{<placeholder:number>}{<placeholder:line>}'
48 white_line_template = r'''\CDR@Line{White}{<placeholder:number>}{<placeholder:line>}'
49 black_line_template = r'''\CDR@Line{Black}{<placeholder:number>}{<placeholder:line>}'
50 def __init__(self, *args, **kwargs):
51     super().__init__(*args, **kwargs)
52     self.inline_p = self.ensure_bool(self.is_inline)
53     self.pyg_sty_p = Path(self.pyg_sty_p or '')

```

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.

```
54 class PygOpts(BaseOpts):
55     style = 'default'
56     nobackground = False
57     linenos = False
58     linenostart = 1
59     linenostep = 1
60     commandprefix = 'Py'
61     texcomments = False
62     mathescape = False
63     escapeinside = ""
64     envname = 'Verbatim'
65     lang = 'tex'
66     def __init__(self, *args, **kwargs):
67         super().__init__(*args, **kwargs)
68         self.linenos = self.ensure_bool(self.linenos)
69         self.linenostart = abs(int(self.linenostart))
70         self.linenostep = abs(int(self.linenostep))
71         self.texcomments = self.ensure_bool(self.texcomments)
72         self.mathescape = self.ensure_bool(self.mathescape)
```

3.3 FVclass

```
73 class FVOpts(BaseOpts):
74     gobble = 0
75     tabsize = 4
76     linenosep = 'Opt'
77     commentchar = ''
78     frame = 'none'
79     label = ''
80     labelposition = 'none'
81     numbers = 'left'
82     numbersep = 'lex'
83     firstnumber = 'auto'
84     stepnumber = 1
85     numberblanklines = True
86     firstline = ''
87     lastline = ''
88     baselinestretch = 'auto'
89     resetmargins = True
90     xleftmargin = 'Opt'
91     xrightmargin = 'Opt'
92     hfuzz = '2pt'
93     samepage = False
94     def __init__(self, *args, **kwargs):
95         super().__init__(*args, **kwargs)
96         self.gobble = abs(int(self.gobble))
97         self.tabsize = abs(int(self.tabsize))
98         if self.firstnumber != 'auto':
```

```

99     self.firstnumber = abs(int(self.firstnumber))
100    self.stepnumber = abs(int(self.stepnumber))
101    self.numberblanklines = self.ensure_bool(self.numberblanklines)
102    self.resetmargins = self.ensure_bool(self.resetmargins)
103    self.samepage = self.ensure_bool(self.samepage)

```

3.4 Argumentsclass

```

104 class Arguments(BaseOpts):
105     cache = False
106     debug = False
107     source = ""
108     style = "default"
109     json = ""
110     directory = "."
111     texopts = TeXOpts()
112     pygopts = PygOpts()
113     fv_opts = FV0pts()

```

4 Controller main class

```
114 class Controller:
```

4.1 Static methods

<code>object_hook</code>	Helper for json parsing.
--------------------------	--------------------------

```

115 @staticmethod
116 def object_hook(d):
117     __cls__ = d.get('__cls__', 'Arguments')
118     if __cls__ == 'PygOpts':
119         return PygOpts(d)
120     elif __cls__ == 'FVOpts':
121         return FVOpts(d)
122     elif __cls__ == 'TeXOpts':
123         return TeXOpts(d)
124     else:
125         return Arguments(d)

```

lua_command	self.lua_command(<i>(asynchronous lua command)</i>)
lua_command_now	self.lua_command_now(<i>(synchronous lua command)</i>)
lua_debug	Wraps the given command between markers. It will

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

```

126 @staticmethod
127 def lua_command(cmd):
128     print(f'<<<<*LUA:{cmd}>>>>')
129 @staticmethod
130 def lua_command_now(cmd):

```

```

131     print(f'<<<<!LUA:{cmd}>>>>')
132     @staticmethod
133     def lua_debug(msg):
134         print(f'<<<<?LUA:{msg}>>>>')

```

```

lua_text_escape self.lua_text_escape((text))

```

Wraps the given command between [=...=[and]=...=] with as many equal signs as necessary to ensure a correct lua syntax.

```

135     @staticmethod
136     def lua_text_escape(s):
137         k = 0
138         for m in re.findall('+=', s):
139             if len(m) > k: k = len(m)
140         k = (k + 1) * "="
141         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 ??.)

```

142     _json_p = None
143     @property
144     def json_p(self):
145         p = self._json_p
146         if p:
147             return p
148         else:
149             p = self.arguments.json
150             if p:
151                 p = Path(p).resolve()
152             self._json_p = p
153         return p

```

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

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

```

154     @property
155     def parser(self):
156         parser = argparse.ArgumentParser(
157             prog=sys.argv[0],
158             description='''
159 Writes to the output file a set of LaTeX macros describing
160 the syntax highlighting of the input file as given by pygments.
161 '''
162         )
163         parser.add_argument(
164             "-v", "--version",
165             help="Print the version and exit",
166             action='version',

```



```

167     version=f'coder-tool version {__version__},'
168     ' (c) {__YEAR__} by Jérôme LAURENS.'
169 )
170 parser.add_argument(
171     "--debug",
172     action='store_true',
173     default=None,
174     help="display informations useful for debugging"
175 )
176 parser.add_argument(
177     "--create_style",
178     action='store_true',
179     default=None,
180     help="create the style definitions"
181 )
182 parser.add_argument(
183     "--base",
184     action='store',
185     default=None,
186     help="the path of the file to be colored, with no extension"
187 )
188 parser.add_argument(
189     "json",
190     metavar="<json data file>",
191     help=""
192 file name with extension, contains processing information.
193 """
194 )
195 return parser
196

```

4.3 Methods

4.3.1 __init__

__init__ Constructor. Reads the command line arguments.

```

197 def __init__(self, argv = sys.argv):
198     argv = argv[1:] if re.match(".*coder\-.tool\.py$", argv[0]) else argv
199     ns = self.parser.parse_args(
200         argv if len(argv) else ['-h']
201     )
202     with open(ns.json, 'r') as f:
203         self.arguments = json.load(
204             f,
205             object_hook = Controller.object_hook
206         )
207     args = self.arguments
208     args.json = ns.json
209     self.texopts = args.texopts
210     pygopts = self.pygopts = args.pygopts
211     fv_opts = self.fv_opts = args.fv_opts

```

```

212     self.formatter = LatexFormatter(
213         style = pygopts.style,
214         nobackground = pygopts.nobackground,
215         commandprefix = pygopts.commandprefix,
216         texcomments = pygopts.texcomments,
217         mathescape = pygopts.mathescape,
218         escapeinside = pygopts.escapeinside,
219         envname = 'CDR@Pyg@Verbatim',
220     )
221
222     try:
223         lexer = self.lexer = get_lexer_by_name(pygopts.lang)
224     except ClassNotFound as err:
225         sys.stderr.write('Error: ')
226         sys.stderr.write(str(err))
227
228     escapeinside = pygopts.escapeinside
229     # When using the LaTeX formatter and the option 'escapeinside' is
230     # specified, we need a special lexer which collects escaped text
231     # before running the chosen language lexer.
232     if len(escapeinside) == 2:
233         left = escapeinside[0]
234         right = escapeinside[1]
235         lexer = self.lexer = LatexEmbeddedLexer(left, right, lexer)
236
237     gobble = fv_opts.gobble
238     if gobble:
239         lexer.add_filter('gobble', n=gobble)
240     tabsize = fv_opts.tabsize
241     if tabsize:
242         lexer.tabsize = tabsize
243     lexer.encoding = ''
244     args.base = ns.base
245     args.create_style = ns.create_style
246     if ns.debug:
247         args.debug = True
248     # IN PROGRESS: support for extra keywords
249     # EXTRA_KEYWORDS = set(('foo', 'bar', 'foobar', 'barfoo', 'spam', 'eggs'))
250     # def over(self, text):
251     #     for index, token, value in lexer.__class__.get_tokens_unprocessed(self, text):
252     #         if token is Name and value in EXTRA_KEYWORDS:
253     #             yield index, Keyword.Pseudo, value
254     #         else:
255     #             yield index, token, value
256     # lexer.get_tokens_unprocessed = over.__get__(lexer)
257

```

4.3.2 create_style

```
self.create_style self.create_style()
```

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

```
258 def create_style(self):
```

```

259     args = self.arguments
260     if not args.create_style:
261         return
262     texopts = args.texopts
263     pyg_sty_p = texopts.pyg_sty_p
264     if args.cache and pyg_sty_p.exists():
265         return
266     texopts = self.texopts
267     style = self.pygopts.style
268     formatter = self.formatter
269     style_defs = formatter.get_style_defs() \
270         .replace(r'\makeatletter', '') \
271         .replace(r'\makeatother', '') \
272         .replace('\n', '%\n')
273     sty = self.texopts.sty_template.replace(
274         '<placeholder:style_name>',
275         style,
276     ).replace(
277         '<placeholder:style_defs>',
278         style_defs,
279     ).replace(
280         '{%}',
281         '{%}\n}{',
282     ).replace(
283         '[%]',
284         '[%]\n}%',
285     ).replace(
286         '{}%',
287         '{%[\n]}%',
288     )
289     with pyg_sty_p.open(mode='w', encoding='utf-8') as f:
290         f.write(sty)
291     if args.debug:
292         print('STYLE', os.path.relpath(pyg_sty_p))

```

4.3.3 pygmentize

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

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

```

293     def pygmentize(self, source):
294         source = highlight(source, self.lexer, self.formatter)
295         m = re.match(
296             r'\begin{CDR@Pyg@Verbatim}.*?\n(.*)\n\\end{CDR@Pyg@Verbatim}\s*\Z',
297             source,
298             flags=re.S
299         )
300         assert(m)
301         highlighted = m.group(1)
302         texopts = self.texopts
303         if texopts.is_inline:
304             return highlighted.replace(' ', r'\CDR@Sp '), 0
305         fv_opts = self.fv_opts

```

```

306     lines = highlighted.split('\n')
307     ans_code = []
308     try:
309         firstnumber = abs(int(fv_opts.firstnumber))
310     except ValueError:
311         firstnumber = 1
312     number = firstnumber
313     stepnumber = fv_opts.stepnumber
314     numbering = fv_opts.numbers != 'none'
315     def more(template, line):
316         nonlocal number
317         ans_code.append(template.replace(
318             '<placeholder:number>', f'{number}',
319             ).replace(
320                 '<placeholder:line>', line,
321             ))
322         number += 1
323     if len(lines) == 1:
324         more(texopts.single_line_template, lines.pop(0))
325     elif len(lines):
326         more(texopts.first_line_template, lines.pop(0))
327         more(texopts.second_line_template, lines.pop(0))
328         if stepnumber < 2:
329             def template():
330                 return texopts.black_line_template
331         elif stepnumber % 5 == 0:
332             def template():
333                 return texopts.black_line_template if number %\
334                     stepnumber == 0 else texopts.white_line_template
335         else:
336             def template():
337                 return texopts.black_line_template if (number - firstnumber) %\
338                     stepnumber == 0 else texopts.white_line_template
339
340         for line in lines:
341             more(template(), line)
342
343     highlighted = '\n'.join(ans_code)
344     return highlighted, number-firstnumber

```

4.3.4 create_pygmented

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

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

```

345     def create_pygmented(self):
346         args = self.arguments
347         base = args.base
348         if not base:
349             return False
350         source = args.source
351         if not source:
352             tex_p = Path(base).with_suffix('.tex')

```

```

353     with open(tex_p, 'r') as f:
354         source = f.read()
355     pyg_tex_p = Path(base).with_suffix('.pyg.tex')
356     highlighted, count = self.pygmentize(source)
357     with pyg_tex_p.open(mode='w',encoding='utf-8') as f:
358         if count:
359             f.write(rf'''\CDR@Total{{{count}}}'')
360         f.write(highlighted)
361     if args.debug:
362         print('HIGHLIGHTED', os.path.relpath(pyg_tex_p))

```

4.4 Main entry

```

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

```

File III

coder.sty implementation

```

1 %<*sty>
2 \makeatletter

```

1 Installation test

```

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

```

2 Messages

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

3 Constants

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

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

(End definition for \c_CDR_tag and \c_CDR_Tags. These variables are documented on page ??.)

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

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

(End definition for \c_CDR_tag_get and \c_CDR_slash. These variables are documented on page ??.)

4 Implementation details

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

5 Variables

5.1 Internal scratch variables

These local variables are used in a very limited scope.

`\l_CDR_bool` Local scratch variable.

```
25 \bool_new:N \l_CDR_bool
```

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

`\l_CDR_tl` Local scratch variable.

```
26 \tl_new:N \l_CDR_tl
```

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

`\l_CDR_str` Local scratch variable.

```
27 \str_new:N \l_CDR_str
```

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

`\l_CDR_seq` Local scratch variable.

28 `\seq_new:N \l_CDR_seq`

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

`\l_CDR_prop` Local scratch variable.

29 `\prop_new:N \l_CDR_prop`

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

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

30 `\clist_new:N \l_CDR_clist`

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

5.2 Files

`\l_CDR_in` Input file identifier

31 `\ior_new:N \l_CDR_in`

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

`\l_CDR_out` Output file identifier

32 `\iow_new:N \l_CDR_out`

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

5.3 Global variables

Line number counter for the source code chunks.

`\g_CDR_source_int` Chunk number counter.

33 `\int_new:N \g_CDR_source_int`

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

`\g_CDR_source_prop` Global source property list.

34 `\prop_new:N \g_CDR_source_prop`

(End definition for \g_CDR_source_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 ??.)

