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 LuaTeX 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 hilighting.

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

LaTeX identifiers related to coder start with CDR, including both commands and eviron-ment. expl3 identifiers also start with CDR, after and eventual leading c_, l_ or g_. l3keys module path's first component is either CDR or starts with CDR@.

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

The c argument specifier is used here in a more general acception. Normaly , it means that the argument is turned to a command sequence name. Here, it means that the argument is part of something bigger which is turned to a command sequence name. As such, there is no need to explicty expand such an argument.

5   Presentation

coder is a triptych of three complementary components

1. coder.sty, on the LaTeX side,

2. coder-util.lua, to manage some 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 in-stert 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 var-ious parameters, including display engines declared with either \CDRCodeEngineNew or \CDRBlockEngineNew.

5.1   Code flow

The normal code flow is

1. from coder.sty, LaTeX parses a code snippet as \CDRCode argument of CDRBlock environment body, somehow stores it, and calls CDR:hilight_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 LaTeX coloring commands. These are stored in a *.pyg.tex file named after the md5 digest of the original code chunk, a *.pyg.sty LaTeX style file is recorded as well. On return, coder-tool.py gives to coder-util.lua some information, to allow the input of both the *.pyg.sty and the *.pyg.tex file, which are finally executed and the code is displayed with colors. coder-tool.py is also partially responsible of code line numbering in conjunc-tion with coder.sty.

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 LaTeX logic as possible. It receives instructions from coder.sty as command line arguments, LaTeX options, pygments options and fancyvrb options.

5.2 File exports

1. The `\CDRExport` command declares a file path, a list of tags and other usefull 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 LaTeX 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 predefined. Initially `tt`.

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

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

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

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

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

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

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

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

🔴 **commentchar=**⟨*character*⟩ lines starting with this character are ignored. Initially empty.

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

🔴 **frame=none|leftline|topline|bottomline|lines|single** type of frame around the verbatim environment. With `leftline` and `single` modes, a space of a length given by the LATEX `\fboxsep` macro is added between the left vertical line and the text. Initially `none`: no frame.

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

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

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

🔴 **numbersep=**⟨*dimension*⟩ gap between numbers and verbatim lines. Initially 12pt.

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

🔴 **`stepnumber=⟨integer⟩`** interval at which line numbers are printed. Initially 1: all lines are numbered.

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

🔴 **`firstline=⟨integer⟩`** first line to print. Initially empty: all lines from the first are printed.

🔴 **`lastline=⟨integer⟩`** last line to print. Initially empty: all lines until the last one are printed.

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

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

🔴 **`xleftmargin=⟨dimension⟩`** indentation to add at the start of each line. Initially `0pt`: no left margin.

🔴 **`xrightmargin=⟨dimension⟩`** right margin to add after each line. Initially `0pt`: no right margin.

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

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

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

6.2 **pygments** options

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

🔴 **`style=⟨name⟩`** the `pygments` style to use. Initially `default`.

🚫 **`full`** Tells the formatter to output a `full` document, i.e. a complete self-contained document (default: `false`). Forbidden.

🚫 **`title`** If `full` is true, the title that should be used to caption the document (default empty). Forbidden.

🚫 **encoding** If given, must be an encoding name. This will be used to convert the Unicode token strings to byte strings in the output. If it is or None, Unicode strings will be written to the output file, which most file-like objects do not support (default: None).

🚫 **outencoding** Overrides `encoding` if given.

🚫 **docclass** If the `full` option is enabled, this is the document class to use (default: `article`). Forbidden.

🚫 **preamble** If the `full` option is enabled, this can be further preamble commands, e.g. "\usepackage" (default `empty`). Forbidden.

🚫 **linenos[=true|false]** If set to `true`, output line numbers. Initially `false`: no numbering. Ignored in `code` mode.

🚫 **linenostart=⟨*integer*⟩** The line number for the first line. Initially 1: numbering starts from 1. Ignored in `code` mode.

🚫 **linenostep=⟨*integer*⟩** If set to a number n > 1, only every nth 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 LaTeX source text where `<placeholder:number>` should be replaced by a line number and `<placeholder:line>` should be replaced by the hilighted line code provided by `pygments`. They should not include a trailing newline char. The ⟨*type*⟩ is used to describe the line more precisely.

🛑 **Single** It may contain tag related information and number as well. When the block consists of only one line.

🛑 **First** 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** If the first line did not, display the line number, but only when required.

🛑 **Black** for numbered lines,

🛑 **White** 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 `kpsewhich` 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 ⟨***true code***⟩ if ⟨***string***⟩ is the string "true", ⟨***false code***⟩ otherwise.

```
23 local function is_truthy(s)
24   return s == 'true'
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.

8

```
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 ⟨*jobname*⟩.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 ⟨*jobname*⟩.pygd/⟨*jobname*⟩

(*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(⟨*macro name*⟩)

The command named ⟨*macro name*⟩ contains the path to a file. Read the content of that file and print the result to the TEX 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 ⟨*variable*⟩ = safe_equals(⟨*string*⟩)

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

```
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 CDR:load_exec(⟨*lua code chunk*⟩)

Class method. Loads the given ⟨`lua code chunk`⟩ 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`⟩ sring 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 TₑX.

!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 TₑX 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

---
hilight_set

CDR:hilight_set(...)

Hilight 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 hilight_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 hilight_set_var(self, key, var)
145   self:hilight_set(key, assert(token.get_macro(var or 'l_CDR_tl')))
146 end
```

---
hilight_source

CDR:hilight_source(⟨*src*⟩, ⟨*sty*⟩)

Hilight 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. Set the \l_CDR_pyg_sty_tl and \l_CDR_pyg_tex_tl macros on return, depending on ⟨*src*⟩ and ⟨*sty*⟩.

```
147 local function hilight_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     token.set_macro('l_CDR_pyg_sty_tl', pyg_sty_p)
160     texopts.pyg_sty_p = pyg_sty_p
161     local mode,_,__ = lfs.attributes(pyg_sty_p, 'mode')
162     if not mode or not use_cache then
163       use_py = true
164       if debug then
165         print('PYTHON STYLE:')
```

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

```
220    if debug then
221      print('CDR>'..cmd)
222    end
223    local o = io.popen(cmd):read('a')
224    self:load_exec_output(o)
225    if debug then
226      print('PYTHON', o)
227    end
228  end
229  self:cache_record(
230    sty and pyg_sty_p or nil,
231    src and pyg_tex_p or nil
232  )
233 end
```

5.2   Code

hilight_code_setup   CDR:hilight_code_setup()

Hilight 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.

```
234 local function hilight_complete(self, count)
235   token.set_macro('l_CDR_count_tl', count)
236 end
```

5.3   Code

hilight_code_setup   CDR:hilight_code_setup()

Hilight 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.

```
237 local function hilight_code_setup(self)
238   self['.arguments'] = {
239     __cls__ = 'Arguments',
240     source  = '',
241     cache   = true,
242     debug   = false,
243     pygopts = {
244       __cls__ = 'PygOpts',
245       lang    = 'tex',
246       style   = 'default',
247     },
248     texopts = {
249       __cls__ = 'TeXOpts',
250       tags    = '',
251       is_inline = true,
252       pyg_sty_p = '',
253     },
```

```
254      fv_opts = {
255          __cls__ = 'FVOpts',
256      }
257    }
258    self.hilight_json_written = false
259 end
260
```

5.4 Block

<div style="border-top: 1px solid black; width: 100px;"></div>

`hilight_block_setup`   CDR:hilight_block_setup($\langle$*tags clist var*$\rangle$)

Records the contents of the $\langle$*tags clist var*$\rangle$ LATEX variable to prepare block hilighting.

```
261 local function hilight_block_setup(self, tags_clist_var)
262    local tags_clist = assert(token.get_macro(assert(tags_clist_var)))
263    local t = {}
264    for tag in string.gmatch(tags_clist, '([^,]+)') do
265      t[#t+1]=tag
266    end
267    self['.tags clist']  = tags_clist
268    self['.block tags']  = t
269    self['.lines'] = {}
270    self['.arguments'] = {
271      __cls__ = 'Arguments',
272      cache   = false,
273      debug   = false,
274      source  = nil,
275      pygopts = {
276        __cls__ = 'PygOpts',
277        lang = 'tex',
278        style = 'default',
279      },
280      texopts = {
281        __cls__ = 'TeXOpts',
282        tags    = tags_clist,
283        is_inline = false,
284        pyg_sty_p = '',
285      },
286      fv_opts = {
287          __cls__ = 'FVOpts',
288      }
289    }
290    self.hilight_json_written = false
291 end
292
```

`record_line`   CDR:record_line($\langle$*line variable name*$\rangle$)

Store the content of the given named variable.

```
293 local function record_line(self, line_variable_name)
294    local line = assert(token.get_macro(assert(line_variable_name)))
```

```
295    local ll = assert(self['.lines'])
296    ll[#ll+1] = line
297    local lt = self['lines by tag'] or {}
298    self['lines by tag'] = lt
299    for _,tag in ipairs(self['.block tags']) do
300      ll = lt[tag] or {}
301      lt[tag] = ll
302      ll[#ll+1] = line
303    end
304  end
```

---

hilight_advance   CDR:hilight_advance(⟨*count*⟩)

⟨*count*⟩ is the number of line hilighted.

```
305  local function hilight_advance(self, count)
306  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.

```
307  local function export_file(self, file_name)
308    self['.name'] = assert(token.get_macro(assert(file_name)))
309    self['.export'] = {}
310  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.

```
311  local function export_file_info(self, key, value)
312    local export = self['.export']
313    value = assert(token.get_macro(assert(value)))
314    export[key] = value
315  end
```

---

export_complete   CDR:export_complete()

This is called at export time.

```
316 local function export_complete(self)
317   local name    = self['.name']
318   local export  = self['.export']
319   local records = self['.records']
320   local tt = {}
321   local s = export.preamble
322   if s then
323     tt[#tt+1] = s
324   end
325   for _,tag in ipairs(export.tags) do
326     s = records[tag]:concat('\n')
327     tt[#tt+1] = s
328     records[tag] = { [1] = s }
329   end
330   s = export.postamble
331   if s then
332     tt[#tt+1] = s
333   end
334   if #tt>0 then
335     local fh = assert(io.open(name,'w'))
336     fh:write(tt:concat('\n'))
337     fh:close()
338   end
339   self['.file'] = nil
340   self['.exportation'] = nil
341 end
```

7 Caching

We save some computation time by pygmentizing files only when necessary. The coder-tool.py is expected to create a `*.pyg.sty` file for a style and a `*.pyg.tex` file for hilighted code. These files are cached during one whole LaTeX run and possibly between different LaTeX runs. Lua keeps track of both the style files created and hilighted code files created.

cache_clean_all   CDR:cache_clean_all()
cache_record   CDR:cache_record(⟨*style name.pyg.sty*⟩, ⟨*digest.pyg.tex*⟩)
cache_clean_unused   CDR:cache_clean_unused()

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.

```
342 local function cache_clean_all(self)
343   local to_remove = {}
344   for f in lfs.dir(dir_p) do
345     to_remove[f] = true
346   end
347   for k,_ in pairs(to_remove) do
```

```
348        os.remove(dir_p .. k)
349      end
350  end
351  local function cache_record(self, pyg_sty_p, pyg_tex_p)
352    if pyg_sty_p then
353      self['.style_set']  [pyg_sty_p] = true
354    end
355    if pyg_tex_p then
356      self['.colored_set'][pyg_tex_p] = true
357    end
358  end
359  local function cache_clean_unused(self)
360    local to_remove = {}
361    for f in lfs.dir(dir_p) do
362      f = dir_p .. f
363      if not self['.style_set'][f] and not self['.colored_set'][f] then
364        to_remove[f] = true
365      end
366    end
367    for f,_ in pairs(to_remove) do
368      os.remove(f)
369    end
370  end
```

_DESCRIPTION   Short text description of the module.

```
371  local _DESCRIPTION = [[Global coder utilities on the lua side]]
```

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

8   Return the module

```
372  return {
```

Known fields are

```
373    _DESCRIPTION       = _DESCRIPTION,
```

_VERSION to store ⟨*version string*⟩,

```
374    _VERSION           = token.get_macro('fileversion'),
```

date to store ⟨*date string*⟩,

```
375    date               = token.get_macro('filedate'),
```

Various paths ,

```
376    CDR_PY_PATH        = CDR_PY_PATH,
377    PYTHON_PATH        = PYTHON_PATH,
378    set_python_path    = set_python_path,
```

is_truthy

18

```
379   is_truthy          = is_truthy,
```

escape

```
380   escape             = escape,
```

make_directory

```
381   make_directory     = make_directory,
```

load_exec

```
382   load_exec          = load_exec,

383   load_exec_output   = load_exec_output,
```

record_line

```
384   record_line        = record_line,
```

hilight common

```
385   hilight_set        = hilight_set,
386   hilight_set_var    = hilight_set_var,
387   hilight_source     = hilight_source,
388   hilight_advance    = hilight_advance,
389   hilight_complete        = hilight_complete,
```

hilight code

```
390   hilight_code_setup = hilight_code_setup,
```

hilight_block_setup

```
391   hilight_block_setup = hilight_block_setup,
```

cache_clean_all

```
392   cache_clean_all    = cache_clean_all,
```

cache_record

```
393   cache_record       = cache_record,
```

cache_clean_unused

```
394   cache_clean_unused = cache_clean_unused,
```

Internals

```
395   ['.style_set']    = {},
396   ['.colored_set']  = {},
397   ['.options']      = {},
398   ['.export']       = {},
399   ['.name']         = nil,
```

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

```
400   already           = false,
```

Other

```
401   json_p            = json_p,

402 }

403 %</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.