<code>\CDR_tag_get_path:cc</code>	★	<code>\CDR_tag_get_path:cc {<tag name>} {<relative key path>}</code>
<code>\CDR_tag_get_path:c</code>	★	<code>\CDR_tag_get_path:c {<relative key path>}</code>

Internal: return a unique key based on the arguments. Used to store and retrieve values. In the second version, the `<tag name>` is not provided and set to `__local`.

```
48 \cs_new:Npn \CDR_tag_get_path:cc #1 #2 {
49   \c_CDR_tag_get @ #1 / #2
50 }
51 \cs_new:Npn \CDR_tag_get_path:c {
52   \CDR_tag_get_path:cc { __local }
53 }
```

6.2 Set

<code>\CDR_tag_set:ccn</code>	★	<code>\CDR_tag_set:ccn {<tag name>} {<relative key path>} {<value>}</code>
<code>\CDR_tag_set:ccV</code>	★	

Store `<value>`, which is further retrieved with the instruction `\CDR_tag_get:cc {<tag name>} {<relative key path>}`. Only `<tag name>` and `<relative key path>` containing no `@` character are supported. Record the relative key path (the part after the tag name) of the current full key path in `g_CDR_tag_path_seq`. All the affectations are made at the current \TeX group level. *Nota Bene:* `\cs_generate_variant:Nn` is buggy when there is a ‘c’ argument.

```
54 \cs_new_protected:Npn \CDR_tag_set:ccn #1 #2 #3 {
55   \seq_put_left:Nx \g_CDR_tag_path_seq { #2 }
56   \cs_set:cpn { \CDR_tag_get_path:cc { #1 } { #2 } } { \exp_not:n { #3 } }
57 }
58 \cs_new_protected:Npn \CDR_tag_set:ccV #1 #2 #3 {
59   \exp_args:NnnV
60   \CDR_tag_set:ccn { #1 } { #2 } #3
61 }
```

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

```

62 \tl_set:Nn \l_CDR_tl { /([~/]*)/(.*)$ } \use_none:n { $ }
63 \tl_put_left:NV \l_CDR_tl \c_CDR_tag
64 \tl_put_left:Nn \l_CDR_tl { ^ }
65 \exp_args:NNV
66 \regex_const:Nn \c_CDR_tag_regex \l_CDR_tl

```

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

`\CDR_tag_set:n` `\CDR_tag_set:n {<value>}`

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.

```

67 \cs_new:Npn \CDR_tag_set:n {
68   \exp_args:NnV
69   \regex_extract_once:NnNTF \c_CDR_tag_regex
70     \l_keys_path_str \l_CDR_seq {
71     \CDR_tag_set:ccn
72     { \seq_item:Nn \l_CDR_seq 2 }
73     { \seq_item:Nn \l_CDR_seq 3 }
74   } {
75     \PackageWarning
76       { coder }
77       { Unexpected-key-path~'\l_keys_path_str' }
78     \use_none:n
79   }
80 }

```

`\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.

```

81 \cs_new:Npn \CDR_tag_set: {
82   \exp_args:NV
83   \CDR_tag_set:n \l_keys_value_tl
84 }

```

`\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.

```

85 \cs_new:Npn \CDR_tag_set:cn #1 {
86   \exp_args:NnV
87   \regex_extract_once:NnNTF \c_CDR_tag_regex
88     \l_keys_path_str \l_CDR_seq {

```

```

89   \CDR_tag_set:ccn
90     { \seq_item:Nn \l_CDR_seq 2 }
91     { #1 }
92   } {
93     \PackageWarning
94       { coder }
95       { Unexpected-key~path~'\l_keys_path_str' }
96     \use_none:n
97   }
98 }

```

\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.

```

99 \regex_const:Nn \c_CDR_root_regex { ^(.*)/.*$ } \use_none:n { $ }
100 \cs_new:Npn \CDR_tag_choices: {
101   \exp_args:NVV
102   \str_if_eq:nnT \l_keys_key_tl \l_keys_choice_tl {
103     \exp_args:NnV
104     \regex_extract_once:NnNT \c_CDR_root_regex
105       \l_keys_path_str \l_CDR_seq {
106       \str_set:Nx \l_keys_path_str {
107         \seq_item:Nn \l_CDR_seq 2
108       }
109     }
110   }
111 }

```

\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.

```

112 \cs_new:Npn \CDR_tag_choices_set: {
113   \CDR_tag_choices:
114   \exp_args:NV
115   \CDR_tag_set:n \l_keys_choice_tl
116 }

```

\CDR_if_tag_truthy:ccTF * \CDR_if_truthy:ccTF {<tag name>} {<relative key path>} {<true code>} {<false code>}

\CDR_if_tag_truthy:ccTF * \CDR_if_truthy:ccTF {<relative key path>} {<true code>} {<false code>}

Execute <true code> when the property for <tag name> and <relative key path> is a truthy value, <false code> otherwise. A truthy value is a text which is not “false” in a case insensitive comparison. In the second version, the <tag name> is not provided and set to __local.

```

117 \prg_new_conditional:Nnn \CDR_if_tag_truthy:cc { p, T, F, TF } {
118   \exp_args:Ne

```

```

119 \str_compare:nNnTF {
120   \exp_args:Ne \str_lowercase:n { \CDR_tag_get:cc { #1 } { #2 } }
121 } = { false } {
122   \prg_return_false:
123 } {
124   \prg_return_true:
125 }
126 }
127 \prg_new_conditional:Nnn \CDR_if_tag_truthy:c { p, T, F, TF } {
128   \exp_args:Ne
129   \str_compare:nNnTF {
130     \exp_args:Ne \str_lowercase:n { \CDR_tag_get:c { #1 } }
131   } = { false } {
132     \prg_return_false:
133   } {
134     \prg_return_true:
135   }
136 }

```

\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”.

```

137 \prg_new_conditional:Nnn \CDR_if_truthy:n { p, T, F, TF } {
138   \exp_args:Nf
139   \str_compare:nNnTF { \str_lowercase:n { #1 } } = { false } {
140     \prg_return_false:
141   } {
142     \prg_return_true:
143   }
144 }
145 \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.

```

146 \cs_new_protected:Npn \CDR_tag_boolean_set:n #1 {
147   \CDR_if_truthy:nTF { #1 } {
148     \CDR_tag_set:n { true }
149   } {
150     \CDR_tag_set:n { false }
151   }
152 }
153 \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/⟨namen⟩` 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.

```
154 \prg_new_conditional:Nnn \CDR_tag_if_exist_here:cc { T, F, TF } {
155   \cs_if_exist:cTF { \CDR_tag_get_path:cc { #1 } { #2 } } {
156     \prg_return_true:
157   } {
158     \prg_return_false:
159   }
160 }
```

```
\CDR_tag_if_exist:ccTF * \CDR_tag_if_exist:ccTF {⟨tag name⟩} ⟨relative key path⟩ {⟨true code⟩} {⟨false code⟩}
\CDR_tag_if_exist:cTF * \CDR_tag_if_exist:cTF ⟨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. In the second version, the `⟨tag name⟩` is not provided and set to `__local`.

```

161 \prg_new_conditional:Nnn \CDR_tag_if_exist:cc { T, F, TF } {
162   \cs_if_exist:cTF { \CDR_tag_get_path:cc { #1 } { #2 } } {
163     \prg_return_true:
164   } {
165     \seq_if_exist:cTF { \CDR_tag_parent_seq:c { #1 } } {
166       \seq_map_tokens:cn
167         { \CDR_tag_parent_seq:c { #1 } }
168         { \CDR_tag_if_exist_f:cn { #2 } }
169     } {
170       \prg_return_false:
171     }
172   }
173 }
174 \prg_new_conditional:Nnn \CDR_tag_if_exist:c { T, F, TF } {
175   \cs_if_exist:cTF { \CDR_tag_get_path:c { #1 } } {
176     \prg_return_true:
177   } {
178     \seq_if_exist:cTF { \CDR_tag_parent_seq:c { __local } } {
179       \seq_map_tokens:cn
180         { \CDR_tag_parent_seq:c { __local } }
181         { \CDR_tag_if_exist_f:cn { #1 } }
182     } {
183       \prg_return_false:
184     }
185   }
186 }
187 \cs_new:Npn \CDR_tag_if_exist_f:cn #1 #2 {
188   \quark_if_no_value:nTF { #2 } {
189     \seq_map_break:n {
190       \prg_return_false:
191     }
192   } {
193     \CDR_tag_if_exist:ccT { #2 } { #1 } {
194       \seq_map_break:n {
195         \prg_return_true:
196       }
197     }
198   }
199 }

```

\CDR_tag_get:cc *	\CDR_tag_get:cc {<tag name>} {<relative key path>}
\CDR_tag_get:c *	\CDR_tag_get:c {<relative key path>}

The property value stored for <tag name> and <relative key path>. Takes care of inheritance. In the second version, the <tag name> is not provided an set to __local.

```

200 \cs_new:Npn \CDR_tag_get:cc #1 #2 {
201   \CDR_tag_if_exist_here:ccTF { #1 } { #2 } {
202     \use:c { \CDR_tag_get_path:cc { #1 } { #2 } }
203   } {
204     \seq_if_exist:cT { \CDR_tag_parent_seq:c { #1 } } {
205       \seq_map_tokens:cn
206         { \CDR_tag_parent_seq:c { #1 } }
207         { \CDR_tag_get_f:cn { #2 } }

```

```

208     }
209   }
210 }
211 \cs_new:Npn \CDR_tag_get_f:cn #1 #2 {
212   \quark_if_no_value:nF { #2 } {
213     \CDR_tag_if_exist_here:ccT { #2 } { #1 } {
214       \seq_map_break:n {
215         \use:c { \CDR_tag_get_path:cc { #2 } { #1 } }
216       }
217     }
218   }
219 }
220 \cs_new:Npn \CDR_tag_get:c {
221   \CDR_tag_get:cc { __local }
222 }

```

\CDR_tag_get:ccN	\CDR_tag_get:ccN {<tag name>} {<relative key path>} {<tl variable>}
\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>. In the second version, the <tag name> is not provided an set to __local.

```

223 \cs_new_protected:Npn \CDR_tag_get:ccN #1 #2 #3 {
224   \tl_set:Nf #3 { \CDR_tag_get:cc { #1 } { #2 } }
225 }
226 \cs_new_protected:Npn \CDR_tag_get:cN {
227   \CDR_tag_get:ccN { __local }
228 }

```

\CDR_tag_get:ccNTF	\CDR_tag_get:ccNTF {<tag name>} {<relative key path>} <tl var> {<true code>}
\CDR_tag_get:cNTF	{<false code>}
	\CDR_tag_get:cNTF {<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>. In the second version, the <tag name> is not provided an set to __local.

```

229 \prg_new_protected_conditional:Nnn \CDR_tag_get:ccN { T, F, TF } {
230   \CDR_tag_if_exist:ccTF { #1 } { #2 } {
231     \CDR_tag_get:ccN { #1 } { #2 } #3
232     \prg_return_true:
233   } {
234     \prg_return_false:
235   }
236 }
237 \prg_new_protected_conditional:Nnn \CDR_tag_get:cN { T, F, TF } {
238   \CDR_tag_if_exist:cTF { #1 } {
239     \CDR_tag_get:cN { #1 } #2
240     \prg_return_true:
241   } {
242     \prg_return_false:
243   }
244 }

```

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.

```

245 \cs_new:Npn \CDR_tag_parent_seq:c #1 {
246   g_CDR:parent.tag @ #1 _seq
247 }
```

\CDR_tag_inherit:cn	\CDR_tag_inherit:cn {⟨child name⟩} {⟨parent names comma list⟩}
---------------------	--

\CDR_tag_inherit:(cf cV)	Set the parents of ⟨child name⟩ to the given list.
--------------------------	--

```

248 \cs_new:Npn \CDR_tag_inherit:cn #1 #2 {
249   \seq_set_from_clist:cn { \CDR_tag_parent_seq:c { #1 } } { #2 }
250   \seq_remove_duplicates:c \l_CDR_tl
251   \seq_remove_all:cn \l_CDR_tl {}
252   \seq_put_right:cn \l_CDR_tl { \q_no_value }
253 }
254 \cs_new:Npn \CDR_tag_inherit:cf {
255   \exp_args:Nnf \CDR_tag_inherit:cn
256 }
257 \cs_new:Npn \CDR_tag_inherit:cV {
258   \exp_args:NnV \CDR_tag_inherit:cn
259 }
```

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.

```

260 \AddToHook { begindocument/before } {
261   \IfFileExists {./\jobname.aux} {} {
262     \lua_now:n {CDR:cache_clean_all()}
263   }
264 }
```

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

```

265 \AddToHook { enddocument/end } {
266   \lua_now:n {CDR:cache_clean_unused()}
267 }
```


8 Utilities

\CDR_clist_map_inline:Nnn \CDR_clist_map_inline:Nnn *<clist var>* *{<empty code>}* *{<non empty code>}*

Execute *<empty code>* when the list is empty, otherwise call `\clist_map_inline:Nn` with *<non empty code>*.

```

268 \cs_new:Npn \CDR_clist_map_inline:Nnn #1 #2 {
269   \clist_if_empty:NTF #1 {
270     #2
271     \use_none:n
272   } {
273     \clist_map_inline:Nn #1
274   }
275 }
```

\CDR_if_block_p: * \CDR_if_block:TF *{<true code>}* *{<false code>}*

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

```

276 \prg_new_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
277   \PackageError
278     { coder }
279     { Conditional~not~available }
280 }
```

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

```

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

9 l3keys 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`.

```

282 \cs_generate_variant:Nn \keys_define:nn { Vn, xn }
283 \cs_new:Npn \CDR_tag_keys_define:nn #1 {
284   \keys_define:xn { \c_CDR_tag / \exp_not:n { #1 } }
285 }
286 \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.