```python
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   def __init__(self, d={}):
27     for k, v in d.items():
28       if type(v) == str:
29         if v.lower() == 'true':
30           setattr(self, k, True)
31           continue
32         elif v.lower() == 'false':
33           setattr(self, k, False)
34           continue
35       setattr(self, k, v)
```

3.1 TeXOpts class

```python
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

```python
40   sty_template=r'''% !TeX root=...
41 \makeatletter
42 \CDR@StyleDefine{<placeholder:style_name>} {%
43   <placeholder:style_defs>}%
44 \makeatother'''
45   line_template =r'''\CDR@Line{<placeholder:type>}{<placeholder:number>}{<placeholder:line>}'''
46   def __init__(self, *args, **kvargs):
47     super().__init__(*args, **kvargs)
48     self.inline_p  = self.ensure_bool(self.is_inline)
49     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.

```python
50 class PygOpts(BaseOpts):
51   style = 'default'
52   nobackground = False
53   linenos = False
```

```
54    linenostart = 1
55    linenostep = 1
56    commandprefix = 'Py'
57    texcomments = False
58    mathescape =  False
59    escapeinside = ""
60    envname = 'Verbatim'
61    lang = 'tex'
62    def __init__(self, *args, **kvargs):
63      super().__init__(*args, **kvargs)
64      self.linenos = self.ensure_bool(self.linenos)
65      self.linenostart = abs(int(self.linenostart))
66      self.linenostep  = abs(int(self.linenostep))
67      self.texcomments = self.ensure_bool(self.texcomments)
68      self.mathescape  = self.ensure_bool(self.mathescape)
```

3.3   FVclass

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

3.4   Argumentsclass

```
100  class Arguments(BaseOpts):
101    cache  = False
```

```
102  debug  = False
103  source = ""
104  style  = "default"
105  json   = ""
106  directory = "."
107  texopts = TeXOpts()
108  pygopts = PygOpts()
109  fv_opts = FVOpts()
```

4   Controller main class

```
110  class Controller:
```

4.1   Static methods

object_hook   Helper for json parsing.

```
111  @staticmethod
112  def object_hook(d):
113    __cls__ = d.get('__cls__', 'Arguments')
114    if __cls__ == 'PygOpts':
115      return PygOpts(d)
116    elif __cls__ == 'FVOpts':
117      return FVOpts(d)
118    elif __cls__ == 'TeXOpts':
119      return TeXOpts(d)
120    else:
121      return Arguments(d)
```

lua_command   self.lua_command(⟨*asynchronous lua command*⟩)
lua_command_now   self.lua_command_now(⟨*synchronous lua command*⟩)
lua_debug

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 ot executed synchronously.

```
122  @staticmethod
123  def lua_command(cmd):
124    print(f'<<<<<*LUA:{cmd}>>>>>')
125  @staticmethod
126  def lua_command_now(cmd):
127    print(f'<<<<<!LUA:{cmd}>>>>>')
128  @staticmethod
129  def lua_debug(msg):
130    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.

```
131   @staticmethod
132   def lua_text_escape(s):
133     k = 0
134     for m in re.findall('=+', s):
135       if len(m) > k: k = len(m)
136     k = (k + 1) * "="
137     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* **??**.)

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

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

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

```
150   @property
151   def parser(self):
152     parser = argparse.ArgumentParser(
153       prog=sys.argv[0],
154       description='''
155 Writes to the output file a set of LaTeX macros describing
156 the syntax hilighting of the input file as given by pygments.
157 '''
158     )
159     parser.add_argument(
160       "-v", "--version",
161       help="Print the version and exit",
162       action='version',
163       version=f'coder-tool version {__version__},'
164       ' (c) {__YEAR__} by Jérôme LAURENS.'
165     )
166     parser.add_argument(
167       "--debug",
168       action='store_true',
169       default=None,
170       help="display informations useful for debugging"
171     )
172     parser.add_argument(
173       "--create_style",
```

```
174        action='store_true',
175        default=None,
176        help="create the style definitions"
177      )
178      parser.add_argument(
179        "--base",
180        action='store',
181        default=None,
182        help="the path of the file to be colored, with no extension"
183      )
184      parser.add_argument(
185        "json",
186        metavar="<json data file>",
187        help="""
188 file name with extension, contains processing information.
189 """
190      )
191      return parser
192
```

4.3   Methods

4.3.1   __init__

__init__   Constructor. Reads the command line arguments.

```
193   def __init__(self, argv = sys.argv):
194     argv = argv[1:] if re.match(".*coder\-tool\.py$", argv[0]) else argv
195     ns = self.parser.parse_args(
196       argv if len(argv) else ['-h']
197     )
198     with open(ns.json, 'r') as f:
199       self.arguments = json.load(
200         f,
201         object_hook = Controller.object_hook
202       )
203     args = self.arguments
204     args.json = ns.json
205     self.texopts = args.texopts
206     pygopts = self.pygopts = args.pygopts
207     fv_opts = self.fv_opts = args.fv_opts
208     self.formatter = LatexFormatter(
209       style = pygopts.style,
210       nobackground = pygopts.nobackground,
211       commandprefix = pygopts.commandprefix,
212       texcomments = pygopts.texcomments,
213       mathescape = pygopts.mathescape,
214       escapeinside = pygopts.escapeinside,
215       envname = 'CDR@Pyg@Verbatim',
216     )
217
218     try:
```

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

4.3.2 `create_style`

`self.create_style()`

Where the ⟨*style*⟩ is created. Does quite nothing if the style is already available.

```
254  def create_style(self):
255    args = self.arguments
256    if not args.create_style:
257      return
258    texopts = args.texopts
259    pyg_sty_p = texopts.pyg_sty_p
260    if args.cache and pyg_sty_p.exists():
261      return
262    texopts = self.texopts
263    style = self.pygopts.style
264    formatter = self.formatter
265    style_defs = formatter.get_style_defs() \
```

```
266        .replace(r'\makeatletter', '') \
267        .replace(r'\makeatother', '') \
268        .replace('\n', '%\n')
269    sty = self.texopts.sty_template.replace(
270        '<placeholder:style_name>',
271        style,
272    ).replace(
273        '<placeholder:style_defs>',
274        style_defs,
275    ).replace(
276        '{}%',
277        '{%}\n}%{'
278    ).replace(
279        '[}%',
280        '[%]\n}%'
281    ).replace(
282        '{]}%',
283        '{%[\n]}%'
284    )
285    with pyg_sty_p.open(mode='w',encoding='utf-8') as f:
286        f.write(sty)
287    if args.debug:
288        print('STYLE', os.path.relpath(pyg_sty_p))
```

4.3.3 pygmentize

---

self.pygmentize

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

Where the $\langle code\rangle$ is hilighted by pygments.

```
289  def pygmentize(self, source):
290    source = hilight(source, self.lexer, self.formatter)
291    m = re.match(
292      r'\\begin{CDR@Pyg@Verbatim}.*?\n(.*?)\n\\end{CDR@Pyg@Verbatim}\s*\Z',
293      source,
294      flags=re.S
295    )
296    assert(m)
297    hilighted = m.group(1)
298    texopts = self.texopts
299    if texopts.is_inline:
300      return hilighted.replace(' ', r'\CDR@Sp '), 0
301    fv_opts = self.fv_opts
302    lines = hilighted.split('\n')
303    ans_code = []
304    try:
305      firstnumber = abs(int(fv_opts.firstnumber))
306    except ValueError:
307      firstnumber = 1
308    number = firstnumber
309    stepnumber = fv_opts.stepnumber
310    numbering = fv_opts.numbers != 'none'
311    def more(type, line):
312      nonlocal number
```

27

```
313      ans_code.append(texopts.line_template.replace(
314          '<placeholder:type>', f'{type}',
315        ).replace(
316          '<placeholder:number>', f'{number}',
317        ).replace(
318          '<placeholder:line>', line,
319      ))
320      number += 1
321    if len(lines) == 1:
322      more('Single', lines.pop(0))
323    elif len(lines):
324      more('First', lines.pop(0))
325      more('Second', lines.pop(0))
326      if stepnumber < 2:
327        def template():
328          return 'Black'
329      elif stepnumber % 5 == 0:
330        def template():
331          return 'Black' if number %\
332            stepnumber == 0 else 'White'
333      else:
334        def template():
335          return 'Black' if (number - firstnumber) %\
336            stepnumber == 0 else 'White'
337
338      for line in lines:
339        more(template(), line)
340
341    hilighted = '\n'.join(ans_code)
342    return hilighted, 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.

```
343  def create_pygmented(self):
344    args = self.arguments
345    base = args.base
346    if not base:
347      return False
348    source = args.source
349    if not source:
350      tex_p = Path(base).with_suffix('.tex')
351      with open(tex_p, 'r') as f:
352        source = f.read()
353    pyg_tex_p = Path(base).with_suffix('.pyg.tex')
354    hilighted, count = self.pygmentize(source)
355    with pyg_tex_p.open(mode='w',encoding='utf-8') as f:
356      f.write(hilighted)
357    if args.debug:
358      print('HILIGHTED', os.path.relpath(pyg_tex_p))
359    self.lua_command_now(f'self:hilight_complete({count})')
```

28

4.4 Main entry

```
360 if __name__ == '__main__':
361   try:
362     ctrl = Controller()
363     x = ctrl.create_style() or ctrl.create_pygmented()
364     print(f'{sys.argv[0]}: done')
365     sys.exit(x)
366   except KeyboardInterrupt:
367     sys.exit(1)
368 %</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 pakage. The latter is a subpath of the former.

```
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 pakage.

```
23 \str_const:Nn \c_CDR_tag_get { CDR@tag@get }
```

(*End definition for* `\c_CDR_tag_get`*. This variable is 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.

```
24 \bool_new:N \l_CDR_bool
```

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

`\l_CDR_tl` Local scratch variable.

```
25 \tl_new:N \l_CDR_tl
```

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

`\l_CDR_str` Local scratch variable.

```
26 \str_new:N \l_CDR_str
```

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

`\l_CDR_seq` Local scratch variable.

```
27 \seq_new:N \l_CDR_seq
```

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

`\l_CDR_prop` Local scratch variable.

```
28 \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.

```
29 \clist_new:N \l_CDR_clist
```

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

5.2 Files

\l_CDR_ior Input file identifier

30 `\ior_new:N \l_CDR_ior`

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

\l_CDR_iow Output file identifier

31 `\iow_new:N \l_CDR_iow`

(*End definition for* \l_CDR_iow*. 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.

32 `\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.

33 `\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.

34 `\tl_new:N \g_CDR_chunks_tl`
35 `\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.

36 `\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.

37 `\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_kv_clist keyval storage.

    38 \clist_new:N \l_CDR_kv_clist

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

\l_CDR_options_tl options storage.

    39 \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.

    40 \tl_new:N \l_CDR_recorded_tl

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

\l_CDR_count_tl Contains the number of lines processed by pygments as tokens.

    41 \tl_new:N \l_CDR_count_tl

(*End definition for* \l_CDR_count_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* **??**.)

5.5 Counters

`\CDR_int_new:cn`

`\CDR_int_new:cn` {⟨*tag name*⟩} {⟨*value*⟩}

Create an integer after ⟨*tag name*⟩ and set it globally to ⟨*value*⟩.

```
47 \cs_new:Npn \CDR_int_new:cn #1 #2 {
48   \int_new:c { g_CDR@int.#1 }
49   \int_gset:cn { g_CDR@int.#1 } { #2 }
50 }
```

`\g_CDR@int.default`
`\g_CDR@int.<tag␣name>`

Generic and named line number counter.