```
287 \cs_new:Npn \CDR_tag_keys_set:nn #1 {
288   \exp_args:Nx
289   \keys_set:nn { \c_CDR_tag / \exp_not:n { #1 } }
290 }
291 \cs_generate_variant:Nn \CDR_tag_keys_set:nn { nV }
```

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>.

```
292 \cs_new:Npn \CDR_keys_set_known:nnN #1 #2 {
293   \keys_set_known:nnnN { #1 } { #2 } { #1 }
294 }
295 \cs_generate_variant:Nn \CDR_keys_set_known:nnN { x, VV }
```

```
\CDR_keys_inherit:nnn \CDR_keys_inherit:nnn {<tag root>} {<tag name>} {<parents comma list>}
```

The <tag name> and parents are given relative to <tag root>. Set the inheritance.

```
296 \cs_new:Npn \CDR_keys_inherit__:nnn #1 #2 #3 {
297   \keys_define:nn { #1 } { #2 .inherit:n = { #3 } }
298 }
299 \cs_new:Npn \CDR_keys_inherit:nnn #1 #2 #3 {
300   \tl_if_empty:nTF { #1 } {
301     \CDR_keys_inherit__:nnn { } { #2 } { #3 }
302   } {
303     \clist_set:Nn \l_CDR_clist { #3 }
304     \exp_args:Nnnx
305     \CDR_keys_inherit__:nnn { #1 } { #2 } {
306       #1 / \clist_use:Nn \l_CDR_clist { ,#1/ }
307     }
308   }
309 }
310 \cs_generate_variant:Nn \CDR_keys_inherit:nnn { VnV, Vnn }
```

```
\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.

```

311 \cs_generate_variant:Nn \keys_set_known:nnnN { VVV, nVx }
312 \cs_new:Npn \CDR_tag_keys_set_known:nnN #1 {
313   \CDR_keys_set_known:xnN { \c_CDR_tag / \exp_not:n { #1 } }
314 }
315 \cs_generate_variant:Nn \CDR_tag_keys_set_known:nnN { nV }

```

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

```

316 \tl_set:Nn \l_CDR_tl { /([~/*])(?:/(.))*? $ } \use_none:n { $ }
317 \tl_put_left:NV \l_CDR_tl \c_CDR_tag
318 \tl_put_left:Nn \l_CDR_tl { ^ }
319 \exp_args:NNV
320 \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 `'/`.

```

321 \regex_const:Nn \c_CDR_engine_regex { ^[~/*]*\sengine\soptions$ } \use_none:n { $ }
322 \cs_new:Npn \CDR_tag_provide_from_clist:n #1 {
323   \exp_args:NNx
324   \regex_extract_once:NnNTF \c_CDR_provide_regex {
325     \c_CDR_Tags / #1
326   } \l_CDR_seq {
327     \tl_set:Nx \l_CDR_tl { \seq_item:Nn \l_CDR_seq 3 }
328     \exp_args:Nx
329     \clist_map_inline:nn {
330       \seq_item:Nn \l_CDR_seq 2
331     } {
332       \exp_args:NV
333       \keys_if_exist:nnF \c_CDR_tag { ##1 } {
334         \CDR_keys_inherit:Vnn \c_CDR_tag { ##1 } {
335           __pygments, __pygments.block,
336           default.block, default.code, default,
337           __fancyvrb, __fancyvrb.block, __fancyvrb.all
338         }
339         \keys_define:Vn \c_CDR_tag {
340           ##1 .code:n = \CDR_tag_keys_set:nn { ##1 } { #####1 },
341           ##1 .value_required:n = true,
342         }
343       }
344       \exp_args:NxV
345       \keys_if_exist:nnF { \c_CDR_tag / ##1 } \l_CDR_tl {
346         \exp_args:NNV
347         \regex_match:NnT \c_CDR_engine_regex
348         \l_CDR_tl {
349           \CDR_tag_keys_define:nx { ##1 } {
350             \l_CDR_tl .code:n = \exp_not:n { \CDR_tag_set:n { #####1 } },

```

```

351         \l_CDR_tl .value_required:n = true,
352     }
353 }
354 }
355 }
356 } {
357     \regex_match:NnT \c_CDR_engine_regex { #1 } {
358         \CDR_tag_keys_define:nn { default } {
359             #1 .code:n = \CDR_tag_set:n { ##1 },
360             #1 .value_required:n = true,
361         }
362     }
363 }
364 }
365 \cs_new:Npn \CDR_tag_provide_from_clist:nn #1 #2 {
366     \CDR_tag_provide_from_clist:n { #1 }
367 }
368 \cs_new:Npn \CDR_tag_provide_from_keyval:n {
369     \keyval_parse:nnn {
370         \CDR_tag_provide_from_clist:n
371     } {
372         \CDR_tag_provide_from_clist:nn
373     }
374 }
375 \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.
---	--

```

376 \sys_get_shell:nnN {which-pygmentize} {} \l_CDR_tl
377 \prg_new_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } { }
378 \tl_if_in:NnTF \l_CDR_tl { pygmentize } {
379     \prg_set_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } {
380         \prg_return_true:
381     }
382 } {
383     \prg_set_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } {
384         \prg_return_false:
385     }
386 }

```

9.2.2 `__pygments` `l3keys` module

```

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

388   lang .code:n = \CDR_tag_set:,
389   lang .value_required:n = true,

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

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

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

391   style .code:n = \CDR_tag_set:,
392   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.

393   commandprefix .code:n = \CDR_tag_set:,
394   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.

395   mathescape .code:n = \CDR_tag_boolean_set:x { #1 },
396   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.

397   escapeinside .code:n = \CDR_tag_set:,
398   escapeinside .value_required:n = true,

  ● __initialize Initializer.

399   __initialize .meta:n = {
400     lang = tex,
401     pygments = \CDR_has_pygments:TF { true } { false },
402     style=default,
403     commandprefix=PY,
404     mathescape=false,
405     escapeinside=,
406   },
407   __initialize .value_forbidden:n = true,

408 }
409 \AtBeginDocument{
410   \CDR_tag_keys_set:nn { __pygments } { __initialize }
411 }

```

9.2.3 \c_CDR_tag / __pygments.block l3keys module

```
412 \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**.

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

- **__initialize** Initializer.

```
415 __initialize .meta:n = {
416 texcomments=false,
417 },
418 __initialize .value_forbidden:n = true,

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

9.3 Specific to coder

9.3.1 default l3keys module

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

Keys are:

- **format**=*(format commands)* the format used to display the code (mainly font, size and color), after the font has been selected. Initially empty.

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

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

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

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

```
427 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.

```
428 post-processor .code:n = \CDR_tag_set:,
429 post-processor .value_required:n = true,
```

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

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

● **engine=⟨engine name⟩** to specify the engine used to display inline code or blocks. Initially default.

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

● **default engine options=⟨default engine options⟩** to specify the corresponding options,

```
434   default~engine~options .code:n = \CDR_tag_set:,
435   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.

```
436   __initialize .meta:n = {
437     format = ,
438     cache = true,
439     debug = false,
440     post-processor = ,
441     parskip = \the\parskip,
442     engine = default,
443     default~engine~options = ,
444   },
445   __initialize .value_forbidden:n = true,
446 }
447 \AtBeginDocument{
448   \CDR_tag_keys_set:nn { default } { __initialize }
449 }
```

9.3.2 default.code l3keys module

Void for the moment.

```
450 \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.

```
451   __initialize .meta:n = {
452   },
453   __initialize .value_forbidden:n = true,
454 }
455 \AtBeginDocument{
456   \CDR_tag_keys_set:nn { default.code } { __initialize }
457 }
```

9.3.3 default.block l3keys module

```
458 \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.

```
459 tags .code:n = {  
460   \clist_set:Nn \l_CDR_tags_clist { #1 }  
461   \clist_remove_duplicates:N \l_CDR_tags_clist  
462   \exp_args:NV  
463   \CDR_tag_set:n \l_CDR_tags_clist  
464 },  
465 tags .value_required:n = true,
```

● **tags format** = <format commands> , where <format> is used the format used to display the tag names (mainly font, size and color), after it is appended to the numbers format. Initially empty.

```
466 tags~format .code:n = \CDR_tag_set:,  
467 tags~format .value_required:n = true,
```

● **numbers format** = <format commands> , where <format> is used the format used to display line numbers (mainly font, size and color).

```
468 numbers~format .code:n = \CDR_tag_set:,  
469 numbers~format .value_required:n = true,
```

● **show tags** [=true|false] whether tags should be displayed.

```
470 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.

```
471 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,

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

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

```
473 blockskip .code:n = \CDR_tag_set:,  
474 blockskip .value_required:n = true,
```

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


```

475 __initialize .meta:n = {
476   tags = ,
477   show-tags = true,
478   only-top = true,
479   use-margin = true,
480   numbers-format = {
481     \sffamily
482     \scriptsize
483     \color{gray}
484   },
485   tags-format = {
486     \bfseries
487   },
488   blockskip = \topsep,
489 },
490 __initialize .value_forbidden:n = true,
491 }
492 \AtBeginDocument{
493   \CDR_tag_keys_set:nn { default.block } { __initialize }
494 }

```

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 __fancyvrb l3keys module

```

495 \CDR_tag_keys_define:nn { __fancyvrb } {


```

 **formatcom**=*<command>* execute before printing verbatim text. Initially empty.

```

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


```

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

```

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

```

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

```

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

```

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

```

🔴 **fontseries=⟨series name⟩** L^AT_EX font series to use. Initially auto: the same as the current font.

```

504 fontseries .code:n = \CDR_tag_set:,
505 fontseries .value_required:n = true,

```

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

```

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

```

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

```

507 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.

```

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

```

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

```

509 tabsize .code:n = \CDR_tag_set:,
510 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.

```

511 defineactive .code:n = \CDR_tag_set:,
512 defineactive .value_required:n = true,

```

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

```

513 relabel .code:n = \CDR_tag_set:,
514 relabel .value_required:n = true,

```

✅ **__initialize** Initialization.

```

515 __initialize .meta:n = {
516   formatcom = ,
517   fontfamily = tt,
518   fontsize = auto,
519   fontseries = auto,
520   fontshape = auto,
521   showspaces = false,
522   showtabs = false,
523   obeytabs = false,
524   tabsize = 2,
525   defineactive = ,
526   relabel = ,
527 },
528 __initialize .value_forbidden:n = true,

```

```

529 }
530 \AtBeginDocument{
531   \CDR_tag_keys_set:nn { __fancyvrb } { __initialize }
532 }

```

9.4.2 `__fancyvrb.block l3keys` module

Block specific options, except numbering.

```

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

```

- **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.

```

535   frame .choices:nn =
536     { none, leftline, topline, bottomline, lines, single }
537     { \CDR_tag_choices_set: },

```

- **framerule**=`<dimension>` width of the rule of the frame if any. Initially 0.4pt.

```

538   framerule .code:n = \CDR_tag_set:,
539   framerule .value_required:n = true,

```

- **framesep**=`<dimension>` width of the gap between the frame (if any) and the text. Initially `\fboxsep`.

```

540   framesep .code:n = \CDR_tag_set:,
541   framesep .value_required:n = true,

```

- **rulecolor**=`<color command>` color of the frame rule, expressed in the standard `LATEX` way. Initially black.

```

542   rulecolor .code:n = \CDR_tag_set:,
543   rulecolor .value_required:n = true,

```

- **rulecolor**=`<color command>` color used to fill the space between the frame and the text (its thickness is given by `framesep`). Initially empty.

```

544   fillcolor .code:n = \CDR_tag_set:,
545   fillcolor .value_required:n = true,

```

- **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.

```

546 label .code:n = \CDR_tag_set:,
547 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.

```

548 labelposition .choices:nn =
549 { none, topline, bottomline, all }
550 { \CDR_tag_choices_set: },

```

- **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.

```

551 baselinestretch .code:n = \CDR_tag_set:,
552 baselinestretch .value_required:n = true,

```

- ⊗ **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 **Opt**: no left margin.

```

553 xleftmargin .code:n = \CDR_tag_set:,
554 xleftmargin .value_required:n = true,

```

- **xrightmargin= $\langle dimension \rangle$** right margin to add after each line. Initially **Opt**: no right margin.

```

555 xrightmargin .code:n = \CDR_tag_set:,
556 xrightmargin .value_required:n = true,

```

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

```

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

```

- **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**.

```

558 hfuzz .code:n = \CDR_tag_set:,
559 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**.

```

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

```

✓ **__initialize** Initialization.

```
561 __initialize .meta:n = {
562     frame = none,
563     label = ,
564     labelposition = none,% auto?
565     baselinestretch = auto,
566     resetmargins = true,
567     xleftmargin = 0pt,
568     xrightmargin = 0pt,
569     hfuzz = 2pt,
570     samepage = false,
571 },
572 __initialize .value_forbidden:n = true,

573 }
574 \AtBeginDocument{
575     \CDR_tag_keys_set:nn { __fancyvrb.block } { __initialize }
576 }
```

9.4.3 **__fancyvrb.number l3keys** module

Block line numbering.

```
577 \CDR_tag_keys_define:nn { __fancyvrb.number } {
```

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

```
578 commentchar .code:n = \CDR_tag_set:,
579 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.

```
580 gobble .choices:nn = {
581     0,1,2,3,4,5,6,7,8,9
582 } {
583     \CDR_tag_choices_set:
584 },
```

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

```
585 numbers .choices:nn =
586     { none, left, right }
587     { \CDR_tag_choices_set: },
```

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

```
588 numbersep .code:n = \CDR_tag_set:,
589 numbersep .value_required:n = true,
```

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

```
590 firstnumber .code:n = {
591   \regex_match:NnTF \c_CDR_integer_regex { #1 } {
592     \CDR_tag_set:
593   } {
594     \str_case:nnF { #1 } {
595       { auto } { \CDR_tag_set: }
596       { last } { \CDR_tag_set: }
597     } {
598       \PackageWarning
599         { CDR }
600         { Value~‘#1’~not~in~auto,~last. }
601     }
602   }
603 },
604 firstnumber .value_required:n = true,
```

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

```
605 stepnumber .code:n = \CDR_tag_set:,
606 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.

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

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

```
608 firstline .code:n = \CDR_tag_set:,
609 firstline .value_required:n = true,
```

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

```
610 lastline .code:n = \CDR_tag_set:,
611 lastline .value_required:n = true,
```

- ✅ **__initialize** Initialization.

```
612 __initialize .meta:n = {
613   commentchar = ,
614   gobble = 0,
615   numbers = left,
616   numbersep = 1ex,
617   firstnumber = auto,
618   stepnumber = 1,
```

```

619     numberblanklines = true,
620     firstline = ,
621     lastline = ,
622 },
623 __initialize .value_forbidden:n = true,

624 }
625 \AtBeginDocument{
626   \CDR_tag_keys_set:nn { __fancyvrb.number } { __initialize }
627 }

```


9.4.4 __fancyvrb.all l3keys module

Options available when pygments is not used.

```

628 \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.

```

629   commandchars .code:n = \CDR_tag_set:,
630   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.

```

631   codes .code:n = \CDR_tag_set:,
632   codes .value_required:n = true,

```

-  **__initialize** Initialization.

```

633   __initialize .meta:n = {
634     commandchars = ,
635     codes = ,
636   },
637   __initialize .value_forbidden:n = true,

638 }
639 \AtBeginDocument{
640   \CDR_tag_keys_set:nn { __fancyvrb.all } { __initialize }
641 }

```

10 \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

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

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

```
643   only~description .choices:nn = { false, true, {} } {
644     \int_compare:nNnTF \l_keys_choice_int = 1 {
645       \prg_set_conditional:Nnn \CDR_if_only_description: { p, T, F, TF } { \prg_return_true: }
646     } {
647       \prg_set_conditional:Nnn \CDR_if_only_description: { p, T, F, TF } { \prg_return_false: }
648     }
649   },
650   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.

```
651   python-path .code:n = {
652     \str_set:Nn \l_CDR_str { #1 }
653     \lua_now:n { CDR:set_python_path('l_CDR_str') }
654   },
655 }
```

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.

```
656 \exp_args_generate:n { xV, nnV }
657 \cs_new:Npn \CDR_check_unknown:N #1 {
658   \tl_if_empty:NF #1 {
659     \cs_set:Npn \CDR_check_unknown:n ##1 {
660       \PackageWarning
661         { coder }
662         { Unknow~key~'##1' }
663     }
664     \cs_set:Npn \CDR_check_unknown:nn ##1 ##2 {
665       \CDR_check_unknown:n { ##1 }
```



```

666     }
667     \exp_args:NnnV
668     \keyval_parse:nnn {
669         \CDR_check_unknown:n
670     } {
671         \CDR_check_unknown:nn
672     } #1
673 }
674 }

675 \NewDocumentCommand \CDRSet { m } {
676     \CDR_keys_set_known:nnN { CDR@Set } { #1 } \l_CDR_keyval_tl
677     \clist_map_inline:nn {
678         __pygments, __pygments.block,
679         default.block, default.code, default,
680         __fancyvrb, __fancyvrb.block, __fancyvrb.all
681     } {
682         \CDR_tag_keys_set_known:nVN { ##1 } \l_CDR_keyval_tl \l_CDR_keyval_tl
683     }
684     \CDR_keys_set_known:VVN \c_CDR_Tags \l_CDR_keyval_tl \l_CDR_keyval_tl
685     \CDR_tag_provide_from_keyval:V \l_CDR_keyval_tl
686     \CDR_keys_set_known:VVN \c_CDR_Tags \l_CDR_keyval_tl \l_CDR_keyval_tl
687     \CDR_tag_keys_set:nV { default } \l_CDR_keyval_tl
688 }

```

11 \CDRExport

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

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

11.1 Storage

\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.

```

689 \cs_new:Npn \CDR_export_get_path:cc #1 #2 {
690     CDR @ export @ get @ #1 / #2
691 }

```

\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

```

692 \cs_new_protected:Npn \CDR_export_set:ccn #1 #2 #3 {
693     \cs_set:cpn { \CDR_export_get_path:cc { #1 } { #2 } } { \exp_not:n { #3 } }
694 }
695 \cs_new_protected:Npn \CDR_export_set:Vcn #1 {

```

```

696 \exp_args:NV
697 \CDR_export_set:ccn { #1 }
698 }
699 \cs_new_protected:Npn \CDR_export_set:VcV #1 #2 #3 {
700 \exp_args:NvNv
701 \CDR_export_set:ccn #1 { #2 } #3
702 }

```

\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.

```

703 \prg_new_conditional:Nnn \CDR_export_if_exist:cc { p, T, F, TF } {
704 \cs_if_exist:cTF { \CDR_export_get_path:cc { #1 } { #2 } } {
705 \prg_return_true:
706 } {
707 \prg_return_false:
708 }
709 }

```

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

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

```

710 \cs_new:Npn \CDR_export_get:cc #1 #2 {
711 \CDR_export_if_exist:ccT { #1 } { #2 } {
712 \use:c { \CDR_export_get_path:cc { #1 } { #2 } }
713 }
714 }

```

\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.

```

715 \prg_new_protected_conditional:Nnn \CDR_export_get:ccN { T, F, TF } {
716 \CDR_export_if_exist:ccTF { #1 } { #2 } {
717 \tl_set:Nx #3 { \CDR_export_get:cc { #1 } { #2 } }
718 \prg_return_true:
719 } {
720 \prg_return_false:
721 }
722 }

```

11.2 Storage

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

```

723 \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.

```
724 \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.
`\g_CDR_tags_clist`

```
725 \clist_new:N \l_CDR_tags_clist
```

```
726 \clist_new:N \g_CDR_tags_clist
```

(End definition for `\l_CDR_tags_clist` and `\g_CDR_tags_clist`. These variables are 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.

```
727 \prop_new:N \l_CDR_export_prop
```

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

11.3 CDR@Export l3keys module

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

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

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

```
729   file .tl_set:N = \l_CDR_file_tl,
```

```
730   file .value_required:n = true,
```

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

```
731   tags .code:n = {
```

```
732     \clist_set:Nn \l_CDR_tags_clist { #1 }
```

```
733     \clist_remove_duplicates:N \l_CDR_tags_clist
```

```
734     \prop_put:NV \l_CDR_prop \l_keys_key_str \l_CDR_tags_clist
```

```
735   },
```

```
736   tags .value_required:n = true,
```

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

```
737   lang .code:n = {
```

```
738     \prop_put:NVn \l_CDR_prop \l_keys_key_str { #1 }
```

```
739   },
```

```
740   lang .value_required:n = true,
```

● **preamble** the added preamble. Initially empty.

```

741 preamble .code:n = {
742   \prop_put:NVn \l_CDR_prop \l_keys_key_str { #1 }
743 },
744 preamble .value_required:n = true,

```

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

```

745 postamble .code:n = {
746   \prop_put:NVn \l_CDR_prop \l_keys_key_str { #1 }
747 },
748 postamble .value_required:n = true,

```

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

```

749 raw .choices:nn = { false, true, {} } {
750   \prop_put:NVx \l_CDR_prop \l_keys_key_str {
751     \int_compare:nNnTF
752       \l_keys_choice_int = 1 { false } { true }
753   }
754 },

```

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

```

755 __initialize .meta:n = {
756   __initialize_prop = #1,
757   file=,
758   tags=,
759   lang=tex,
760   preamble=,
761   postamble=,
762   raw=false,
763 },
764 __initialize .default:n = \l_CDR_export_prop,

```

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

```

765 __initialize_prop .code:n = \prop_clear:N #1,
766 __initialize_prop .value_required:n = true,
767 }

```

11.4 Implementation

```

768 \NewDocumentCommand \CDRExport { m } {
769   \keys_set:nn { CDR@Export } { __initialize }
770   \keys_set:nn { CDR@Export } { #1 }
771   \tl_if_empty:NTF \l_CDR_file_tl {
772     \PackageWarning
773       { coder }
774       { Missing~key~‘file’ }
775   } {
776     \CDR_export_set:VcV \l_CDR_file_tl { file } \l_CDR_file_tl
777     \prop_map_inline:Nn \l_CDR_prop {
778       \CDR_export_set:Vcn \l_CDR_file_tl { ##1 } { ##2 }
779     }

```

The list of tags must not be empty, raise an error otherwise. Records the list in `\g_CDR_tags_clist`, it will be the default list of forthcoming code blocks.

```

780 \tl_if_empty:NTF \l_CDR_tags_clist {
781   \PackageWarning
782     { coder }
783     { Missing-key~‘tags’ }
784 } {
785   \clist_set_eq:NN \g_CDR_tags_clist \l_CDR_tags_clist
786   \CDR_export_set:VcV \l_CDR_file_tl { file } \l_CDR_file_tl

```

If a `lang` is given, forwards the declaration to all the code chunks tagged within `\l_CDR_tags_clist`.

```

787   \exp_args:NV
788   \CDR_export_get:ccNT \l_CDR_file_tl { lang } \l_CDR_tl {
789     \clist_map_inline:Nn \l_CDR_tags_clist {
790       \CDR_tag_set:ccV { ##1 } { lang } \l_CDR_tl
791     }
792   }
793 }
794 }
795 }

```

Files are created at the end of the typesetting process.

```

796 \AddToHook { enddocument / end } {
797   \prop_map_inline:Nn \g_CDR_export_prop {
798     \tl_set:Nn \l_CDR_prop { #2 }
799     \str_set:Nx \l_CDR_str {
800       \prop_item:Nn \l_CDR_prop { file }
801     }
802     \lua_now:n { CDR:export_file('l_CDR_str') }
803     \clist_map_inline:nn {
804       tags, raw, preamble, postamble
805     } {
806       \str_set:Nx \l_CDR_str {
807         \prop_item:Nn \l_CDR_prop { ##1 }
808       }
809       \lua_now:n {
810         CDR:export_file_info('##1', 'l_CDR_str')
811       }
812     }
813     \lua_now:n { CDR:export_file_complete() }
814   }
815 }

```

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.

```

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

```

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

```

816 \cs_set:Npn \CDR@StyleDefine #1 {
817   \tl_gset:cn { g_CDR@Style/#1 }
818 }

```

\CDR@StyleUse	\CDR@StyleUse {<pygments style name>}
CDR@StyleUseTag	\CDR@StyleUseTag

Use the definitions for the given <pygments style name>. No safe check is made. The \CDR@StyleUseTag version finds the <pygments style name> from the context.

```

819 \cs_set:Npn \CDR@StyleUse #1 {
820   \tl_use:c { g_CDR@Style/#1 }
821 }
822 \cs_set:Npn \CDR@StyleUseTag {
823   \CDR@StyleUse { \CDR_tag_get:c { style } }
824 }

```

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

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

```

825 \prg_new_conditional:Nnn \CDR@StyleIfExist:c { TF } {
826   \tl_if_exist:cTF { g_CDR@Style/#1 } {
827     \prg_return_true:
828   } {
829     \prg_return_false:
830   }
831 }
832 \cs_set_eq:NN \CDR@StyleIfExist \CDR@StyleIfExist:cTF

```

13 Creating display engines

13.1 Utilities

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

```

833 \cs_new:Npn \CDR_code_engine:c #1 {
834   CDR@colored/code/#1:nn
835 }
836 \cs_new:Npn \CDR_block_engine:c #1 {
837   CDR@colored/block/#1
838 }
839 \cs_new:Npn \CDR_code_engine:V {
840   \exp_args:NV \CDR_code_engine:c
841 }
842 \cs_new:Npn \CDR_block_engine:V {
843   \exp_args:NV \CDR_block_engine:c
844 }

```

`\l_CDR_engine_tl` Storage for an engine name.