```
51 \CDR_int_new:cn { default } { 1 }
52 \CDR_int_new:cn { @ } { 1 }
```

(*End definition for* `\g_CDR@int.default` *and* `\g_CDR@int.<tag name>`*. These variables are documented on page* **??**.)

`\CDR_int_if_exist_p:c` ⋆
`\CDR_int_if_exist:c`*TF* ⋆

`\CDR_int_if_exist:cTF` {⟨*tag name*⟩} {⟨*true code*⟩} {⟨*false code*⟩}

Execute ⟨*true code*⟩ when an integer named after ⟨*tag name*⟩ exists, ⟨*false code*⟩ otherwise.

```
53 \prg_new_conditional:Nnn \CDR_int_if_exist:c { p, T, F, TF } {
54   \int_if_exist:cTF { g_CDR@int.#1 } {
55     \prg_return_true:
56   } {
57     \prg_return_false:
58   }
59 }
```

`\CDR_int_compare_p:cNn` ⋆
`\CDR_int_compare:cNn`*TF* ⋆

`\CDR_int_compare:cNnTF` {⟨*tag name*⟩} ⟨*operator*⟩ {⟨*intexpr*$_2$⟩} {⟨*true code*⟩} {⟨*false code*⟩}

Forwards to `\int_compare...` with `\CDR_int_use:c { #1 }`.

```
60 \prg_new_conditional:Nnn \CDR_int_compare:cNn { p, T, F, TF } {
61   \int_compare:nNnTF { \CDR_int_use:c { #1 } } #2 { #3 } {
62     \prg_return_true:
63   } {
64     \prg_return_false:
65   }
66 }
```

\CDR_int_set:cn
\CDR_int_gset:cn

\CDR_int_set:cn {⟨*tag name*⟩} {⟨*value*⟩}

Set the integer named after ⟨*tag name*⟩ to the ⟨*value*⟩. \CDR_int_gset:cn makes a global change.

```
67 \cs_new:Npn \CDR_int_set:cn #1 #2 {
68   \int_set:cn { g_CDR@int.#1 } { #2 }
69 }
70 \cs_new:Npn \CDR_int_gset:cn #1 #2 {
71   \int_gset:cn { g_CDR@int.#1 } { #2 }
72 }
```

\CDR_int_set:cc
\CDR_int_gset:cc

\CDR_int_set:cc {⟨*tag name*⟩} {⟨*other tag name*⟩}

Set the integer named after ⟨*tag name*⟩ to the value of the integer named after ⟨*other tag name*⟩. \CDR_int_gset:cc makes a global change.

```
73 \cs_new:Npn \CDR_int_set:cc #1 #2 {
74   \CDR_int_set:cn { #1 } { \CDR_int_use:c { #2 } }
75 }
76 \cs_new:Npn \CDR_int_gset:cc #1 #2 {
77   \CDR_int_gset:cn { #1 } { \CDR_int_use:c { #2 } }
78 }
```

\CDR_int_add:cn
\CDR_int_gadd:cn

\CDR_int_add:cn {⟨*tag name*⟩} {⟨*value*⟩}

Add the ⟨*value*⟩ to the integer named after ⟨*tag name*⟩. \CDR_int_gadd:cn makes a global change.

```
79 \cs_new:Npn \CDR_int_add:cn #1 #2 {
80   \int_add:cn { g_CDR@int.#1 } { #2 }
81 }
82 \cs_new:Npn \CDR_int_gadd:cn #1 #2 {
83   \int_gadd:cn { g_CDR@int.#1 } { #2 }
84 }
```

\CDR_int_add:cc
\CDR_int_gadd:cc

\CDR_int_add:cn {⟨*tag name*⟩} {⟨*other tag name*⟩}

Add to the integer named after ⟨*tag name*⟩ the value of the integer named after ⟨*other tag name*⟩. \CDR_int_gadd:cc makes a global change.

```
85 \cs_new:Npn \CDR_int_add:cc #1 #2 {
86   \CDR_int_add:cn { #1 } { \CDR_int_use:c { #2 } }
87 }
88 \cs_new:Npn \CDR_int_gadd:cc #1 #2 {
89   \CDR_int_gadd:cn { #1 } { \CDR_int_use:c { #2 } }
90 }
```

`\CDR_int_sub:cn`	`\CDR_int_sub:cn {`⟨*tag name*⟩`} {`⟨*value*⟩`}`
`\CDR_int_gsub:cn`	

Substract the ⟨`value`⟩ from the integer named after ⟨`tag name`⟩. `\CDR_int_gsub:n` makes a global change.

```
91 \cs_new:Npn \CDR_int_sub:cn #1 #2 {
92   \int_sub:cn { g_CDR@int.#1 } { #2 }
93 }
94 \cs_new:Npn \CDR_int_gsub:cn #1 #2 {
95   \int_gsub:cn { g_CDR@int.#1 } { #2 }
96 }
```

`\CDR_int_use:c` ⋆	`\CDR_int_use:n {`⟨*tag name*⟩`}`

Use the integer named after ⟨`tag name`⟩.

```
97 \cs_new:Npn \CDR_int_use:c #1 {
98   \int_use:c { g_CDR@int.#1 }
99 }
```

6 Tag properties

The tag properties concern the code chunks. They are set from different paths, 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

`\CDR_tag_get_path:cc` ⋆	`\CDR_tag_get_path:cc {`⟨*tag name*⟩`} {`⟨*relative key path*⟩`}`
`\CDR_tag_get_path:c` ⋆	`\CDR_tag_get_path:c {`⟨*relative key path*⟩`}`

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`.

```
100 \cs_new:Npn \CDR_tag_get_path:cc #1 #2 {
101   \c_CDR_tag_get @ #1 / #2
102 }
103 \cs_new:Npn \CDR_tag_get_path:c {
104   \CDR_tag_get_path:cc { __local }
105 }
```

6.2 Set

`\CDR_tag_set:ccn`	`\CDR_tag_set:ccn {`⟨*tag name*⟩`} {`⟨*relative key path*⟩`} {`⟨*value*⟩`}`
`\CDR_tag_set:ccV`	

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. 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.

```
106 \cs_new_protected:Npn \CDR_tag_set:ccn #1 #2 #3 {
107   \cs_set:cpn { \CDR_tag_get_path:cc { #1 } { #2 } } { \exp_not:n { #3 } }
108 }
109 \cs_new_protected:Npn \CDR_tag_set:ccV #1 #2 #3 {
110   \exp_args:NnnV
111   \CDR_tag_set:ccn { #1 } { #2 } #3
112 }
```

\c_CDR_tag_regex   To parse a l3keys full key path.

```
113 \tl_set:Nn \l_CDR_tl { /([^/]*)/(.*)$ } \use_none:n { $ }
114 \tl_put_left:NV \l_CDR_tl \c_CDR_tag
115 \tl_put_left:Nn \l_CDR_tl { ^ }
116 \exp_args:NNV
117 \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 error is raised on the contrary. This is meant to be called from \keys_define:nn argument. Implementation detail: the last argument is parsed by the last command.

```
118 \cs_new_protected:Npn \CDR_tag_set:n {
119   \exp_args:NnV
120   \regex_extract_once:NnNTF \c_CDR_tag_regex
121     \l_keys_path_str \l_CDR_seq {
122     \CDR_tag_set:ccn
123       { \seq_item:Nn \l_CDR_seq 2 }
124       { \seq_item:Nn \l_CDR_seq 3 }
125   } {
126     \PackageWarning
127       { coder }
128       { Unexpected~key~path~'\l_keys_path_str' }
129     \use_none:n
130   }
131 }
```

\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.

```
132 \cs_new_protected:Npn \CDR_tag_set: {
133   \exp_args:NV
134   \CDR_tag_set:n \l_keys_value_tl
135 }
```

\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.

```
136 \cs_new:Npn \CDR_tag_set:cn #1 {
137   \exp_args:NnV
138   \regex_extract_once:NnNTF \c_CDR_tag_regex
139       \l_keys_path_str \l_CDR_seq {
140     \CDR_tag_set:ccn
141       { \seq_item:Nn \l_CDR_seq 2 }
142       { #1 }
143   } {
144     \PackageWarning
145       { coder }
146       { Unexpected~key~path~'\l_keys_path_str' }
147     \use_none:n
148   }
149 }
```

\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.

```
150 \regex_const:Nn \c_CDR_root_regex { ^(.*)/.*$ } \use_none:n { $ }
151 \cs_new:Npn \CDR_tag_choices: {
152   \exp_args:NVV
153   \str_if_eq:nnT \l_keys_key_tl \l_keys_choice_tl {
154     \exp_args:NnV
155     \regex_extract_once:NnNT \c_CDR_root_regex
156         \l_keys_path_str \l_CDR_seq {
157       \str_set:Nx \l_keys_path_str {
158         \seq_item:Nn \l_CDR_seq 2
159       }
160     }
161   }
162 }
```

\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.

```
163 \cs_new_protected:Npn \CDR_tag_choices_set: {
164   \CDR_tag_choices:
165   \exp_args:NV
166   \CDR_tag_set:n \l_keys_choice_tl
167 }
```

\CDR_if_tag_truthy_p:cc ⋆	\CDR_if_tag_truthy:ccTF {⟨*tag name*⟩} {⟨*relative key path*⟩} {⟨*true code*⟩} {⟨*false code*⟩}
\CDR_if_tag_truthy:cc*TF* ⋆	
\CDR_if_tag_truthy_p:c ⋆	\CDR_if_tag_truthy:cTF {⟨*relative key path*⟩} {⟨*true code*⟩} {⟨*false code*⟩}
\CDR_if_tag_truthy:c*TF* ⋆	

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.

```
168 \prg_new_conditional:Nnn \CDR_if_tag_truthy:cc { p, T,  F, TF } {
169   \exp_args:Ne
170   \str_compare:nNnTF {
171     \exp_args:Ne \str_lowercase:n { \CDR_tag_get:cc { #1 } { #2 } }
172   } = { true } {
173     \prg_return_true:
174   } {
175     \prg_return_false:
176   }
177 }
178 \prg_new_conditional:Nnn \CDR_if_tag_truthy:c { p, T,  F, TF } {
179   \exp_args:Ne
180   \str_compare:nNnTF {
181     \exp_args:Ne \str_lowercase:n { \CDR_tag_get:c { #1 } }
182   } = { true } {
183     \prg_return_true:
184   } {
185     \prg_return_false:
186   }
187 }
```

\CDR_if_truthy_p:n ⋆	\CDR_if_truthy:nTF {⟨*token list*⟩} {⟨*true code*⟩} {⟨*false code*⟩}
\CDR_if_truthy:n*TF* ⋆	

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".

```
188 \prg_new_conditional:Nnn \CDR_if_truthy:n { p, T,  F, TF } {
189   \exp_args:Ne
190   \str_compare:nNnTF { \exp_args:Ne \str_lowercase:n { #1 } } = { true } {
191     \prg_return_true:
192   } {
193     \prg_return_false:
194   }
195 }
```

\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.

```
196 \cs_new_protected:Npn \CDR_tag_boolean_set:n #1 {
197   \CDR_if_truthy:nTF { #1 } {
198     \CDR_tag_set:n { true }
199   } {
200     \CDR_tag_set:n { false }
201   }
202 }
203 \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/`⟨*name*$_1$⟩, ..., `\c_CDR_tag_get/`⟨*name*$_n$⟩ for each tag name of the ordered tags list

2. `\c_CDR_tag_get/default.block`

3. `\c_CDR_tag_get/default`

4. `\c_CDR_tag_get/__pygments`

5. `\c_CDR_tag_get/__pygments.block`

6. `\c_CDR_tag_get/__fancyvrb`

7. `\c_CDR_tag_get/__fancyvrb.block`

8. `\c_CDR_tag_get/__fancyvrb.all` when no using pygments

`\CDR_tag_if_exist_here:cc`*TF* ★   `\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.

```
204 \prg_new_conditional:Nnn \CDR_tag_if_exist_here:cc { T, F, TF } {
205   \cs_if_exist:cTF { \CDR_tag_get_path:cc { #1 } { #2 } } {
206     \prg_return_true:
207   } {
208     \prg_return_false:
209   }
210 }
```

`\CDR_tag_if_exist:ccTF` ⋆	
`\CDR_tag_if_exist:cTF` ⋆	

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

`\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`.