845 `\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

`\CDRCodeEngineNew` `\CDRCodeEngineNew {<engine name>}{<engine body>}`
`\CDRCodeEngineRenew` `\CDRCodeEngineRenew{<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.

```

846 \NewDocumentCommand \CDRCodeEngineNew { mm } {
847   \exp_args:Nx
848   \tl_if_empty:nTF { #1 } {
849     \PackageWarning
850       { coder }
851     { The~engine~cannot~be~void. }
852   } {
853     \cs_new:cpn { \CDR_code_engine:c {#1} } ##1 ##2 {
854       \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
855       #2
856     }
857     \ignorespaces
858   }
859 }

860 \NewDocumentCommand \CDRCodeEngineRenew { mm } {
861   \exp_args:Nx
862   \tl_if_empty:nTF { #1 } {
863     \PackageWarning
864       { coder }
865     { The~engine~cannot~be~void. }
866     \use_none:n
867   } {
868     \cs_if_exist:cTF { \CDR_code_engine:c { #1 } } {
869       \cs_set:cpn { \CDR_code_engine:c { #1 } } ##1 ##2 {
870         \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
871         #2
872       }
873     } {
874       \PackageWarning
875         { coder }
876       { No~code~engine~#1.}

```

```

877     }
878     \ignorespaces
879   }
880 }

```

\CDR@CodeEngineApply \CDR@CodeEngineApply {<source>}

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

```

881 \cs_new:Npn \CDR@CodeEngineApply #1 {
882   \CDR_tag_get:cN { engine } \l_CDR_engine_tl
883   \CDR_if_code_engine:VF \l_CDR_engine_tl {
884     \PackageError
885       { coder }
886       { \l_CDR_engine_tl\space code~engine~unknown,~replaced-by~'default' }
887       { See~\CDRCodeEngineNew~in~the~coder~manual }
888     \tl_set:Nn \l_CDR_engine_tl { default }
889   }
890   \CDR_tag_get:cN { engine~options } \l_CDR_options_tl
891   \tl_if_empty:NTF \l_CDR_options_tl {
892     \CDR_tag_get:cN { \l_CDR_engine_tl\space engine~options } \l_CDR_options_tl
893   } {
894     \tl_put_left:Nx \l_CDR_options_tl {
895       \CDR_tag_get:c { \l_CDR_engine_tl\space engine~options } ,
896     }
897   }
898   \exp_args:NnV
899   \use:c { \CDR_code_engine:V \l_CDR_engine_tl } \l_CDR_options_tl {
900     \CDR_tag_get:c { format }
901     #1
902   }
903 }

```

\CDRBlockEngineNew \CDRBlockEngineNew {<engine name>} {<begin instructions>} {<end instructions>}

\CDRBlockEngineRenew \CDRBlockEngineRenew {<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.

```

904 \NewDocumentCommand \CDRBlockEngineNew { mm } {
905   \NewDocumentEnvironment { \CDR_block_engine:c { #1 } } { m } {
906     \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
907     #2
908   }
909 }

```



```

910 \NewDocumentCommand \CDRBlockEngineRenew { mm } {
911   \tl_if_empty:nTF { #1 } {
912     \PackageWarning
913       { coder }
914       { The~engine~cannot~be~void. }
915     \use_none:n
916   } {
917     \RenewDocumentEnvironment { \CDR_block_engine:c { #1 } } { m } {
918       \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
919       #2
920     }
921   }
922 }

```

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>.

```

923 \prg_new_conditional:Nnn \CDR_if_code_engine:c { p, T, F, TF } {
924   \cs_if_exist:cTF { \CDR_code_engine:c { #1 } } {
925     \prg_return_true:
926   } {
927     \prg_return_false:
928   }
929 }
930 \prg_new_conditional:Nnn \CDR_if_code_engine:V { p, T, F, TF } {
931   \cs_if_exist:cTF { \CDR_code_engine:V #1 } {
932     \prg_return_true:
933   } {
934     \prg_return_false:
935   }
936 }

```

\CDR_if_block_engine:cTF ★ \CDR_if_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>.

```

937 \prg_new_conditional:Nnn \CDR_if_block_engine:c { p, T, F, TF } {
938   \cs_if_exist:cTF { \CDR_block_engine:c { #1 } } {
939     \prg_return_true:
940   } {
941     \prg_return_false:
942   }
943 }
944 \prg_new_conditional:Nnn \CDR_if_block_engine:V { p, T, F, TF } {
945   \cs_if_exist:cTF { \CDR_block_engine:V #1 } {
946     \prg_return_true:
947   } {
948     \prg_return_false:

```

```

949 }
950 }

```

13.4 Default code engine

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

```

951 \CDRCodeEngineNew { default } { #2 }

```

13.5 Default block engine

The default block engine does nothing.

```

952 \CDRBlockEngineNew { default } { } { }

```

13.6 efbox code engine

```

953 \AtBeginDocument {
954   \@ifpackageloaded{efbox} {
955     \CDRCodeEngineNew {efbox} {
956       \efbox[#1]{#2}%
957     }
958   }
959 }

```

13.7 Block mode default engine

```

960 \CDRBlockEngineNew {} {
961 } {
962 }

```

13.8 tcolorbox related engine

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

14 \CDRCode function

14.1 API

\CDR@Sp

\CDR@Sp

Private method to eventually make the space character visible using \FancyVerbSpace base on `showspaces` value.

```

963 \cs_new:Npn \CDR@DefineSp {
964   \CDR_if_tag_truthy:cTF { showspaces } {
965     \cs_set:Npn \CDR@Sp {{\FancyVerbSpace}}
966   } {
967     \cs_set_eq:NN \CDR@Sp \space
968   }
969 }

```

\CDRCode

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

Public method to declare inline code.

14.2 Storage

`\l_CDR_tag_tl` To store the tag given.

```
970 \tl_new:N \l_CDR_tag_tl
```

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

14.3 `__code l3keys` module

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

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

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

```
972   tag .tl_set:N = \l_CDR_tag_tl,
973   tag .value_required:n = true,
```

🔴 **engine options=***<engine options>* options forwarded to the engine. They are appended to the options given with key *<engine name>* engine options.

```
974   engine~options .code:n = \CDR_tag_set:,
975   engine~options .value_required:n = true,
```

🔴 **__initialize** initialize

```
976   __initialize .meta:n = {
977     tag = default,
978     engine~options = ,
979   },
980   __initialize .value_forbidden:n = true,
981 }
```

14.4 Implementation

`\CDR_code_format:` `\CDR_code_format:`

Private utility to setup the formatting.

```
982 \cs_new:Npn \CDR_brace_if_contains_comma:n #1 {
983   \tl_if_in:nnTF { #1 } { , } { { #1 } } { #1 }
984 }
985 \cs_generate_variant:Nn \CDR_brace_if_contains_comma:n { V }
986 \cs_new:Npn \CDR_code_format: {
987   \frenchspacing
988   \CDR_tag_get:cN { baselinestretch } \l_CDR_tl
989   \tl_if_eq:NnF \l_CDR_tl { auto } {
990     \exp_args:NNV
991     \def \baselinestretch \l_CDR_tl
992   }
993   \CDR_tag_get:cN { fontfamily } \l_CDR_tl
994   \tl_if_eq:NnT \l_CDR_tl { tt } { \tl_set:Nn \l_CDR_tl { lmtt } }
```

```

995 \exp_args:NV
996 \fontfamily \l_CDR_tl
997 \clist_map_inline:nn { series, shape } {
998   \CDR_tag_get:cN { font##1 } \l_CDR_tl
999   \tl_if_eq:NnF \l_CDR_tl { auto } {
1000     \exp_args:NnV
1001     \use:c { font##1 } \l_CDR_tl
1002   }
1003 }
1004 \CDR_tag_get:cN { fontsize } \l_CDR_tl
1005 \tl_if_eq:NnF \l_CDR_tl { auto } {
1006   \tl_use:N \l_CDR_tl
1007 }
1008 \selectfont
1009 % \@noligs ?? this is in fancyvrb but does not work here as is
1010 }

```

\CDR@Callback \CDR@Callback {<pyg sty path>} {<pyg tex path>}

Private utility to load a style or a pygmented source.

```

1011 \cs_new:Npn \CDR@Callback #1 #2 {
1012   \typeout{CDR@Callback:#1/#2/}
1013   \tl_if_empty:nF { #1 } {
1014     \input{#1}
1015     \CDR@StyleUseTag
1016   }
1017   \tl_if_empty:nF { #2 } {
1018     \input{#2}
1019   }
1020 }

```

\CDR_code:n \CDR_code:n <delimiter>

Main utility used by \CDRCode.

```

1021 \cs_new:Npn \CDR_code:n #1 {
1022   \CDR_if_tag_truthy:cTF {pygments} {
1023     \cs_set:Npn \CDR@StyleUseTag {
1024       \CDR@StyleUse { \CDR_tag_get:c { style } }
1025       \cs_set_eq:NN \CDR@StyleUseTag \prg_do_nothing:
1026     }
1027     \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1028       __fancyvrb,
1029     }
1030     \CDR_tag_keys_set:nV { __local } \l_CDR_keyval_tl
1031     \DefineShortVerb { #1 }
1032     \SaveVerb [
1033       aftersave = {
1034         \exp_args:Nx \UndefineShortVerb { #1 }
1035         \lua_now:n { CDR:highlight_code_prepare() }
1036         \CDR_tag_get:cN {lang} \l_CDR_tl
1037         \lua_now:n { CDR:highlight_set_var('lang') }

```

```

1038 \CDR_tag_get:cN {cache} \l_CDR_tl
1039 \lua_now:n { CDR:highlight_set_var('cache') }
1040 \CDR_tag_get:cN {debug} \l_CDR_tl
1041 \lua_now:n { CDR:highlight_set_var('debug') }
1042 \CDR_tag_get:cN {style} \l_CDR_tl
1043 \lua_now:n { CDR:highlight_set_var('style') }
1044 \lua_now:n { CDR:highlight_set_var('source', 'FV@SV@CDR@Source') }
1045 \FV@UseKeyValues
1046 \frenchspacing
1047 % \FV@SetupFont Break
1048 \FV@DefineWhiteSpace
1049 \FancyVerbDefineActive
1050 \FancyVerbFormatCom
1051 \CDR_code_format:
1052 \CDR@DefineSp
1053 \CDR_tag_get:c { format }
1054 \CDR@DefineSp
1055 \CDR@CodeEngineApply {
1056 \makeatletter
1057 \CDR@StyleIfExist { \l_CDR_tl } {
1058 \CDR@StyleUseTag
1059 \lua_now:n { CDR:highlight_source(false, true) }
1060 } {
1061 \lua_now:n { CDR:highlight_source(true, true) }
1062 }
1063 \makeatother
1064 }
1065 \group_end:
1066 }
1067 ] { CDR@Source } #1
1068 } {
1069 \exp_args:NV \fvset \l_CDR_keyval_tl
1070 \DefineShortVerb { #1 }
1071 \SaveVerb [
1072 aftersave = {
1073 \UndefineShortVerb { #1 }
1074 \cs_set_eq:NN \CDR@FormattingPrep \FV@FormattingPrep
1075 \cs_set:Npn \FV@FormattingPrep {
1076 \CDR@FormattingPrep
1077 \CDR_tag_get:c { format }
1078 }
1079 \CDR@CodeEngineApply { \mbox {
1080 \FV@UseKeyValues
1081 \FV@FormattingPrep
1082 \FV@SV@CDR@Code
1083 } }
1084 \group_end:
1085 }
1086 ] { CDR@Code } #1
1087 }
1088 }

1089 \NewDocumentCommand \CDRCode { 0{ } } {
1090 \group_begin:
1091 \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {

```

```

1092     \prg_return_false:
1093 }
1094 \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1095     __code, default.code, __pygments, default,
1096 }
1097 \CDR_tag_keys_set_known:nnN { __local } { #1 } \l_CDR_keyval_tl
1098 \CDR_tag_provide_from_keyval:V \l_CDR_keyval_tl
1099 \CDR_tag_keys_set_known:nVN { __local } \l_CDR_keyval_tl \l_CDR_keyval_tl
1100 \exp_args:NNV
1101 \def \FV@KeyValues \l_CDR_keyval_tl
1102 \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1103     __fancyvrb,
1104 }
1105 \CDR_tag_keys_set:nV { __local } \l_CDR_keyval_tl
1106 \CDR_tag_inherit:cf { __local } {
1107     \tl_if_empty:NF \l_CDR_tag_tl { \l_CDR_tag_tl, }
1108     __code, default.code, __pygments, default, __fancyvrb,
1109 }
1110 \CDR_code:n
1111 }
1112 \tl_new:N \l_CDR_style_tl
1113 \cs_set:Npn \CDR_code:n #1 {
1114     \CDR_if_tag_truthy:cTF {pygments} {
1115         \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1116             __fancyvrb,
1117         }
1118         \CDR_tag_keys_set:nV { __local } \l_CDR_keyval_tl
1119         \DefineShortVerb { #1 }
1120         \SaveVerb [
1121             aftersave = {
1122                 \exp_args:Nx \UndefineShortVerb { #1 }
1123                 \lua_now:n { CDR:highlight_code_prepare() }
1124                 \CDR_tag_get:cN {lang} \l_CDR_tl
1125                 \lua_now:n { CDR:highlight_set_var('lang') }
1126                 \CDR_tag_get:cN {cache} \l_CDR_tl
1127                 \lua_now:n { CDR:highlight_set_var('cache') }
1128                 \CDR_tag_get:cN {debug} \l_CDR_tl
1129                 \lua_now:n { CDR:highlight_set_var('debug') }
1130                 \CDR_tag_get:cN {style} \l_CDR_style_tl
1131                 \lua_now:n { CDR:highlight_set_var('style', 'l_CDR_style_tl') }
1132                 \lua_now:n { CDR:highlight_set_var('source', 'FV@SV@CDR@Source') }
1133                 \exp_args:NNV
1134                 \def \FV@KeyValues \l_CDR_keyval_tl
1135                 \typeout{DEBUG::::: \l_CDR_keyval_tl}
1136                 \FV@UseKeyValues
1137                 \frenchspacing
1138                 % \FV@SetupFont Break
1139                 \FV@DefineWhiteSpace
1140                 \FancyVerbDefineActive
1141                 \FancyVerbFormatCom
1142                 \CDR@DefineSp
1143                 \CDR_code_format:
1144                 \CDR_tag_get:c { format }
1145 \typeout{DEBUG.....1.\CDR_tag_get:c { style }.\l_CDR_style_tl.\CDR_tag_get:c { sho

```

```

1146     \CDR@CodeEngineApply {
1147         \makeatletter
1148         \CDR@StyleIfExist { \l_CDR_style_tl } {
1149             \CDR@StyleUseTag
1150             \lua_now:n { CDR:highlight_source(false, true) }
1151 \typeout{DEBUG.....2A}
1152         } {
1153             \lua_now:n { CDR:highlight_source(true, true) }
1154 \typeout{DEBUG.....2B}
1155         }
1156         \makeatother
1157     }
1158 \typeout{DEBUG.....3}
1159 \group_end:
1160 }
1161 ] { CDR@Source } #1
1162 } {
1163     \DefineShortVerb { #1 }
1164     \SaveVerb [
1165         aftersave = {
1166             \UndefineShortVerb { #1 }
1167             \cs_set_eq:NN \CDR@FormattingPrep \FV@FormattingPrep
1168             \cs_set:Npn \FV@FormattingPrep {
1169                 \CDR@FormattingPrep
1170                 \CDR_tag_get:c { format }
1171             }
1172             \CDR@CodeEngineApply { A \mbox { a
1173                 \exp_args:NNV
1174                 \def \FV@KeyValues \l_CDR_keyval_tl
1175                 \FV@UseKeyValues
1176                 \FV@FormattingPrep
1177                 \@nameuse{FV@SV@CDR@Code}
1178                 z } Z }
1179             \group_end:
1180         }
1181     ] { CDR@Code } #1
1182 }
1183 }

1184 \RenewDocumentCommand \CDRCode { 0{} } {
1185     \group_begin:
1186     \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
1187         \prg_return_false:
1188     }
1189     \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1190         __code, default.code, __pygments, default,
1191     }
1192     \CDR_tag_keys_set_known:nnN { __local } { #1 } \l_CDR_keyval_tl
1193     \CDR_tag_provide_from_keyval:V \l_CDR_keyval_tl
1194     \CDR_tag_keys_set_known:nVN { __local } \l_CDR_keyval_tl \l_CDR_keyval_tl
1195     \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1196         __fancyvrb,
1197     }
1198     \CDR_tag_keys_set:nV { __local } \l_CDR_keyval_tl
1199     \CDR_tag_inherit:cf { __local } {

```

```

1200     \tl_if_empty:NF \l_CDR_tag_tl { \l_CDR_tag_tl, }
1201     __code, default.code, __pygments, default, __fancyvrb,
1202   }
1203   \fvset{showspaces}
1204   \CDR_code:n
1205 }

```

15 CDRBlock environment

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

15.1 Storage

`\l_CDR_block_prop`

```

1206 \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.

```

1207 \CDR_tag_keys_define:nm { __block } {


```

 **no export[=true|false]** to ignore this code chunk at export time.

```

1208   no-export .code:n = \CDR_tag_boolean_set:x { #1 },
1209   no-export .default:n = true,


```

 **no export format=<format commands>** a format appended to tags format and numbers format when no export is true.. Initially empty.

```

1210   no-export~format .code:n = \CDR_tag_set:,
1211   no-export~format .value_required:n = true,


```

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

```

1212   test .code:n = \CDR_tag_boolean_set:x { #1 },
1213   test .default:n = true,


```

 **engine options=<engine options>** options forwarded to the engine. They are appended to the options given with key <engine name> engine options. Mainly a convenient user interface shortcut.

```

1214   engine-options .code:n = \CDR_tag_set:,
1215   engine-options .value_required:n = true,

```

 **__initialize** initialize



```

1216 __initialize .meta:n = {
1217     no~export = false,
1218     no~export~format = ,
1219     test = false,
1220     engine~options = ,
1221 },
1222 __initialize .value_forbidden:n = true,
1223 }

```

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.

```

1224 \clist_map_inline:nn { i, ii, iii, iv } {
1225     \cs_set_eq:cc { CDR@ListProcessLine@ #1 } { FV@ListProcessLine@ #1 }
1226 }
1227 \cs_new:Npn \CDR_process_line:n #1 {
1228     \str_set:Nn \l_CDR_str { #1 }
1229     \lua_now:n {CDR:record_line('l_CDR_str')}
1230 }

1231 \def\FVB@CDRBlock #1 {
1232     \@sphack
1233     \group_begin:
1234     \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
1235         \prg_return_true:
1236     }
1237     \CDR_tag_keys_set:nn { __block } { __initialize }

```

By default, this code chunk will have the same list of tags as the last code block or last \CDRExport stored in \g_CDR_tags_clist.

```

1238 \clist_set_eq:NN \l_CDR_tags_clist \g_CDR_tags_clist
1239 \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1240     __block, __pygments.block, default.block,
1241     __pygments, default,
1242 }
1243 \exp_args:NnV
1244 \CDR_tag_keys_set_known:nnN { __local } \FV@KeyValues \l_CDR_keyval_tl
1245 \CDR_tag_provide_from_keyval:V \l_CDR_keyval_tl
1246 \exp_args:NnV
1247 \CDR_tag_keys_set_known:nnN { __local } \l_CDR_keyval_tl \l_CDR_keyval_tl
1248 \clist_if_empty:NT \l_CDR_tags_clist {
1249     \PackageWarning
1250         { coder }
1251         { No~(default)~tags~provided }
1252 }

```

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

```

1253 \clist_map_inline:Nn \l_CDR_tags_clist {
1254   \CDR_if_truthy:ccT { ##1 } { pygments } {
1255     \clist_map_break:n {
1256       \bool_set_true:N \l_CDR_pygments_bool
1257     }
1258   }
1259 }
1260 \bool_if:NTF \l_CDR_pygments_bool {
1261   \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1262     __fancyvrb.number
1263   }
1264   \CDR_tag_keys_set_known:nVN { __local } \l_CDR_keyval_tl \l_CDR_keyval_tl
1265   \exp_args:NV \fvset \l_CDR_keyval_tl
1266   \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1267     __fancyvrb, __fancyvrb.block
1268   }
1269   \exp_args:NnV
1270   \CDR_tag_keys_set:nn { __local } \l_CDR_keyval_tl

```

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

```

1271 \CDR_tag_inherit:cf { __local } {
1272   \l_CDR_tags_clist,
1273   __block, default.block, __pygments.block, __fancyvrb.block,
1274   __pygments, default, __fancyvrb,
1275 }
1276 \lua_now:n {
1277   CDR:hilight_block_prepare('l_CDR_tags_clist')
1278 }
1279 \def\FV@KeyValues{
1280 \CDR_tag_get:cN {lang} \l_CDR_tl
1281 \lua_now:n { CDR:hilight_set_var('lang') }
1282 \CDR_tag_get:cN {cache} \l_CDR_tl
1283 \lua_now:n { CDR:hilight_set_var('cache') }
1284 \CDR_tag_get:cN {debug} \l_CDR_tl
1285 \lua_now:n { CDR:hilight_set_var('debug') }
1286 \CDR_tag_get:cN {style} \l_CDR_tl
1287 \lua_now:n { CDR:hilight_set_var('style') }
1288 \CDR@StyleIfExist { \l_CDR_tl } { } {
1289   ???
1290 }
1291 } {
1292 \exp_args:NNV
1293 \def \FV@KeyValues \l_CDR_keyval_tl
1294 \CDR_tag_inherit:cf { __local } {
1295   \l_CDR_tags_clist,
1296   __block, default.block, __pygments.block, __fancyvrb.block,
1297   __pygments, default, __fancyvrb, __fancyvrb.all,
1298 }
1299 }
1300 \exp_args:Nnx
1301 \CDR_if_tag_truthy:cTF {no-export} {
1302   \bool_if:NT \l_CDR_pygments_bool {

```

```

1303     \cs_map_inline:nn { i, ii, iii, iv } {
1304         \cs_set:cpn { FV@ListProcessLine@ #####1 } ##1 {
1305             \CDR_hilight_record:n { ##1 }
1306         }
1307     }
1308 }
1309 } {
1310     \bool_if:NTF \l_CDR_pygments_bool {
1311         \cs_map_inline:nn { i, ii, iii, iv } {
1312             \cs_set:cpn { FV@ListProcessLine@ #####1 } ##1 {
1313                 \CDR_hilight_record:n { ##1 }
1314                 \CDR_export_record:n { ##1 }
1315             }
1316         }
1317     } {
1318         \cs_map_inline:nn { i, ii, iii, iv } {
1319             \cs_set:cpn { FV@ListProcessLine@ #####1 } ##1 {
1320                 \CDR_export_record:n { ##1 }
1321                 \use:c { CDR@ListProcessLine@ #####1 } { ##1 }
1322             }
1323         }
1324     }
1325 }
1326 \CDR_tag_get:cN { \l_CDR_engine_tl~engine~options } \l_CDR_options_tl
1327 \tl_if_empty:NTF \l_CDR_options_tl {

```

No `\begin` works here. Why? This may be related to the required `\relax` below.

```

1328     \use:c { \CDR_block_engine:V \l_CDR_engine_tl }
1329 } {
1330     \exp_args:NnNV
1331     \use:c { \CDR_block_engine:V \l_CDR_engine_tl }
1332     [ \l_CDR_options_tl ]
1333 }
1334 \relax
1335 \cs_set_eq:NN \CDR@FormattingPrep \FV@FormattingPrep
1336 \cs_set:Npn \FV@FormattingPrep {
1337     \CDR@FormattingPrep
1338     \CDR_tag_get:c { format }
1339 }
1340 \FV@VerbatimBegin
1341 \FV@Scan
1342 }
1343 \def\FVE@CDRBlock{
1344     \FV@VerbatimEnd
1345     \bool_if:NT \l_CDR_pygments_bool {
1346         \lua_now:n { CDR:hilight_source(true, true) }
1347     }
1348     \use:c { end \CDR_block_engine:V \l_CDR_engine_tl }
1349     \group_end:
1350     \@esphack
1351 }
1352 \DefineVerbatimEnvironment{CDRBlock}{CDRBlock}{-}
1353

```

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.

```

1354 \def\FVB@CDR@Pyg@Verbatim #1 {
1355   \group_begin:
1356   \FV@VerbatimBegin
1357   \FV@Scan
1358 }
1359 \def\FVE@CDR@Pyg@Verbatim{
1360   \FV@VerbatimEnd
1361   \group_end:
1362 }
1363 \DefineVerbatimEnvironment{CDR@Pyg@Verbatim}{CDR@Pyg@Verbatim}{}
1364

```

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 pygments is used. *Implementation details:* we assume that if \l_CDR_tags_clist is not empty then we are in a CDRBlock environment.

```

1365 \prg_new_conditional:Nnn \CDR_if_record: { T, F, TF } {
1366   \clist_if_empty:NTF \l_CDR_tags_clist {
1367     \prg_return_false:
1368   } {
1369     \CDR_if_use_pygments:TF {
1370       \prg_return_true:
1371     } {
1372       \prg_return_false:
1373     }
1374   }
1375 }

1376 \cs_new:Npn \CDR_process_recordNO: {
1377   \tl_put_right:Nx \l_CDR_recorded_tl { \the\verbatim@line \iow_newline: }
1378   \group_begin:
1379   \tl_set:Nx \l_tmpa_tl { \the\verbatim@line }
1380   \lua_now:e {CDR.records.append([==[\l_tmpa_tl]==])}
1381   \group_end:
1382 }

```

```

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

```

```

1383 \newenvironment{CDR}{
1384   \def \verbatim@processline {
1385     \group_begin:

```

```

1386 \CDR_process_line_code_append:
1387 \group_end:
1388 }
1389 % \CDR_if_show_code:T {
1390 % \CDR_if_use_minted:TF {
1391 % \Needspace* { 2\baselineskip }
1392 % } {
1393 % \frenchspacing\@vobeyspaces
1394 % }
1395 % }
1396 } {
1397 \CDR:nNTF { lang } \l_tmpa_tl {
1398 \tl_if_empty:NT \l_tmpa_tl {
1399 \clist_map_inline:Nn \l_CDR_clist {
1400 \CDR:nnNT { ##1 } { lang } \l_tmpa_tl {
1401 \tl_if_empty:NF \l_tmpa_tl {
1402 \clist_map_break:
1403 }
1404 }
1405 }
1406 \tl_if_empty:NT \l_tmpa_tl {
1407 \tl_set:Nn \l_tmpa_tl { tex }
1408 }
1409 }
1410 } {
1411 \tl_set:Nn \l_tmpa_tl { tex }
1412 }
1413 % NO WAY
1414 \clist_map_inline:Nn \l_CDR_clist {
1415 \CDR:gput:nnV { ##1 } { lang } \l_tmpa_tl
1416 }
1417 }

```

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

```

1418 \newenvironment{CDR_M}{
1419 \setkeys { FV } { firstnumber=last, }
1420 \clist_if_empty:NTF \l_CDR_clist {
1421 \exp_args:Nnx \setkeys { FV } {
1422 firstnumber=\CDR_int_use:n { },
1423 } } {
1424 \clist_map_inline:Nn \l_CDR_clist {
1425 \exp_args:Nnx \setkeys { FV } {
1426 firstnumber=\CDR_int_use:n { ##1 },
1427 }
1428 \clist_map_break:
1429 } }
1430 \iow_open:Nn \minted@code { \jobname.pyg }
1431 \tl_set:Nn \l_CDR_line_tl {
1432 \tl_set:Nx \l_tmpa_tl { \the\verbatim@line }
1433 \exp_args:NNV \iow_now:Nn \minted@code \l_tmpa_tl
1434 }
1435 } {

```