```
211 \prg_new_conditional:Nnn \CDR_tag_if_exist:cc { T, F, TF } {
212   \cs_if_exist:cTF { \CDR_tag_get_path:cc { #1 } { #2 } } {
213     \prg_return_true:
214   } {
215     \seq_if_exist:cTF { \CDR_tag_parent_seq:c { #1 } } {
216       \seq_map_tokens:cn
217         { \CDR_tag_parent_seq:c { #1 } }
218         { \CDR_tag_if_exist_f:cn { #2 } }
219     } {
220       \prg_return_false:
221     }
222   }
223 }
224 \prg_new_conditional:Nnn \CDR_tag_if_exist:c { T, F, TF } {
225   \cs_if_exist:cTF { \CDR_tag_get_path:c { #1 } } {
226     \prg_return_true:
227   } {
228     \seq_if_exist:cTF { \CDR_tag_parent_seq:c { __local } } {
229       \seq_map_tokens:cn
230         { \CDR_tag_parent_seq:c { __local } }
231         { \CDR_tag_if_exist_f:cn { #1 } }
232     } {
233       \prg_return_false:
234     }
235   }
236 }
237 \cs_new:Npn \CDR_tag_if_exist_f:cn #1 #2 {
238   \quark_if_no_value:nTF { #2 } {
239     \seq_map_break:n {
240       \prg_return_false:
241     }
242   } {
243     \CDR_tag_if_exist:ccT { #2 } { #1 } {
244       \seq_map_break:n {
245         \prg_return_true:
246       }
247     }
248   }
249 }
```

`\CDR_tag_get:cc` ⋆	
`\CDR_tag_get:c` ⋆	

`\CDR_tag_get:cc {⟨tag name⟩} {⟨relative key path⟩}`

`\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`.

```
250 \cs_new:Npn \CDR_tag_get:cc #1 #2 {
251   \CDR_tag_if_exist_here:ccTF { #1 } { #2 } {
252     \use:c { \CDR_tag_get_path:cc { #1 } { #2 } }
253   } {
254     \seq_if_exist:cT { \CDR_tag_parent_seq:c { #1 } } {
255       \seq_map_tokens:cn
256         { \CDR_tag_parent_seq:c { #1 } }
257         { \CDR_tag_get_f:cn { #2 } }
258     }
259   }
260 }
261 \cs_new:Npn \CDR_tag_get_f:cn #1 #2 {
262   \quark_if_no_value:nF { #2 } {
263     \CDR_tag_if_exist_here:ccT { #2 } { #1 } {
264       \seq_map_break:n {
265         \use:c { \CDR_tag_get_path:cc { #2 } { #1 } }
266       }
267     }
268   }
269 }
270 \cs_new:Npn \CDR_tag_get:c {
271   \CDR_tag_get:cc { __local }
272 }
```

\CDR_tag_get:ccN
\CDR_tag_get:cN

\CDR_tag_get:ccN {⟨tag name⟩} {⟨relative key path⟩} {⟨tl variable⟩}
\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.

```
273 \cs_new_protected:Npn \CDR_tag_get:ccN #1 #2 #3 {
274   \tl_set:Nf #3 { \CDR_tag_get:cc { #1 } { #2 } }
275 }
276 \cs_new_protected:Npn \CDR_tag_get:cN {
277   \CDR_tag_get:ccN { __local }
278 }
```

\CDR_tag_get:ccN*TF*
\CDR_tag_get:cN*TF*

\CDR_tag_get:ccNTF {⟨tag name⟩} {⟨relative key path⟩} ⟨tl var⟩ {⟨true code⟩}
{⟨false code⟩}
\CDR_tag_get:cNTF {⟨relative key path⟩} ⟨tl var⟩ {⟨true code⟩} {⟨false code⟩}

Getter with branching. If the ⟨relative key path⟩ is knwon, 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.

```
279 \prg_new_protected_conditional:Nnn \CDR_tag_get:ccN { T, F, TF } {
280   \CDR_tag_if_exist:ccTF { #1 } { #2 } {
281     \CDR_tag_get:ccN { #1 } { #2 } #3
282     \prg_return_true:
283   } {
284     \prg_return_false:
285   }
```

```
286 }
287 \prg_new_protected_conditional:Nnn \CDR_tag_get:cN { T, F, TF } {
288   \CDR_tag_if_exist:cTF { #1 } {
289     \CDR_tag_get:cN { #1 } #2
290     \prg_return_true:
291   } {
292     \prg_return_false:
293   }
294 }
```

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.

```
295 \cs_new:Npn \CDR_tag_parent_seq:c #1 {
296   g_CDR:parent.tag @ #1 _seq
297 }
```

`\CDR_tag_inherit:cn`
`\CDR_tag_inherit:(cf|cV)`

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

Set the parents of ⟨*child name*⟩ to the given list.

```
298 \cs_new:Npn \CDR_tag_inherit:cn #1 #2 {
299   \seq_set_from_clist:cn { \CDR_tag_parent_seq:c { #1 } } { #2 }
300   \seq_remove_duplicates:c \l_CDR_tl
301   \seq_remove_all:cn \l_CDR_tl {}
302   \seq_put_right:cn \l_CDR_tl { \q_no_value }
303 }
304 \cs_new:Npn \CDR_tag_inherit:cf {
305   \exp_args:Nnf \CDR_tag_inherit:cn
306 }
307 \cs_new:Npn \CDR_tag_inherit:cV {
308   \exp_args:NnV \CDR_tag_inherit:cn
309 }
```

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.

```
310 \AddToHook { begindocument/before } {
311   \IfFileExists {./\jobname.aux} {} {
312     \lua_now:n {CDR:cache_clean_all()}
313   }
314 }
```

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

```
315 \AddToHook { enddocument/end } {
316   \lua_now:n {CDR:cache_clean_unused()}
317 }
```

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*⟩.

```
318 \cs_new:Npn \CDR_clist_map_inline:Nnn #1 #2 {
319   \clist_if_empty:NTF #1 {
320     #2
321     \use_none:n
322   } {
323     \clist_map_inline:Nn #1
324   }
325 }
```

`\CDR_if_block_p:` ⋆
`\CDR_if_block:TF` ⋆

\CDR_if_block:TF {⟨*true code*⟩} {⟨*false code*⟩}

Execute ⟨*true code*⟩ when inside a code block, ⟨*false code*⟩ when inside an inline code. Raises an error otherwise.

```
326 \prg_new_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
327   \PackageError
328     { coder }
329     { Conditional~not~available }
330 }
```

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

```
331 \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.

```
332 \cs_generate_variant:Nn \keys_define:nn { Vn, xn }
333 \cs_new:Npn \CDR_tag_keys_define:nn #1 {
334   \keys_define:xn { \c_CDR_tag / \exp_not:n { #1 } }
335 }
336 \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.

```
337 \cs_new:Npn \CDR_tag_keys_set:nn #1 {
338   \exp_args:Nx
339   \keys_set:nn { \c_CDR_tag / \exp_not:n { #1 } }
340 }
341 \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*⟩.

```
342 \cs_new:Npn \CDR_keys_set_known:nnN #1 #2 {
343   \keys_set_known:nnnN { #1 } { #2 } { #1 }
344 }
345 \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.

```
346 \cs_new:Npn \CDR_keys_inherit__:nnn #1 #2 #3 {
347   \keys_define:nn { #1 } { #2 .inherit:n = { #3 } }
348 }
349 \cs_new:Npn \CDR_keys_inherit:nnn #1 #2 #3 {
350   \tl_if_empty:nTF { #1 } {
351     \CDR_keys_inherit__:nnn { } { #2 } { #3 }
352   } {
```

```
353    \clist_set:Nn \l_CDR_clist { #3 }
354    \exp_args:Nnnx
355    \CDR_keys_inherit__:nnn { #1 } { #2 } {
356      #1 / \clist_use:Nn \l_CDR_clist { ,#1/ }
357    }
358  }
359 }
360 \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.

```
361 \cs_generate_variant:Nn \keys_set_known:nnnN { VVV, nVx }
362 \cs_new:Npn \CDR_tag_keys_set_known:nnN #1 {
363   \CDR_keys_set_known:xnN { \c_CDR_tag / \exp_not:n { #1 } } }
364 }
365 \cs_generate_variant:Nn \CDR_tag_keys_set_known:nnN { nV }
```

\c_CDR_provide_regex   To parse a l3keys full key path.

```
366 \tl_set:Nn \l_CDR_tl { /([^/]*)(?:/(.*))?$ } \use_none:n { $ }
367 \tl_put_left:NV \l_CDR_tl \c_CDR_tag
368 \tl_put_left:Nn \l_CDR_tl { ^ }
369 \exp_args:NNV
370 \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_kv:n   \CDR_tag_provide_from_kv: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 '/'.

```
371 \regex_const:Nn \c_CDR_engine_regex { ^[^/]*\sengine\soptions$ } \use_none:n { $ }
372 \cs_new:Npn \CDR_tag_provide_from_clist:n #1 {
373   \exp_args:NNx
374   \regex_extract_once:NnNTF \c_CDR_provide_regex {
375     \c_CDR_Tags / #1
376   } \l_CDR_seq {
377     \tl_set:Nx \l_CDR_tl { \seq_item:Nn \l_CDR_seq 3 }
378     \exp_args:Nx
379     \clist_map_inline:nn {
380       \seq_item:Nn \l_CDR_seq 2
381     } {
382       \exp_args:NV
383       \keys_if_exist:nnF \c_CDR_tag { ##1 } {
384         \CDR_keys_inherit:Vnn \c_CDR_tag { ##1 } {
385           __pygments, __pygments.block,
```

```
386          default.block, default.code, default,
387          __fancyvrb, __fancyvrb.block, __fancyvrb.all
388        }
389      \keys_define:Vn \c_CDR_tag {
390        ##1 .code:n = \CDR_tag_keys_set:nn { ##1 } { ####1 },
391        ##1 .value_required:n = true,
392      }
393    }
394    \exp_args:NxV
395    \keys_if_exist:nnF { \c_CDR_tag / ##1 } \l_CDR_tl {
396      \exp_args:NNV
397      \regex_match:NnT \c_CDR_engine_regex
398        \l_CDR_tl {
399      \CDR_tag_keys_define:nx { ##1 } {
400        \l_CDR_tl .code:n = \exp_not:n { \CDR_tag_set:n { ####1 } },
401        \l_CDR_tl .value_required:n = true,
402      }
403      }
404    }
405    }
406  } {
407    \regex_match:NnT \c_CDR_engine_regex { #1 } {
408      \CDR_tag_keys_define:nn { default } {
409        #1 .code:n = \CDR_tag_set:n { ##1 },
410        #1 .value_required:n = true,
411      }
412    }
413  }
414 }
415 \cs_new:Npn \CDR_tag_provide_from_clist:nn #1 #2 {
416   \CDR_tag_provide_from_clist:n { #1 }
417 }
418 \cs_new:Npn \CDR_tag_provide_from_kv:n {
419   \keyval_parse:nnn {
420     \CDR_tag_provide_from_clist:n
421   } {
422     \CDR_tag_provide_from_clist:nn
423   }
424 }
425 \cs_generate_variant:Nn \CDR_tag_provide_from_kv: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 nor used.

9.2.1 Utilities

<table>
<tr><td>\CDR_has_pygments_p: ★</td></tr>
<tr><td>\CDR_has_pygments:<u>TF</u> ★</td></tr>
</table>

\CDR_has_pygments:TF {⟨*true code*⟩} {⟨*false code*⟩}

Execute ⟨***true code***⟩ when pygments is available, ⟨*false code*⟩ otherwise. *Implementation detail*: we define the conditionals and set them afterwards.

```
426 \sys_get_shell:nnN {which~pygmentize} {} \l_CDR_tl
427 \prg_new_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } { }
428 \tl_if_in:NnTF \l_CDR_tl { pygmentize } {
429   \prg_set_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } {
430     \prg_return_true:
431   }
432 } {
433   \prg_set_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } {
434     \prg_return_false:
435   }
436 }
```

9.2.2 `__pygments` l3keys module

```
437 \CDR_tag_keys_define:nn { __pygments } {
```

🔴 `lang=`⟨***language name***⟩ where ⟨***language name***⟩ is recognized by pygments, including a void string,

```
438   lang .code:n = \CDR_tag_set:,
439   lang .value_required:n = true,
```

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

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

🔴 `style=`⟨***style name***⟩ where ⟨***style name***⟩ is recognized by pygments, including a void string,

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

🔴 `commandprefix=`⟨***text***⟩ The LaTeX commands used to produce colored output are constructed using this prefix and some letters. Initially `Py`.

```
443   commandprefix .code:n = \CDR_tag_set:,
444   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`.

```
445   mathescape .code:n = \CDR_tag_boolean_set:x { #1 },
446   mathescape .default:n = true,
```

47

● **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.

```
447  escapeinside .code:n = \CDR_tag_set:,
448  escapeinside .value_required:n = true,
```

● **__initialize** Initializer.

```
449  __initialize .meta:n = {
450    lang = tex,
451    pygments = \CDR_has_pygments:TF { true } { false },
452    style=default,
453    commandprefix=PY,
454    mathescape=false,
455    escapeinside=,
456  },
457  __initialize .value_forbidden:n = true,

458 }
459 \AtBeginDocument{
460  \CDR_tag_keys_set:nn { __pygments } { __initialize }
461 }
```

9.2.3 \c_CDR_tag / __pygments.block **l3keys** module

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

● **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`.

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

● **__initialize** Initializer.