```

1436 \CDR_if_show_code:T {
1437   \CDR_if_use_minted:TF {
1438     \iow_close:N \minted@code
1439     \vspace* { \dimexpr -\topsep-\parskip }
1440     \tl_if_empty:NF \l_CDR_info_tl {
1441       \tl_use:N \l_CDR_info_tl
1442       \vspace* { \dimexpr -\topsep-\parskip-\baselineskip }
1443       \par\noindent
1444     }
1445     \exp_args:NV \minted@pygmentize \l_tmpa_tl
1446     \DeleteFile { \jobname.pyg }
1447     \vspace* { \dimexpr -\topsep -\partopsep }
1448   } {
1449     \@esphack
1450   }
1451 }
1452 }

```

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

```

1453 \newenvironment{CDR_P}{
1454   \if_mode_vertical:
1455     \noindent
1456   \else
1457     \vspace*{ \topsep }
1458     \par\noindent
1459   \fi
1460   \CDR_gset_chunks:
1461   \tl_if_empty:NTF \g_CDR_chunks_tl {
1462     \CDR_if:nTF {show_lineno} {
1463       \CDR_if_use_margin:TF {

```

No chunk name, line numbers in the margin

```

1464     \tl_set:Nn \l_CDR_info_tl {
1465       \hbox_overlap_left:n {
1466         \CDR:n { format/code }
1467         {
1468           \CDR:n { format/name }
1469           \CDR:n { format/lineno }
1470           \clist_if_empty:NTF \l_CDR_clist {
1471             \CDR_int_use:n { }
1472           } {
1473             \clist_map_inline:Nn \l_CDR_clist {
1474               \CDR_int_use:n { ##1 }
1475             \clist_map_break:
1476           }
1477         }
1478       }
1479       \hspace*{1ex}
1480     }
1481   }
1482 } {

```

No chunk name, line numbers not in the margin

```

1483 \tl_set:Nn \l_CDR_info_tl {
1484 {
1485 \CDR:n { format/code }
1486 {
1487 \CDR:n { format/name }
1488 \CDR:n { format/lineno }
1489 \hspace*{3ex}
1490 \hbox_overlap_left:n {
1491 \clist_if_empty:NTF \l_CDR_clist {
1492 \CDR_int_use:n { }
1493 } {
1494 \clist_map_inline:Nn \l_CDR_clist {
1495 \CDR_int_use:n { ##1 }
1496 \clist_map_break:
1497 }
1498 }
1499 }
1500 \hspace*{1ex}
1501 }
1502 }
1503 }
1504 }
1505 } {

```

No chunk name, no line numbers

```

1506 \tl_clear:N \l_CDR_info_tl
1507 }
1508 } {
1509 \CDR_if:nTF {show_lineno} {

```

Chunk names, line numbers, in the margin

```

1510 \tl_set:Nn \l_CDR_info_tl {
1511 \hbox_overlap_left:n {
1512 \CDR:n { format/code }
1513 {
1514 \CDR:n { format/name }
1515 \g_CDR_chunks_tl :
1516 \hspace*{1ex}
1517 \CDR:n { format/lineno }
1518 \clist_map_inline:Nn \l_CDR_clist {
1519 \CDR_int_use:n { #####1 }
1520 \clist_map_break:
1521 }
1522 }
1523 \hspace*{1ex}
1524 }
1525 \tl_set:Nn \l_CDR_info_tl {
1526 \hbox_overlap_left:n {
1527 \CDR:n { format/code }
1528 {
1529 \CDR:n { format/name }
1530 \CDR:n { format/lineno }

```

```

1531         \clist_map_inline:Nn \l_CDR_clist {
1532             \CDR_int_use:n { ####1 }
1533             \clist_map_break:
1534         }
1535     }
1536     \hspace*{1ex}
1537 }
1538 }
1539 }
1540 } {

```

Chunk names, no line numbers, in the margin

```

1541     \tl_set:Nn \l_CDR_info_tl {
1542         \hbox_overlap_left:n {
1543             \CDR:n { format/code }
1544             {
1545                 \CDR:n { format/name }
1546                 \g_CDR_chunks_tl :
1547             }
1548             \hspace*{1ex}
1549         }
1550     \tl_clear:N \l_CDR_info_tl
1551 }
1552 }
1553 }
1554 \CDR_if_use_minted:F {
1555     \tl_set:Nn \l_CDR_line_tl {
1556         \noindent
1557         \hbox_to_wd:nn { \textwidth } {
1558             \tl_use:N \l_CDR_info_tl
1559             \CDR:n { format/code }
1560             \the\verbatim@line
1561             \hfill
1562         }
1563         \par
1564     }
1565     \@bsphack
1566 }
1567 } {
1568     \vspace*{ \topsep }
1569     \par
1570     \@esphack
1571 }

```

18 Management

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

```

1572 \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.

```

1573 \prg_new_conditional:Nnn \CDR_if_show_code: { T, F, TF } {
1574   \bool_if:nTF {
1575     \g_CDR_in_impl_bool && !\g_CDR_with_impl_bool
1576   } {
1577     \prg_return_false:
1578   } {
1579     \prg_return_true:
1580   }
1581 }
```

\g_CDR_with_impl_bool

```

1582 \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.

```

1583 \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.

```

1584 \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.

```

1585 \prg_new_conditional:Nnn \CDR_if_use_minted: { T, F, TF } {
1586   \bool_if:NTF \g_CDR_use_minted_bool
1587   { \prg_return_true: }
1588   { \prg_return_false: }
1589 }
```

_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.

```

1590 \cs_set:Npn \_CDR_minted_on: {
1591   \bool_gset_true:N \g_CDR_minted_on_bool
1592   \RequirePackage{minted}
1593   \setkeys{ minted@opt@g } { linenos=false }
1594   \minted@def@opt{post-processor}
1595   \minted@def@opt{post-processor~args}
```

```

1596 \pretocmd\minted@inputpyg{
1597   \CDR@postprocesspyg {\minted@outputdir\minted@infile}
1598 }{\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

```

1599 \newcommand{\CDR@postprocesspyg}[1]{%
1600   \group_begin:
1601   \tl_set:Nx \l_tmpa_tl {\CDR:n { post_processor } }
1602   \tl_if_empty:NF \l_tmpa_tl {

```

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

```

1603     \tl_set:Nx \l_tmpb_tl {\CDR:n { post_processor_args } }
1604     \exp_args:Nx
1605     \sys_shell_now:n {
1606       python3\space
1607       \l_tmpa_tl\space
1608       ##1\space
1609       \l_tmpb_tl
1610     }
1611   }
1612   \group_end:
1613 }
1614 }

1615 %\AddToHook { begindocument / end } {
1616 %   \cs_set_eq:NN \_CDR_minted_on: \prg_do_nothing:
1617 %}

```

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

```

1618 \ProvideDocumentCommand{\CDREmph}{m}{\textcolor{red}{#1}}

```

```

\CDRPreamble \CDRPreamble {\variable} {\file name}

```

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

```

1619 \DeclareDocumentCommand \CDRPreamble { m m } {
1620   \msg_info:nnn
1621   { coder }
1622   { :n }
1623   { Reading-preamble-from-file-"#2". }
1624   \group_begin:
1625   \tl_set:Nn \l_tmpa_tl { #2 }
1626   \exp_args:NNNx
1627   \group_end:
1628   \tl_set:Nx #1 { \lua_now:n {CDR.print_file_content('l_tmpa_tl')} }
1629 }

```

20 Section separators

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

21 Finale

```

1630 \newcounter{CDR@impl@page}
1631 \DeclareDocumentCommand \CDRImplementation {} {
1632   \bool_if:NF \g_CDR_with_impl_bool {
1633     \clearpage
1634     \bool_gset_true:N \g_CDR_in_impl_bool
1635     \let\CDR@old@part\part
1636     \DeclareDocumentCommand\part{som}{}
1637     \let\CDR@old@section\section
1638     \DeclareDocumentCommand\section{som}{}
1639     \let\CDR@old@subsection\subsection
1640     \DeclareDocumentCommand\subsection{som}{}
1641     \let\CDR@old@subsubsection\subsubsection
1642     \DeclareDocumentCommand\subsubsection{som}{}
1643     \let\CDR@old@paragraph\paragraph
1644     \DeclareDocumentCommand\paragraph{som}{}
1645     \let\CDR@old@subparagraph\subparagraph
1646     \DeclareDocumentCommand\subparagraph{som}{}
1647     \cs_if_exist:NT \refsection{ \refsection }
1648     \setcounter{ CDR@impl@page }{ \value{page} }
1649   }
1650 }
1651 \DeclareDocumentCommand\CDRFinale {} {
1652   \bool_if:NF \g_CDR_with_impl_bool {
1653     \clearpage
1654     \bool_gset_false:N \g_CDR_in_impl_bool
1655     \let\part\CDR@old@part
1656     \let\section\CDR@old@section
1657     \let\subsection\CDR@old@subsection
1658     \let\subsubsection\CDR@old@subsubsection
1659     \let\paragraph\CDR@old@paragraph
1660     \let\subparagraph\CDR@old@subparagraph
1661     \setcounter { page } { \value{ CDR@impl@page } }
1662   }
1663 }
1664 \cs_set_eq:NN \CDR_line_number: \prg_do_nothing:

```

22 Finale

```

1665 \AddToHook { cmd/FancyVerbFormatLine/before } {
1666   \CDR_line_number:
1667 }
1668 \AddToHook { shipout/before } {

```

```

1669 \tl_gclear:N \g_CDR_chunks_tl
1670 }

1671 % =====
1672 % Auxiliary:
1673 %   finding the widest string in a comma
1674 %   separated list of strings delimited by parenthesis
1675 % =====
1676
1677 % arguments:
1678 % #1) text: a comma separated list of strings
1679 % #2) formatter: a macro to format each string
1680 % #3) dimension: will hold the result
1681
1682 \cs_new:Npn \CDRWidest (#1) #2 #3 {
1683   \group_begin:
1684   \dim_set:Nn #3 { 0pt }
1685   \clist_map_inline:nn { #1 } {
1686     \hbox_set:Nn \l_tmpa_box { #2{##1} }
1687     \dim_set:Nn \l_tmpa_dim { \dim_eval:n { \box_wd:N \l_tmpa_box } }
1688     \dim_compare:nNnT { #3 } < { \l_tmpa_dim } {
1689       \dim_set_eq:NN #3 \l_tmpa_dim
1690     }
1691   }
1692   \exp_args:NNNV
1693   \group_end:
1694   \dim_set:Nn #3 #3
1695 }
1696 \ExplSyntaxOff
1697

```

23 pygmentex implementation

```

1698 % =====
1699 % fancyvrb new commands to append to a file
1700 % =====
1701
1702 % See http://tex.stackexchange.com/questions/47462/inputenc-error-with-unicode-chars-and-verbatim
1703
1704 \ExplSyntaxOn
1705
1706 \seq_new:N \l_CDR_records_seq
1707
1708 \long\def\unexpanded@write#1#2{\write#1{\unexpanded{#2}}}%
1709
1710 \def\CDRAppend{\FV@Environment{}}{CDRAppend}}
1711
1712 \def\FVB@CDRAppend#1{%
1713   \@bsphack
1714   \begingroup
1715     \seq_clear:N \l_CDR_records_seq
1716     \FV@UseKeyValues
1717     \FV@DefineWhiteSpace

```

```

1718 \def\FV@Space{\space}%
1719 \FV@DefineTabOut
1720 \def\FV@ProcessLine{%##1
1721   \seq_put_right:Nn \l_CDR_records_seq { ##1 }%
1722   \immediate\unexpanded@write#1{##1}
1723 }%
1724 \let\FV@FontScanPrep\relax
1725 \let\@noligs\relax
1726 \FV@Scan
1727 }
1728 \def\FVE@CDRAAppend{
1729   \seq_use:Nn \l_CDR_records_seq /
1730   \endgroup
1731   \@esphack
1732 }
1733 \DefineVerbatimEnvironment{CDRAAppend}{CDRAAppend}{}
1734
1735 \DeclareDocumentEnvironment { Inline } { m } {
1736   \clist_clear:N \l_CDR_clist
1737   \keys_set:nn { CDR_code } { #1 }
1738   \clist_map_inline:Nn \l_CDR_clist {
1739     \CDR_int_if_exist:nF { ##1 } {
1740       \CDR_int_new:nn { ##1 } { 1 }
1741       \seq_new:c { g/CDR/chunks/##1 }
1742     }
1743   }
1744   \CDR_if:nT {reset} {
1745     \CDR_clist_map_inline:Nnn \l_CDR_clist {
1746       \CDR_int_gset:nn { } 1
1747     } {
1748       \CDR_int_gset:nn { ##1 } 1
1749     }
1750   }
1751   \tl_clear:N \l_CDR_code_name_tl
1752   \clist_map_inline:Nn \l_CDR_clist {
1753     \prop_concat:ccc
1754       {g/CDR/Code/}
1755       {g/CDR/Code/##1/}
1756       {g/CDR/Code/}
1757     \tl_set:Nn \l_CDR_code_name_tl { ##1 }
1758     \clist_map_break:
1759   }
1760   \int_gset:Nn \g_CDR_int
1761     { \CDR_int_use:n { \l_CDR_code_name_tl } }
1762   \tl_clear:N \l_CDR_info_tl
1763   \tl_clear:N \l_CDR_name_tl
1764   \tl_clear:N \l_CDR_recorded_tl
1765   \tl_clear:N \l_CDR_chunks_tl
1766   \cs_set:Npn \verbatim@processline {
1767     \CDR_process_record:
1768   }
1769   \CDR_if_show_code:TF {
1770     \exp_args:NNx
1771     \skip_set:Nn \parskip { \CDR:n { parskip } }

```

```

1772 \clist_if_empty:NTF \l_CDR_clist {
1773   \tl_gclear:N \g_CDR_chunks_tl
1774 } {
1775   \clist_set_eq:NN \l_tmpa_clist \l_CDR_clist
1776   \clist_sort:Nn \l_tmpa_clist {
1777     \str_compare:nNnTF { ##1 } > { ##2 } {
1778       \sort_return_swapped:
1779     } {
1780       \sort_return_same:
1781     }
1782   }
1783   \tl_set:Nx \l_tmpa_tl { \clist_use:Nn \l_tmpa_clist , }
1784   \CDR_if:nT {show_name} {
1785     \CDR_if:nT {use_margin} {
1786       \CDR_if:nT {only_top} {
1787         \tl_if_eq:NNT \l_tmpa_tl \g_CDR_chunks_tl {
1788           \tl_gset_eq:NN \g_CDR_chunks_tl \l_tmpa_tl
1789           \tl_clear:N \l_tmpa_tl
1790         }
1791       }
1792       \tl_if_empty:NF \l_tmpa_tl {
1793         \tl_set:Nx \l_CDR_chunks_tl {
1794           \clist_use:Nn \l_CDR_clist ,
1795         }
1796         \tl_set:Nn \l_CDR_name_tl {
1797           {
1798             \CDR:n { format/name }
1799             \l_CDR_chunks_tl :
1800             \hspace*{1ex}
1801           }
1802         }
1803       }
1804     }
1805     \tl_if_empty:NF \l_tmpa_tl {
1806       \tl_gset_eq:NN \g_CDR_chunks_tl \l_tmpa_tl
1807     }
1808   }
1809 }
1810 \if_mode_vertical:
1811 \else:
1812 \par
1813 \fi:
1814 \vspace{ \CDR:n { sep } }
1815 \noindent
1816 \frenchspacing
1817 \@vobeyspaces
1818 \normalfont\ttfamily
1819 \CDR:n { format/code }
1820 \hyphenchar\font\m@ne
1821 \@noligs
1822 \CDR_if_record:F {
1823   \cs_set_eq:NN \CDR_process_record: \prg_do_nothing:
1824 }
1825 \CDR_if_use_minted:F {

```

```

1826 \CDR_if:nT {show_lineno} {
1827   \CDR_if:nTF {use_margin} {
1828     \tl_set:Nn \l_CDR_info_tl {
1829       \hbox_overlap_left:n {
1830         {
1831           \l_CDR_name_tl
1832           \CDR:n { format/name }
1833           \CDR:n { format/lineno }
1834           \int_use:N \g_CDR_int
1835           \int_gincr:N \g_CDR_int
1836         }
1837         \hspace*{1ex}
1838       }
1839     }
1840   } {
1841     \tl_set:Nn \l_CDR_info_tl {
1842       {
1843         \CDR:n { format/name }
1844         \CDR:n { format/lineno }
1845         \hspace*{3ex}
1846         \hbox_overlap_left:n {
1847           \int_use:N \g_CDR_int
1848           \int_gincr:N \g_CDR_int
1849         }
1850       }
1851       \hspace*{1ex}
1852     }
1853   }
1854 }
1855 \cs_set:Npn \verbatim@processline {
1856   \CDR_process_record:
1857   \hspace*{\dimexpr \linewidth-\columnwidth}%
1858   \hbox_to_wd:nn { \columnwidth } {
1859     \l_CDR_info_tl
1860     \the\verbatim@line
1861     \color{lightgray}\dotfill
1862   }
1863   \tl_clear:N \l_CDR_name_tl
1864   \par\noindent
1865 }
1866 }
1867 } {
1868   \@bsphack
1869 }
1870 \group_begin:
1871 \g_CDR_hook_tl
1872 \let \do \@makeother
1873 \dospecials \catcode '\^~M \active
1874 \verbatim@start
1875 } {
1876   \int_gsub:Nn \g_CDR_int {
1877     \CDR_int_use:n { \l_CDR_code_name_tl }
1878   }
1879   \int_compare:nNnT { \g_CDR_int } > { 0 } {

```

```

1880 \CDR_clist_map_inline:Nnn \l_CDR_clist {
1881 \CDR_int_gadd:nn { } { \g_CDR_int }
1882 } {
1883 \CDR_int_gadd:nn { ##1 } { \g_CDR_int }
1884 }
1885 \int_gincr:N \g_CDR_code_int
1886 \tl_set:Nx \l_tmpb_tl { \int_use:N \g_CDR_code_int }
1887 \clist_map_inline:Nn \l_CDR_clist {
1888 \seq_gput_right:cV { g/CDR/chunks/##1 } \l_tmpb_tl
1889 }
1890 \prop_gput:NVV \g_CDR_code_prop \l_tmpb_tl \l_CDR_recorded_tl
1891 }
1892 \group_end:
1893 \CDR_if_show_code:T {
1894 }
1895 \CDR_if_show_code:TF {
1896 \CDR_if_use_minted:TF {
1897 \tl_if_empty:NF \l_CDR_recorded_tl {
1898 \exp_args:Nnx \setkeys { FV } {
1899 firstnumber=\CDR_int_use:n { \l_CDR_code_name_tl },
1900 }
1901 \iow_open:Nn \minted@code { \jobname.pyg }
1902 \exp_args:NNV \iow_now:Nn \minted@code \l_CDR_recorded_tl
1903 \iow_close:N \minted@code
1904 \vspace* { \dimexpr -\topsep-\parskip }
1905 \tl_if_empty:NF \l_CDR_info_tl {
1906 \tl_use:N \l_CDR_info_tl
1907 \skip_vertical:n { \dimexpr -\topsep-\parskip-\baselineskip }
1908 \par\noindent
1909 }
1910 \exp_args:Nnx \minted@pygmentize { \jobname.pyg } { \CDR:n { lang } }
1911 %\DeleteFile { \jobname.pyg }
1912 \skip_vertical:n { -\topsep-\partopsep }
1913 }
1914 } {
1915 \exp_args:Nx \skip_vertical:n { \CDR:n { sep } }
1916 \noindent
1917 }
1918 } {
1919 \@esphack
1920 }
1921 }
1922 % =====
1923 % Main options
1924 % =====
1925
1926 \newif\ifCDR@left
1927 \newif\ifCDR@right
1928
1929

```


23.1 options key-value controls

We accept any value because we do not know in advance the real target. There are 2 ways to collect options:

24 Something else

```
1930
1931 % =====
1932 % pygmented commands and environments
1933 % =====
1934
1935
1936 \newcommand\inputpygmented[2][]{%
1937   \begingroup
1938     \CDR@process@options{#1}%
1939     \immediate\write\CDR@outfile{<@@CDR@input@the\CDR@counter}%
1940     \immediate\write\CDR@outfile{\exp_args:NV\detokenize\CDR@global@options,\detokenize{#1}}%
1941     \immediate\write\CDR@outfile{#2}%
1942     \immediate\write\CDR@outfile{>@@CDR@input@the\CDR@counter}%
1943     %
1944     \csname CDR@snippet@the\CDR@counter\endcsname
1945     \global\advance\CDR@counter by 1\relax
1946   \endgroup
1947 }
1948
1949 \cs_generate_variant:Nn \exp_last_unbraced:NnNo { NxNo }
1950
1951 \newcommand\CDR@snippet@run[1]{%
1952   \group_begin:
1953   \typeout{DEBUG~PY~STYLE:< \CDR:n { style } > }
1954   \use_c:n { PYstyle }
1955   \CDR_when:nT { style } {
1956     \use_c:n { PYstyle \CDR:n { style } }
1957   }
1958   \cs_if_exist:CTF {PY} {PYOK} {PYKO}
1959   \CDR:n {font}
1960   \CDR@process@more@options{ \CDR:n {engine} }%
1961   \exp_last_unbraced:NxNo
1962   \use_c: { \CDR:n {engine} } [ \CDR@remainingOptions ]{#1}%
1963   \group_end:
1964 }
1965
1966 % ERROR: JL undefined \CDR@alllinenos
1967
1968 \ProvideDocumentCommand\captionof{mm}{-}{
1969 \def\CDR@alllinenos{(0)}
1970
1971 \def\FormatLineNumber#1{{\rmfamily\tiny#1}}
1972
1973 \newdimen\CDR@leftmargin
1974 \newdimen\CDR@linenosep
1975
```

```

1976 \def\CDR@lineno@do#1{%
1977 \CDR@linenosep Opt%
1978 \use:c { CDR@ \CDR:n {block_engine} @margin }
1979 \exp_args:NNx
1980 \advance \CDR@linenosep { \CDR:n {linenosep} }
1981 \hbox_overlap_left:n {%
1982 \FormatLineNumber{#1}%
1983 \hspace*{\CDR@linenosep}%
1984 }%
1985 }
1986
1987 \newcommand\CDR@tcbbox@more@options{%
1988 nobeforeafter,%
1989 tcbbox~raise~base,%
1990 left=0mm,%
1991 right=0mm,%
1992 top=0mm,%
1993 bottom=0mm,%
1994 boxsep=2pt,%
1995 arc=1pt,%
1996 boxrule=0pt,%
1997 \CDR_options_if_in:nT {colback} {
1998 colback=\CDR:n {colback}
1999 }
2000 }
2001
2002 \newcommand\CDR@mdframed@more@options{%
2003 leftmargin=\CDR@leftmargin,%
2004 frametitle rule=true,%
2005 \CDR_if_in:nT {colback} {
2006 backgroundcolor=\CDR:n {colback}
2007 }
2008 }
2009
2010 \newcommand\CDR@tcolorbox@more@options{%
2011 grow~to~left~by=-\CDR@leftmargin,%
2012 \CDR_if_in:nNT {colback} {
2013 colback=\CDR:n {colback}
2014 }
2015 }
2016
2017 \newcommand\CDR@boite@more@options{%
2018 leftmargin=\CDR@leftmargin,%
2019 \ifcsname CDR@opt@colback\endcsname
2020 colback=\CDR@opt@colback,%
2021 \fi
2022 }
2023
2024 \newcommand\CDR@mdframed@margin{%
2025 \advance \CDR@linenosep \mdflength{outerlinewidth}%
2026 \advance \CDR@linenosep \mdflength{middlelinewidth}%
2027 \advance \CDR@linenosep \mdflength{innerlinewidth}%
2028 \advance \CDR@linenosep \mdflength{innerleftmargin}%
2029 }

```

```

2030
2031 \newcommand\CDR@tcolorbox@margin{%
2032   \advance \CDR@linenosep \kvtcb@left@rule
2033   \advance \CDR@linenosep \kvtcb@leftupper
2034   \advance \CDR@linenosep \kvtcb@boxsep
2035 }
2036
2037 \newcommand\CDR@boite@margin{%
2038   \advance \CDR@linenosep \boite@leftrule
2039   \advance \CDR@linenosep \boite@boxsep
2040 }
2041
2042 \def\CDR@global@options{}
2043
2044 \newcommand\setpygmented[1]{%
2045   \def\CDR@global@options{/CDR.cd,#1}%
2046 }
2047

```

25 Counters

<code>\CDR_int_new:nn</code>	<code>\CDR_int_new:n {<name>} {<value>}</code>
------------------------------	--

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

```

2048 \cs_new:Npn \CDR_int_new:nn #1 #2 {
2049   \int_new:c {g/CDR/int/#1}
2050   \int_gset:cn {g/CDR/int/#1} { #2 }
2051 }

```

<code>\CDR_int_set:nn</code>	<code>\CDR_int_set:n {<name>} {<value>}</code>
------------------------------	--

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

```

2052 \cs_new:Npn \CDR_int_set:nn #1 #2 {
2053   \int_set:cn {g/CDR/int/#1} { #2 }
2054 }
2055 \cs_new:Npn \CDR_int_gset:nn #1 #2 {
2056   \int_gset:cn {g/CDR/int/#1} { #2 }
2057 }

```

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

```

2058 \cs_new:Npn \CDR_int_add:nn #1 #2 {
2059   \int_add:cn {g/CDR/int/#1} { #2 }
2060 }
2061 \cs_new:Npn \CDR_int_gadd:nn #1 #2 {
2062   \int_gadd:cn {g/CDR/int/#1} { #2 }
2063 }

```

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

```

2064 \cs_new:Npn \CDR_int_sub:nn #1 #2 {
2065   \int_sub:cn {g/CDR/int/#1} { #2 }
2066 }
2067 \cs_new:Npn \CDR_int_gsub:nn #1 #2 {
2068   \int_gsub:cn {g/CDR/int/#1} { #2 }
2069 }

```

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

```

2070 \prg_new_conditional:Nnn \CDR_int_if_exist:n { T, F, TF } {
2071   \int_if_exist:cTF {g/CDR/int/#1} {
2072     \prg_return_true:
2073   } {
2074     \prg_return_false:
2075   }
2076 }

```

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

\g/CDR/int/<name>
2077 \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.

```

2078 \cs_new:Npn \CDR_int_use:n #1 {
2079   \int_use:c {g/CDR/int/#1}
2080 }

```

```

2081 \ExplSyntaxOff

```

```

2082 %</sty>

```