```
465  __initialize .meta:n = {
466    texcomments=false,
467  },
468  __initialize .value_forbidden:n = true,

469 }
470 \AtBeginDocument{
471  \CDR_tag_keys_set:nn { __pygments.block } { __initialize }
472 }
```

9.3 Specifc to **coder**

9.3.1 default **l3keys** module

```
473 \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.

```
474   format .code:n = \CDR_tag_set:,
475   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.

```
476   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.

```
477   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.

```
478   post~processor .code:n = \CDR_tag_set:,
479   post~processor .value_required:n = true,
```

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

```
480   parskip .code:n = \CDR_tag_set:,
481   parskip .value_required:n = true,
```

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

```
482   engine .code:n = \CDR_tag_set:,
483   engine .value_required:n = true,
```

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

```
484   default~engine~options .code:n = \CDR_tag_set:,
485   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.

```
486   __initialize .meta:n = {
487     format = ,
488     cache = true,
489     debug = false,
490     post~processor = ,
```

```
491    parskip = \the\parskip,
492    engine = default,
493    default~engine~options = ,
494  },
495  __initialize .value_forbidden:n = true,

496 }
497 \AtBeginDocument{
498  \CDR_tag_keys_set:nn { default } { __initialize }
499 }
```

9.3.2 `default.code` l3keys module

Void for the moment.

```
500 \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.

```
501  __initialize .meta:n = {
502  },
503  __initialize .value_forbidden:n = true,

504 }
505 \AtBeginDocument{
506  \CDR_tag_keys_set:nn { default.code } { __initialize }
507 }
```

9.3.3 `default.block` l3keys module

```
508 \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`. Set it to false when there happens to be only one tag.

🔴 **`tags=`**⟨*tag name comma list*⟩ to export and display.

```
509  tags .code:n = {
510    \clist_set:Nn \l_CDR_clist { #1 }
511    \clist_remove_duplicates:N \l_CDR_clist
512    \exp_args:NV
513    \CDR_tag_set:n \l_CDR_clist
514  },
515  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.

50

```
516   tags~format .code:n = \CDR_tag_set:,
517   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).

```
518   numbers~format .code:n = \CDR_tag_set:,
519   numbers~format .value_required:n = true,
```

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

```
520   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.

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

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

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

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

```
523   blockskip .code:n = \CDR_tag_set:,
524   blockskip .value_required:n = true,
```

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

```
525   __initialize .meta:n = {
526     tags = ,
527     show~tags = true,
528     only~top = true,
529     use~margin = true,
530     numbers~format = {
531       \sffamily
532       \scriptsize
533       \color{gray}
534     },
535     tags~format = {
536       \bfseries
537     },
538     blockskip = \topsep,
539   },
540   __initialize .value_forbidden:n = true,

541 }
542 \AtBeginDocument{
543   \CDR_tag_keys_set:nn { default.block } { __initialize }
544 }
```

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

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

🛑 **formatcom=**⟨*command*⟩ execute before printing verbatim text. Initially empty.

```
546   formatcom .code:n = \CDR_tag_set:,
547   formatcom .value_required:n = true,
```

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

```
548   fontfamily .code:n = \CDR_tag_set:,
549   fontfamily .value_required:n = true,
```

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

```
550   fontsize .code:n = \CDR_tag_set:,
551   fontsize .value_required:n = true,
```

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

```
552   fontshape .code:n = \CDR_tag_set:,
553   fontshape .value_required:n = true,
```

🛑 **fontseries=**⟨*series name*⟩ LATEX font series to use. Initially auto: the same as the current font.

```
554   fontseries .code:n = \CDR_tag_set:,
555   fontseries .value_required:n = true,
```

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

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

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

```
557   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.

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

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

```
559    tabsize .code:n = \CDR_tag_set:,
560    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.

```
561    defineactive .code:n = \CDR_tag_set:,
562    defineactive .value_required:n = true,
```

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

```
563    reflabel .code:n = \CDR_tag_set:,
564    reflabel .value_required:n = true,
```

✅ **__initialize** Initialization.

```
565    __initialize .meta:n = {
566      formatcom = ,
567      fontfamily = tt,
568      fontsize = auto,
569      fontseries = auto,
570      fontshape = auto,
571      showspaces = false,
572      showtabs = false,
573      obeytabs = false,
574      tabsize = 2,
575      defineactive = ,
576      reflabel = ,
577    },
578    __initialize .value_forbidden:n = true,

579 }
580 \AtBeginDocument{
581   \CDR_tag_keys_set:nn { __fancyvrb } { __initialize }
582 }
```

9.4.2 `__fancyvrb.block` l3keys module

Block specific options, except numbering.

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

🔴 **frame=none|leftline|topline|bottomline|lines|single** type of frame around the ver-batim 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.

```
585   frame .choices:nn =
586     { none, leftline, topline, bottomline, lines, single }
587     { \CDR_tag_choices_set: },
```

🔴 **framerule=**⟨*dimension*⟩ width of the rule of the frame if any. Initially 0.4pt.

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

🔴 **framesep=**⟨*dimension*⟩ width of the gap between the frame (if any) and the text. Initially \fboxsep.

```
590   framesep .code:n = \CDR_tag_set:,
591   framesep .value_required:n = true,
```

🔴 **rulecolor=**⟨*color command*⟩ color of the frame rule, expressed in the standard LaTeX way. Initially black.

```
592   rulecolor .code:n = \CDR_tag_set:,
593   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.

```
594   fillcolor .code:n = \CDR_tag_set:,
595   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.

```
596   label .code:n = \CDR_tag_set:,
597   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.

```
598   labelposition .choices:nn =
599     { none, topline, bottomline, all }
600     { \CDR_tag_choices_set: },
```

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

```
601   baselinestretch .code:n = \CDR_tag_set:,
602   baselinestretch .value_required:n = true,
```

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

🔴 **xleftmargin=**⟨*dimension*⟩ indentation to add at the start of each line. Initially 0pt: no left margin.

```
603   xleftmargin .code:n = \CDR_tag_set:,
604   xleftmargin .value_required:n = true,
```

🔴 **xrightmargin=**⟨*dimension*⟩ right margin to add after each line. Initially 0pt: no right margin.

```
605   xrightmargin .code:n = \CDR_tag_set:,
606   xrightmargin .value_required:n = true,
```

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

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

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

```
608   hfuzz .code:n = \CDR_tag_set:,
609   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.

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

✅ **__initialize** Initialization.

```
611   __initialize .meta:n = {
612     frame = none,
613     label = ,
614     labelposition = none,% auto?
615     baselinestretch = auto,
616     resetmargins = true,
617     xleftmargin = 0pt,
618     xrightmargin = 0pt,
619     hfuzz = 2pt,
620     samepage = false,
621   },
622   __initialize .value_forbidden:n = true,

623 }
624 \AtBeginDocument{
625   \CDR_tag_keys_set:nn { __fancyvrb.block } { __initialize }
626 }
```

9.4.3 `__fancyvrb.number` **l3keys module**

Block line numbering.

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

🔴 **commentchar=**⟨*character*⟩ lines starting with this character are ignored. Initially empty.

```
628    commentchar .code:n = \CDR_tag_set:,
629    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.

```
630    gobble .choices:nn = {
631      0,1,2,3,4,5,6,7,8,9
632    } {
633      \CDR_tag_choices_set:
634    },
```

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

```
635    numbers .choices:nn =
636      { none, left, right }
637      { \CDR_tag_choices_set: },
```

🔴 **numbersep=**⟨*dimension*⟩ gap between numbers and verbatim lines. Initially 12pt.

```
638    numbersep .code:n = \CDR_tag_set:,
639    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.

```
640    firstnumber .code:n = {
641      \regex_match:NnTF \c_CDR_integer_regex { #1 } {
642        \CDR_tag_set:
643      } {
644        \str_case:nnF { #1 } {
645          { auto } { \CDR_tag_set: }
646          { last } { \CDR_tag_set: }
647        } {
648          \PackageWarning
649            { CDR }
650            { Value~'#1'~not~in~auto,~last. }
651        }
652      }
653    },
654    firstnumber .value_required:n = true,
```

🔴 **stepnumber=⟨*integer*⟩** interval at which line numbers are printed. Initially 1: all lines are numbered.

```
655    stepnumber .code:n = \CDR_tag_set:,
656    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.

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

🔴 **firstline=⟨*integer*⟩** first line to print. Initially empty: all lines from the first are printed.

```
658    firstline .code:n = \CDR_tag_set:,
659    firstline .value_required:n = true,
```

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

```
660    lastline .code:n = \CDR_tag_set:,
661    lastline .value_required:n = true,
```

✅ **__initialize** Initialization.

```
662    __initialize .meta:n = {
663      commentchar = ,
664      gobble = 0,
665      numbers = left,
666      numbersep = 1ex,
667      firstnumber = auto,
668      stepnumber = 1,
669      numberblanklines = true,
670      firstline = ,
671      lastline = ,
672    },
673    __initialize .value_forbidden:n = true,

674 }
675 \AtBeginDocument{
676   \CDR_tag_keys_set:nn { __fancyvrb.number } { __initialize }
677 }
```

9.4.4 __fancyvrb.all l3keys module

Options available when pygments is not used.

```
678 \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.

```
679    commandchars .code:n = \CDR_tag_set:,
680    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.

```
681    codes .code:n = \CDR_tag_set:,
682    codes .value_required:n = true,
```

✅ `__initialize` Initialization.

```
683    __initialize .meta:n = {
684      commandchars = ,
685      codes = ,
686    },
687    __initialize .value_forbidden:n = true,

688 }
689 \AtBeginDocument{
690   \CDR_tag_keys_set:nn { __fancyvrb.all } { __initialize }
691 }
```

10 \CDRSet

\CDRSet

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

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

10.1 CDR@Set **l3keys** module

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

🔴 `only description` to typeset only the description section and ignore the implementation section.

```
693    only~description .choices:nn = { false, true, {} } {
694      \int_compare:nNnTF \l_keys_choice_int = 1 {
695        \prg_set_conditional:Nnn \CDR_if_only_description: { p, T, F, TF } { \prg_return_true: }
696      } {
697        \prg_set_conditional:Nnn \CDR_if_only_description: { p, T, F, TF } { \prg_return_false: }
698      }
699    },
700    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.

```
701  python~path .code:n = {
702    \str_set:Nn \l_CDR_str { #1 }
703    \lua_now:n { CDR:set_python_path('l_CDR_str') }
704  },
705 }
```

10.2 Branching

…

\CDR_if_only_description_p: *
\CDR_if_only_description:*TF* *

\CDR_if_only_description:TF {⟨*true code*⟩} {⟨*false code*⟩}

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.

```
706 \exp_args_generate:n { xV, nnV }
707 \cs_new:Npn \CDR_check_unknown:N #1 {
708   \tl_if_empty:NF #1 {
709     \cs_set:Npn \CDR_check_unknown:n ##1 {
710       \PackageWarning
711         { coder }
712         { Unknow~key~'##1' }
713     }
714     \cs_set:Npn \CDR_check_unknown:nn ##1 ##2 {
715       \CDR_check_unknown:n { ##1 }
716     }
717     \exp_args:NnnV
718     \keyval_parse:nnn {
719       \CDR_check_unknown:n
720     } {
721       \CDR_check_unknown:nn
722     } #1
723   }
724 }

725 \NewDocumentCommand \CDRSet { m } {
726   \CDR_keys_set_known:nnN { CDR@Set } { #1 } \l_CDR_kv_clist
727   \clist_map_inline:nn {
728     __pygments, __pygments.block,
729     default.block, default.code, default,
730     __fancyvrb, __fancyvrb.block, __fancyvrb.all
731   } {
732     \CDR_tag_keys_set_known:nVN { ##1 } \l_CDR_kv_clist \l_CDR_kv_clist
733   }
734   \CDR_keys_set_known:VVN \c_CDR_Tags \l_CDR_kv_clist \l_CDR_kv_clist
```

```
735   \CDR_tag_provide_from_kv:V \l_CDR_kv_clist
736   \CDR_keys_set_known:VVN \c_CDR_Tags \l_CDR_kv_clist \l_CDR_kv_clist
737   \CDR_tag_keys_set:nV { default } \l_CDR_kv_clist
738 }
```

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.

```
739 \cs_new:Npn \CDR_export_get_path:cc #1 #2 {
740   CDR @ export @ get @ #1 / #2
741 }
```

\CDR_export_set:ccn
\CDR_export_set:Vcn
\CDR_export_set:VcV

\CDR_export_set:ccn {⟨*file name*⟩} {⟨*relative key path*⟩} {⟨*value*⟩}

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 TEX group level.

```
742 \cs_new_protected:Npn \CDR_export_set:ccn #1 #2 #3 {
743   \cs_set:cpn { \CDR_export_get_path:cc { #1 } { #2 } } { \exp_not:n { #3 } }
744 }
745 \cs_new_protected:Npn \CDR_export_set:Vcn #1 {
746   \exp_args:NV
747   \CDR_export_set:ccn { #1 }
748 }
749 \cs_new_protected:Npn \CDR_export_set:VcV #1 #2 #3 {
750   \exp_args:NVnV
751   \CDR_export_set:ccn #1 { #2 } #3
752 }
```

\CDR_export_if_exist:cc*TF* ⋆

\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.

```
753 \prg_new_conditional:Nnn \CDR_export_if_exist:cc { p, T, F, TF } {
754   \cs_if_exist:cTF { \CDR_export_get_path:cc { #1 } { #2 } } {
755     \prg_return_true:
756   } {
757     \prg_return_false:
758   }
759 }
```

`\CDR_export_get:cc` ⋆	`\CDR_export_get:cc {⟨file name⟩} {⟨relative key path⟩}`

The property value stored for ⟨`file name`⟩ and ⟨`relative key path`⟩.

```
760 \cs_new:Npn \CDR_export_get:cc #1 #2 {
761   \CDR_export_if_exist:ccT { #1 } { #2 } {
762     \use:c { \CDR_export_get_path:cc { #1 } { #2 } }
763   }
764 }
```

`\CDR_export_get:ccN`*TF*	`\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.

```
765 \prg_new_protected_conditional:Nnn \CDR_export_get:ccN { T, F, TF } {
766   \CDR_export_if_exist:ccTF { #1 } { #2 } {
767     \tl_set:Nx #3 { \CDR_export_get:cc { #1 } { #2 } }
768     \prg_return_true:
769   } {
770     \prg_return_false:
771   }
772 }
```

11.2 Storage

`\g_CDR_export_prop`   Global storage for ⟨`file name`⟩=⟨`file export info`⟩

```
773 \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.

```
774 \tl_new:N \l_CDR_file_tl
```

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

`\g_CDR_tags_clist`
`\g_CDR_all_tags_clist`
Store the current list of tags used by `\CDRCode` and the `CDRBlock` environment, or declared by `\CDRExport`. All the tags are recorded, if there is an only one, it is not shown in block code chunks.

```
775 \clist_new:N \g_CDR_tags_clist
776 \clist_new:N \g_CDR_all_tags_clist
```

(*End definition for* `\g_CDR_tags_clist` *and* `\g_CDR_all_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.

```
777 \prop_new:N \l_CDR_export_prop
```

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

61

`\CDR_export_get:cc` ⋆	`\CDR_export_get:cc {⟨file name⟩} {⟨relative key path⟩}`

The property value stored for ⟨`file name`⟩ and ⟨`relative key path`⟩.

```
760 \cs_new:Npn \CDR_export_get:cc #1 #2 {
761   \CDR_export_if_exist:ccT { #1 } { #2 } {
762     \use:c { \CDR_export_get_path:cc { #1 } { #2 } }
763   }
764 }
```

`\CDR_export_get:ccN`*TF*	`\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.

```
765 \prg_new_protected_conditional:Nnn \CDR_export_get:ccN { T, F, TF } {
766   \CDR_export_if_exist:ccTF { #1 } { #2 } {
767     \tl_set:Nx #3 { \CDR_export_get:cc { #1 } { #2 } }
768     \prg_return_true:
769   } {
770     \prg_return_false:
771   }
772 }
```

11.2 Storage

`\g_CDR_export_prop`   Global storage for ⟨`file name`⟩=⟨`file export info`⟩

```
773 \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.

```
774 \tl_new:N \l_CDR_file_tl
```

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

`\g_CDR_tags_clist`
`\g_CDR_all_tags_clist`
Store the current list of tags used by `\CDRCode` and the `CDRBlock` environment, or declared by `\CDRExport`. All the tags are recorded, if there is an only one, it is not shown in block code chunks.

```
775 \clist_new:N \g_CDR_tags_clist
776 \clist_new:N \g_CDR_all_tags_clist
```

(*End definition for* `\g_CDR_tags_clist` *and* `\g_CDR_all_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.

```
777 \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.

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

🔴 `file=`⟨*name*⟩ the output file name, must be provided otherwise an error is raised.

```
779   file .tl_set:N = \l_CDR_file_tl,
780   file .value_required:n = true,
```

🔴 `tags=`⟨*tags comma list*⟩ the list of tags. No exportation when this list is void. Initially empty.

```
781   tags .code:n = {
782     \clist_set:Nn \l_CDR_clist { #1 }
783     \clist_remove_duplicates:N \l_CDR_clist
784     \prop_put:NVV \l_CDR_export_prop \l_keys_key_str \l_CDR_clist
785   },
786   tags .value_required:n = true,
```

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

```
787   lang .code:n = {
788     \prop_put:NVn \l_CDR_export_prop \l_keys_key_str { #1 }
789   },
790   lang .value_required:n = true,
```

🔴 `preamble` the added preamble. Initially empty.

```
791   preamble .code:n = {
792     \prop_put:NVn \l_CDR_export_prop \l_keys_key_str { #1 }
793   },
794   preamble .value_required:n = true,
```

🔴 `postamble` the added postamble. Initially empty.

```
795   postamble .code:n = {
796     \prop_put:NVn \l_CDR_export_prop \l_keys_key_str { #1 }
797   },
798   postamble .value_required:n = true,
```

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

```
799   raw .choices:nn = { false, true, {} } {
800     \prop_put:NVx \l_CDR_export_prop \l_keys_key_str {
801       \int_compare:nNnTF
802         \l_keys_choice_int = 1 { false } { true }
803     }
804   },
```

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

```
805   __initialize .meta:n = {
806     __initialize_prop = #1,
807     file=,
808     tags=,
809     lang=tex,
810     preamble=,
811     postamble=,
812     raw=false,
813   },
814   __initialize .default:n = \l_CDR_export_prop,
```

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

```
815   __initialize_prop .code:n = \prop_clear:N #1,
816   __initialize_prop .value_required:n = true,

817 }
```

11.4   Implementation

```
818 \NewDocumentCommand \CDRExport { m } {
819   \keys_set:nn { CDR@Export } { __initialize }
820   \keys_set:nn { CDR@Export } { #1 }
821   \tl_if_empty:NTF \l_CDR_file_tl {
822     \PackageWarning
823       { coder }
824       { Missing~key~`file' }
825   } {
826     \CDR_export_set:VcV \l_CDR_file_tl { file } \l_CDR_file_tl
827     \prop_map_inline:Nn \l_CDR_export_prop {
828       \CDR_export_set:Vcn \l_CDR_file_tl { ##1 } { ##2 }
829     }
```

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.

```
830     \prop_get:NnNTF \l_CDR_export_prop { tags } \l_CDR_clist {
831       \tl_if_empty:NTF \l_CDR_clist {
832         \PackageWarning
833           { coder }
834           { Missing~key~`tags' }
835       } {
836         \clist_set_eq:NN \g_CDR_tags_clist \l_CDR_clist
837         \clist_put_left:NV \g_CDR_all_tags_clist \l_CDR_clist
838         \clist_remove_duplicates:N \g_CDR_all_tags_clist
839         \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 \g_CDR_tags_clist.

```
840         \exp_args:NV
841         \CDR_export_get:ccNT \l_CDR_file_tl { lang } \l_CDR_tl {
842           \clist_map_inline:Nn \g_CDR_tags_clist {
843             \CDR_tag_set:ccV { ##1 } { lang } \l_CDR_tl
844           }
```

```
845          }
846        }
847      } {
848        \PackageWarning
849          { coder }
850          { Missing~key~`tags' }
851      }
852    }
853 }
```

Files are created at the end of the typesetting process.

```
854 \AddToHook { enddocument / end } {
855   \prop_map_inline:Nn \g_CDR_export_prop {
856     \tl_set:Nn \l_CDR_prop { #2 }
857     \str_set:Nx \l_CDR_str {
858       \prop_item:Nn \l_CDR_prop { file }
859     }
860     \lua_now:n { CDR:export_file('l_CDR_str') }
861     \clist_map_inline:nn {
862       tags, raw, preamble, postamble
863     } {
864       \str_set:Nx \l_CDR_str {
865         \prop_item:Nn \l_CDR_prop { ##1 }
866       }
867       \lua_now:n {
868         CDR:export_file_info('##1','l_CDR_str')
869       }
870     }
871     \lua_now:n { CDR:export_file_complete() }
872   }
873 }
```

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*⟩.

```
874 \cs_set:Npn \CDR@StyleDefine #1 {
875   \tl_gset:cn { g_CDR@Style/#1 }
876 }
```

\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.

```
877 \cs_set:Npn \CDR@StyleUse #1 {
878   \tl_use:c { g_CDR@Style/#1 }
879 }
880 \cs_set:Npn \CDR@StyleUseTag {
881   \CDR@StyleUse { \CDR_tag_get:c { style } }
882 }
```

\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.

```
883 \prg_new_conditional:Nnn \CDR@StyleIfExist:c { TF } {
884   \tl_if_exist:cTF { g_CDR@Style/#1 } {
885     \prg_return_true:
886   } {
887     \prg_return_false:
888   }
889 }
890 \cs_set_eq:NN \CDR@StyleIfExist \CDR@StyleIfExist:cTF
```

13 Creating display engines

13.1 Utilities

\CDR_code_engine:c ★
\CDR_code_engine:V ★
\CDR_block_engine:c ★
\CDR_block_engine:V ★

\CDR_code_engine:c {⟨*engine name*⟩}
\CDR_block_engine:c {⟨*engine name*⟩}

\CDR_code_engine:c builds a command sequence name based on ⟨*engine name*⟩.
\CDR_block_engine:c builds an environment name based on ⟨*engine name*⟩.

```
891 \cs_new:Npn \CDR_code_engine:c #1 {
892   CDR@colored/code/#1:nn
893 }
894 \cs_new:Npn \CDR_block_engine:c #1 {
895   CDR@colored/block/#1
896 }
897 \cs_new:Npn \CDR_code_engine:V {
898   \exp_args:NV \CDR_code_engine:c
899 }
900 \cs_new:Npn \CDR_block_engine:V {
901   \exp_args:NV \CDR_block_engine:c
902 }
```

\l_CDR_engine_tl Storage for an engine name.

```
903 \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.

65

13.2 Implementation

<table>
<tr><td>\CDRCodeEngineNew
\CDRCodeEngineRenew</td><td>\CDRCodeEngineNew {⟨*engine name*⟩}{⟨*engine body*⟩}
\CDRCodeEngineRenew{⟨*engine name*⟩}{⟨*engine body*⟩}</td></tr>
</table>

⟨`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.

```
904 \NewDocumentCommand \CDRCodeEngineNew { mm } {
905   \exp_args:Nx
906   \tl_if_empty:nTF { #1 } {
907     \PackageWarning
908       { coder }
909       { The~engine~cannot~be~void. }
910   } {
911     \cs_new:cpn { \CDR_code_engine:c {#1} } ##1 ##2 {
912       \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
913       #2
914     }
915     \ignorespaces
916   }
917 }

918 \NewDocumentCommand \CDRCodeEngineRenew { mm } {
919   \exp_args:Nx
920   \tl_if_empty:nTF { #1 } {
921     \PackageWarning
922       { coder }
923       { The~engine~cannot~be~void. }
924       \use_none:n
925   } {
926     \cs_if_exist:cTF { \CDR_code_engine:c { #1 } } {
927       \cs_set:cpn { \CDR_code_engine:c { #1 } } ##1 ##2 {
928         \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
929         #2
930       }
931     } {
932       \PackageWarning
933         { coder }
934         { No~code~engine~#1.}
935     }
936     \ignorespaces
937   }
938 }
```

<table>
<tr><td>\CDR@CodeEngineApply</td><td>\CDR@CodeEngineApply {⟨*source*⟩}</td></tr>
</table>

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.

```
939 \cs_new:Npn \CDR@CodeEngineApply #1 {
940   \CDR_tag_get:cN { engine } \l_CDR_engine_tl
941   \CDR_if_code_engine:VF \l_CDR_engine_tl {
942     \PackageError
943       { coder }
944       { \l_CDR_engine_tl\space code~engine~unknown,~replaced~by~'default' }
945       {See~\CDRCodeEngineNew~in~the~coder~manual}
946     \tl_set:Nn \l_CDR_engine_tl { default }
947   }
948   \CDR_tag_get:cN { engine~options } \l_CDR_options_tl
949   \tl_if_empty:NTF \l_CDR_options_tl {
950     \CDR_tag_get:cN { \l_CDR_engine_tl\space engine~options } \l_CDR_options_tl
951   } {
952     \tl_put_left:Nx \l_CDR_options_tl {
953       \CDR_tag_get:c { \l_CDR_engine_tl\space engine~options } ,
954     }
955   }
956   \exp_args:NnV
957   \use:c { \CDR_code_engine:V \l_CDR_engine_tl } \l_CDR_options_tl {
958     \CDR_tag_get:c { format }
959     #1
960   }
961 }
```

\CDRBlockEngineNew	\CDRBlockEngineNew {⟨*engine name*⟩} {⟨*begin instructions*⟩} {⟨*end instructions*⟩}
\CDRBlockEngineRenew	\CDRBlockEngineRenew {⟨*engine name*⟩} {⟨*begin instructions*⟩} {⟨*end instructions*⟩}

Create a LATEX 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.

```
962 \NewDocumentCommand \CDRBlockEngineNew { mm } {
963   \NewDocumentEnvironment { \CDR_block_engine:c { #1 } } { m } {
964     \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
965     #2
966   }
967 }

968 \NewDocumentCommand \CDRBlockEngineRenew { mm } {
969   \tl_if_empty:nTF { #1 } {
970     \PackageWarning
971       { coder }
972       { The~engine~cannot~be~void. }
973       \use_none:n
974   } {
975     \RenewDocumentEnvironment { \CDR_block_engine:c { #1 } } { m } {
976       \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
977       #2
978     }
979   }
980 }
```

13.3 Conditionals

\CDR_if_code_engine:c*TF* ★

\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*⟩.

```
981 \prg_new_conditional:Nnn \CDR_if_code_engine:c { p, T, F, TF } {
982   \cs_if_exist:cTF { \CDR_code_engine:c { #1 } } {
983     \prg_return_true:
984   } {
985     \prg_return_false:
986   }
987 }
988 \prg_new_conditional:Nnn \CDR_if_code_engine:V { p, T, F, TF } {
989   \cs_if_exist:cTF { \CDR_code_engine:V #1 } {
990     \prg_return_true:
991   } {
992     \prg_return_false:
993   }
994 }
```

\CDR_if_block_engine:c*TF* ★

\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*⟩.

```
995  \prg_new_conditional:Nnn \CDR_if_block_engine:c { p, T, F, TF } {
996    \cs_if_exist:cTF { \CDR_block_engine:c { #1 } } {
997      \prg_return_true:
998    } {
999      \prg_return_false:
1000   }
1001 }
1002 \prg_new_conditional:Nnn \CDR_if_block_engine:V { p, T, F, TF } {
1003   \cs_if_exist:cTF { \CDR_block_engine:V #1 } {
1004     \prg_return_true:
1005   } {
1006     \prg_return_false:
1007   }
1008 }
```

13.4 Default code engine

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

```
1009 \CDRCodeEngineNew { default } { #2 }
```

13.5 Default block engine

The default block engine does nothing.

```
1010 \CDRBlockEngineNew { default } { } { }
```

13.6 **efbox** code engine

```
1011 \AtBeginDocument {
1012   \@ifpackageloaded{efbox} {
1013     \CDRCodeEngineNew {efbox} {
1014       \efbox[#1]{#2}%
1015     }
1016   }
1017 }
```

13.7 **Block mode default engine**

```
1018 \CDRBlockEngineNew {} {
1019 } {
1020 }
```

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.

```
1021 \cs_new:Npn \CDR@DefineSp {
1022   \CDR_if_tag_truthy:cTF { showspaces } {
1023     \cs_set:Npn \CDR@Sp {{\FancyVerbSpace}}
1024   } {
1025     \cs_set_eq:NN \CDR@Sp \space
1026   }
1027 }
```

\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.

```
1028 \tl_new:N \l_CDR_tag_tl
```

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

69

14.3 `__code` l3keys module

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

```
1029 \CDR_tag_keys_define:nn { __code } {
```

✅ **tag=**⟨**name**⟩ to use the settings of the already existing named tag to display.

```
1030   tag .tl_set:N = \l_CDR_tag_tl,
1031   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.

```
1032   engine~options .code:n = \CDR_tag_set:,
1033   engine~options .value_required:n = true,
```

🔴 **__initialize** initialize

```
1034   __initialize .meta:n = {
1035     tag = default,
1036     engine~options = ,
1037   },
1038   __initialize .value_forbidden:n = true,
```

```
1039 }
```

14.4 Implementation

\CDR_code_format: \CDR_code_format:

Private utility to setup the formatting.

```
1040 \cs_new:Npn \CDR_brace_if_contains_comma:n #1 {
1041   \tl_if_in:nnTF { #1 } { , } { { #1 } } { #1 }
1042 }
1043 \cs_generate_variant:Nn \CDR_brace_if_contains_comma:n { V }
1044 \cs_new:Npn \CDR_code_format: {
1045   \frenchspacing
1046   \CDR_tag_get:cN { baselinestretch } \l_CDR_tl
1047   \tl_if_eq:NnF \l_CDR_tl { auto } {
1048     \exp_args:NNV
1049     \def \baselinestretch \l_CDR_tl
1050   }
1051   \CDR_tag_get:cN { fontfamily } \l_CDR_tl
1052   \tl_if_eq:NnT \l_CDR_tl { tt } { \tl_set:Nn \l_CDR_tl { lmtt } }
1053   \exp_args:NV
1054   \fontfamily \l_CDR_tl
1055   \clist_map_inline:nn { series, shape } {
1056     \CDR_tag_get:cN { font##1 } \l_CDR_tl
1057     \tl_if_eq:NnF \l_CDR_tl { auto } {
1058       \exp_args:NnV
1059       \use:c { font##1 } \l_CDR_tl
1060     }
```

```
1061    }
1062    \CDR_tag_get:cN { fontsize } \l_CDR_tl
1063    \tl_if_eq:NnF \l_CDR_tl { auto } {
1064      \tl_use:N \l_CDR_tl
1065    }
1066    \selectfont
1067 %  \@noligs ?? this is in fancyvrb but does not work here as is
1068 }
```

\CDR_code:n   \CDR_code:n ⟨delimiter⟩

Main utility used by \CDRCode.

```
1069 \cs_new:Npn \CDR_code:n #1 {
1070    \CDR_if_tag_truthy:cTF {pygments} {
1071      \cs_set:Npn \CDR@StyleUseTag {
1072        \CDR@StyleUse { \CDR_tag_get:c { style } }
1073        \cs_set_eq:NN \CDR@StyleUseTag \prg_do_nothing:
1074      }
1075      \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1076        __fancyvrb,
1077      }
1078      \CDR_tag_keys_set:nV { __local } \l_CDR_kv_clist
1079      \DefineShortVerb { #1 }
1080      \SaveVerb [
1081        aftersave = {
1082          \exp_args:Nx \UndefineShortVerb { #1 }
1083          \lua_now:n { CDR:hilight_code_setup() }
1084          \CDR_tag_get:cN {lang} \l_CDR_tl
1085          \lua_now:n { CDR:hilight_set_var('lang') }
1086          \CDR_tag_get:cN {cache} \l_CDR_tl
1087          \lua_now:n { CDR:hilight_set_var('cache') }
1088          \CDR_tag_get:cN {debug} \l_CDR_tl
1089          \lua_now:n { CDR:hilight_set_var('debug') }
1090          \CDR_tag_get:cN {style} \l_CDR_tl
1091          \lua_now:n { CDR:hilight_set_var('style') }
1092          \lua_now:n { CDR:hilight_set_var('source', 'FV@SV@CDR@Source') }
1093          \FV@UseKeyValues
1094          \frenchspacing
1095          %  \FV@SetupFont Break
1096          \FV@DefineWhiteSpace
1097          \FancyVerbDefineActive
1098          \FancyVerbFormatCom
1099          \CDR_code_format:
1100          \CDR@DefineSp
1101          \CDR_tag_get:c { format }
1102          \CDR@DefineSp
1103          \CDR@CodeEngineApply {
1104            \CDR@StyleIfExist { \l_CDR_tl } {
1105              \CDR@StyleUseTag
1106              \lua_now:n { CDR:hilight_source(false, true) }
1107            } {
1108              \lua_now:n { CDR:hilight_source(true, true) }
1109              \input { \l_CDR_pyg_sty_tl }
```

```
1110            \CDR@StyleUseTag
1111          }
1112          \makeatletter
1113          \input { \l_CDR_pyg_tex_tl }
1114          \makeatother
1115        }
1116        \group_end:
1117      }
1118    ] { CDR@Source } #1
1119  } {
1120    \exp_args:NV \fvset \l_CDR_kv_clist
1121    \DefineShortVerb { #1 }
1122    \SaveVerb [
1123      aftersave = {
1124        \UndefineShortVerb { #1 }
1125        \cs_set_eq:NN \CDR@FormattingPrep \FV@FormattingPrep
1126        \cs_set:Npn \FV@FormattingPrep {
1127          \CDR@FormattingPrep
1128          \CDR_tag_get:c { format }
1129        }
1130        \CDR@CodeEngineApply { \mbox {
1131          \FV@UseKeyValues
1132          \FV@FormattingPrep
1133          \FV@SV@CDR@Code
1134        } }
1135        \group_end:
1136      }
1137    ] { CDR@Code } #1
1138  }
1139 }
1140 \NewDocumentCommand \CDRCode { O{} } {
1141  \group_begin:
1142  \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
1143    \prg_return_false:
1144  }
1145  \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1146    __code, default.code, __pygments, default,
1147  }
1148  \CDR_tag_keys_set_known:nnN { __local } { #1 } \l_CDR_kv_clist
1149  \CDR_tag_provide_from_kv:V \l_CDR_kv_clist
1150  \CDR_tag_keys_set_known:nVN { __local } \l_CDR_kv_clist \l_CDR_kv_clist
1151  \exp_args:NNV
1152  \def \FV@KeyValues \l_CDR_kv_clist
1153  \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1154    __fancyvrb,
1155  }
1156  \CDR_tag_keys_set:nV { __local } \l_CDR_kv_clist
1157  \CDR_tag_inherit:cf { __local } {
1158    \tl_if_empty:NF \l_CDR_tag_tl { \l_CDR_tag_tl, }
1159    __code, default.code, __pygments, default, __fancyvrb,
1160  }
1161  \CDR_code:n
1162 }
1163 \cs_set:Npn \CDR_code:n #1 {
```

```
1164    \CDR_if_tag_truthy:cTF {pygments} {
1165      \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1166        __fancyvrb,
1167      }
1168      \CDR_tag_keys_set:nV { __local } \l_CDR_kv_clist
1169      \DefineShortVerb { #1 }
1170      \SaveVerb [
1171        aftersave = {
1172          \exp_args:Nx \UndefineShortVerb { #1 }
1173          \lua_now:n { CDR:hilight_code_setup() }
1174          \CDR_tag_get:cN {lang} \l_CDR_tl
1175          \lua_now:n { CDR:hilight_set_var('lang') }
1176          \CDR_tag_get:cN {cache} \l_CDR_tl
1177          \lua_now:n { CDR:hilight_set_var('cache') }
1178          \CDR_tag_get:cN {debug} \l_CDR_tl
1179          \lua_now:n { CDR:hilight_set_var('debug') }
1180          \CDR_tag_get:cN {style} \l_CDR_tl
1181          \lua_now:n { CDR:hilight_set_var('style') }
1182          \lua_now:n { CDR:hilight_set_var('source', 'FV@SV@CDR@Source') }
1183          \exp_args:NNV
1184          \def \FV@KeyValues \l_CDR_kv_clist
1185          \FV@UseKeyValues
1186          \frenchspacing
1187          % \FV@SetupFont Break
1188          \FV@DefineWhiteSpace
1189          \FancyVerbDefineActive
1190          \FancyVerbFormatCom
1191          \CDR@DefineSp
1192          \CDR_code_format:
1193          \CDR_tag_get:c { format }
1194          \CDR@CodeEngineApply {
1195            \CDR@StyleIfExist { \CDR_tag_get:c {style} } {
1196              \CDR@StyleUseTag
1197              \lua_now:n { CDR:hilight_source(false, true) }
1198            } {
1199              \lua_now:n { CDR:hilight_source(true, true) }
1200              \input { \l_CDR_pyg_sty_tl }
1201              \CDR@StyleUseTag
1202            }
1203            \makeatletter
1204            \input { \l_CDR_pyg_tex_tl }
1205            \makeatother
1206          }
1207          \group_end:
1208        }
1209      ] { CDR@Source } #1
1210    } {
1211      \DefineShortVerb { #1 }
1212      \SaveVerb [
1213        aftersave = {
1214          \UndefineShortVerb { #1 }
1215          \cs_set_eq:NN \CDR@FormattingPrep \FV@FormattingPrep
1216          \cs_set:Npn \FV@FormattingPrep {
1217            \CDR@FormattingPrep
```

```
1218          \CDR_tag_get:c { format }
1219        }
1220      \CDR@CodeEngineApply { A \mbox { a
1221        \exp_args:NNV
1222        \def \FV@KeyValues \l_CDR_kv_clist
1223        \FV@UseKeyValues
1224        \FV@FormattingPrep
1225        \@nameuse{FV@SV@CDR@Code}
1226      z } Z }
1227        \group_end:
1228      }
1229    ] { CDR@Code } #1
1230  }
1231 }

1232 \RenewDocumentCommand \CDRCode { O{} } {
1233  \group_begin:
1234  \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
1235    \prg_return_false:
1236  }
1237  \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1238    __code, default.code, __pygments, default,
1239  }
1240  \CDR_tag_keys_set_known:nnN { __local } { #1 } \l_CDR_kv_clist
1241  \CDR_tag_provide_from_kv:V \l_CDR_kv_clist
1242  \CDR_tag_keys_set_known:nVN { __local } \l_CDR_kv_clist \l_CDR_kv_clist
1243  \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1244    __fancyvrb,
1245  }
1246  \CDR_tag_keys_set:nV { __local } \l_CDR_kv_clist
1247  \CDR_tag_inherit:cf { __local } {
1248    \tl_if_empty:NF \l_CDR_tag_tl { \l_CDR_tag_tl, }
1249    __code, default.code, __pygments, default, __fancyvrb,
1250  }
1251  \fvset{showspaces}
1252  \CDR_code:n
1253 }
```

15   CDRBlock environment

\begin{CDRBlock}{⟨key[=value] list⟩} ... \end{CDRBlock}

15.1   Storage

\l_CDR_block_prop

```
1254 \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.

```
1255 \CDR_tag_keys_define:nn { __block } {
```

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

```
1256   no~export .code:n = \CDR_tag_boolean_set:x { #1 },
1257   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.

```
1258   no~export~format .code:n = \CDR_tag_set:,
1259   no~export~format .value_required:n = true,
```

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

```
1260   test .code:n = \CDR_tag_boolean_set:x { #1 },
1261   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.

```
1262   engine~options .code:n = \CDR_tag_set:,
1263   engine~options .value_required:n = true,
```

🔴 **__initialize** initialize

```
1264   __initialize .meta:n = {
1265     no~export = false,
1266     no~export~format = ,
1267     test = false,
1268     engine~options = ,
1269   },
1270   __initialize .value_forbidden:n = true,

1271 }
```

15.3 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.

```
1272 \clist_map_inline:nn { i, ii, iii, iv } {
1273   \cs_set_eq:cc { CDR@ListProcessLine@ #1 } { FV@ListProcessLine@ #1 }
1274 }
1275 \cs_new:Npn \CDR_process_line:n #1 {
1276   \str_set:Nn \l_CDR_str { #1 }
1277   \lua_now:n {CDR:record_line('l_CDR_str')}
1278 }

1279 \def\FVB@CDRBlock {
1280   \@bsphack
1281   \group_begin:
1282   \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
1283     \prg_return_true:
1284   }
1285   \CDR_tag_keys_set:nn { __block } { __initialize }
```

Reading the options: we absorb the options available in `\FV@KeyValues`, first for l3keys modules, then for `\fvset`.

```
1286   \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1287     __block, __pygments.block, default.block,
1288     __pygments, default,
1289   }
1290   \CDR_tag_keys_set_known:nVN { __local } \FV@KeyValues \l_CDR_kv_clist
1291   \CDR_tag_provide_from_kv:V \l_CDR_kv_clist
1292   \CDR_tag_keys_set_known:nVN { __local } \l_CDR_kv_clist \l_CDR_kv_clist
```

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`. This can be overwritten with the `tags=...` user interface. At least one tag must be provided.

```
1293   \CDR_tag_inherit:cn { __local } { default.block }
1294   \CDR_tag_get:cN { tags } \l_CDR_clist
1295   \clist_if_empty:NTF \l_CDR_clist {
1296     \clist_if_empty:NT \g_CDR_tags_clist {
1297       \PackageWarning
1298         { coder }
1299         { No~(default)~tags~provided. }
1300     }
1301   } {
1302     \clist_gset_eq:NN \g_CDR_tags_clist \l_CDR_clist
1303   }
1304   \lua_now:n {
1305     CDR:hilight_block_setup('g_CDR_tags_clist')
1306   }
```

`\l_CDR_pygments_bool` is `true` iff one of the tags needs `pygments` or there is no tag and `pygments=true` was given.

```
1307   \bool_set_false:N \l_CDR_pygments_bool
1308   \clist_if_empty:NTF \g_CDR_tags_clist {
1309     \bool_set:Nn \l_CDR_pygments_bool {
1310       \CDR_if_tag_truthy_p:c { pygments }
1311     }
1312   } {
1313     \bool_if:NF \l_CDR_pygments_bool {
1314       \clist_map_inline:Nn \g_CDR_tags_clist {
1315         \CDR_if_tag_truthy:ccT { ##1 } { pygments } {
1316           \clist_map_break:n {
1317             \bool_set_true:N \l_CDR_pygments_bool
1318           }
1319         }
1320       }
1321     }
1322   }
```

Now we setup the full inheritance tree.

```
1323   \CDR_tag_inherit:cf { __local } {
1324     \g_CDR_tags_clist,
1325     __block, default.block, __pygments.block, __fancyvrb.block, __fancyvrb.number,
1326     __pygments, default, __fancyvrb,
```

```
1327    }
1328    \bool_if:NTF \l_CDR_pygments_bool {
1329      \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1330        __fancyvrb.number
1331      }
1332      \CDR_tag_keys_set_known:nVN { __local } \l_CDR_kv_clist \l_CDR_kv_clist
1333      \exp_args:NV \fvset \l_CDR_kv_clist
1334      \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1335        __fancyvrb, __fancyvrb.block
1336      }
1337      \exp_args:NnV
1338      \CDR_tag_keys_set:nn { __local } \l_CDR_kv_clist
1339      \exp_args:NNV
1340      \def \FV@KeyValues \l_CDR_kv_clist
```

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

```
1341      \CDR_tag_get:cN {lang} \l_CDR_tl
1342      \lua_now:n { CDR:hilight_set_var('lang') }
1343      \CDR_tag_get:cN {cache} \l_CDR_tl
1344      \lua_now:n { CDR:hilight_set_var('cache') }
1345      \CDR_tag_get:cN {debug} \l_CDR_tl
1346      \lua_now:n { CDR:hilight_set_var('debug') }
1347      \CDR_tag_get:cN {style} \l_CDR_tl
1348      \lua_now:n { CDR:hilight_set_var('style') }
1349      \CDR@StyleIfExist { \l_CDR_tl } { } {
1350        \lua_now:n { CDR:hilight_source(true, false) }
1351        \input { \l_CDR_pyg_sty_tl }
1352      }
1353      \CDR@StyleUseTag
1354      \CDR_if_tag_truthy:cTF {no~export} {
1355        \clist_map_inline:nn { i, ii, iii, iv } {
1356          \cs_set:cpn { FV@ListProcessLine@ ##1 } ####1 {
1357            \tl_set:Nn \l_CDR_tl { ####1 }
1358            \lua_now:n { CDR:record_line('l_CDR_tl') }
1359          }
1360        }
1361      } {
1362        \clist_map_inline:nn { i, ii, iii, iv } {
1363          \cs_set:cpn { FV@ListProcessLine@ ##1 } ####1 {
1364            \tl_set:Nn \l_CDR_tl { ####1 }
1365            \lua_now:n { CDR:record_line('l_CDR_tl') }
1366          }
1367        }
1368      }
1369      \CDR_tag_get:cN { engine } \l_CDR_engine_tl
1370      \CDR_if_code_engine:VF \l_CDR_engine_tl {
1371        \PackageError
1372          { coder }
1373          { \l_CDR_engine_tl\space block~engine~unknown,~replaced~by~'default' }
1374          {See~\CDRBlockEngineNew~in~the~coder~manual}
1375        \tl_set:Nn \l_CDR_engine_tl { default }
1376      }
1377      \CDR_tag_get:cN { \l_CDR_engine_tl~engine~options } \l_CDR_options_tl
1378      \exp_args:NnV
```

```
1379        \use:c { \CDR_block_engine:V \l_CDR_engine_tl } \l_CDR_options_tl
1380
1381        \def\FV@ProcessLine ##1 {
1382          \tl_set:Nn \l_CDR_tl { ##1 }
1383          \lua_now:n { CDR:record_line('l_CDR_tl') }
1384        }
1385      } {
1386        \exp_args:NNV
1387        \def \FV@KeyValues \l_CDR_kv_clist
1388        \CDR_if_tag_truthy:cF {no~export} {
1389          \clist_map_inline:nn { i, ii, iii, iv } {
1390            \cs_set:cpn { FV@ListProcessLine@ ##1 } ####1 {
1391              \tl_set:Nn \l_CDR_tl { ####1 }
1392              \lua_now:n { CDR:record_line('l_CDR_tl') }
1393              \use:c { CDR@ListProcessLine@ ##1 } { ####1 }
1394            }
1395          }
1396        }
1397        \exp_args:NnV
1398        \use:c { \CDR_block_engine:V \l_CDR_engine_tl } \l_CDR_options_tl
1399        \FV@VerbatimBegin
1400      }
1401      \FV@Scan
1402 }
1403 \def\FVE@CDRBlock {
1404    \bool_if:NT \l_CDR_pygments_bool {
1405      \CDR_tag_get:c { format }
1406      \fvset{ commandchars=\\\{\} }
1407      \CDR@DefineSp
1408      \FV@VerbatimBegin
1409      \lua_now:n { CDR:hilight_source(false, true) }
1410      \makeatletter
1411      \input{ \l_CDR_pyg_tex_tl }
1412      \makeatother
1413    }
1414    \FV@VerbatimEnd
1415    \use:c { end \CDR_block_engine:V \l_CDR_engine_tl }
1416    \group_end:
1417    \@esphack
1418 }
1419 \DefineVerbatimEnvironment{CDRBlock}{CDRBlock}{}
1420
```

16 Management

\g_CDR_in_impl_bool   Whether we are currently in the implementation section.

```
1421 \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 {⟨*true code*⟩} {⟨*false code*⟩}

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

```
1422 \prg_new_conditional:Nnn \CDR_if_show_code: { T, F, TF } {
1423   \bool_if:nTF {
1424     \g_CDR_in_impl_bool && !\g_CDR_with_impl_bool
1425   } {
1426     \prg_return_false:
1427   } {
1428     \prg_return_true:
1429   }
1430 }
```

\g_CDR_with_impl_bool

```
1431 \bool_new:N \g_CDR_with_impl_bool
```

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

\CDRPreamble {⟨*variable*⟩} {⟨*file name*⟩}

Store the content of ⟨*file name*⟩ into the variable ⟨*variable*⟩.

```
1432 \DeclareDocumentCommand \CDRPreamble { m m } {
1433   \msg_info:nnn
1434     { coder }
1435     { :n }
1436     { Reading~preamble~from~file~"#2". }
1437   \group_begin:
1438   \tl_set:Nn \l_tmpa_tl { #2 }
1439   \exp_args:NNNx
1440   \group_end:
1441   \tl_set:Nx #1 { \lua_now:n {CDR.print_file_content('l_tmpa_tl')} }
1442 }
```

17   Section separators

\CDRImplementation
\CDRFinale

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

18   Finale

```
1443 \newcounter{CDR@impl@page}
1444 \DeclareDocumentCommand \CDRImplementation {} {
1445   \bool_if:NF \g_CDR_with_impl_bool {
1446     \clearpage
1447     \bool_gset_true:N \g_CDR_in_impl_bool
1448     \let\CDR@old@part\part
```

79

```
1449      \DeclareDocumentCommand\part{som}{}
1450      \let\CDR@old@section\section
1451      \DeclareDocumentCommand\section{som}{}
1452      \let\CDR@old@subsection\subsection
1453      \DeclareDocumentCommand\subsection{som}{}
1454      \let\CDR@old@subsubsection\subsubsection
1455      \DeclareDocumentCommand\subsubsection{som}{}
1456      \let\CDR@old@paragraph\paragraph
1457      \DeclareDocumentCommand\paragraph{som}{}
1458      \let\CDR@old@subparagraph\subparagraph
1459      \DeclareDocumentCommand\subparagraph{som}{}
1460      \cs_if_exist:NT \refsection{ \refsection }
1461      \setcounter{ CDR@impl@page }{ \value{page} }
1462    }
1463 }
1464 \DeclareDocumentCommand\CDRFinale {} {
1465   \bool_if:NF \g_CDR_with_impl_bool {
1466     \clearpage
1467     \bool_gset_false:N \g_CDR_in_impl_bool
1468     \let\part\CDR@old@part
1469     \let\section\CDR@old@section
1470     \let\subsection\CDR@old@subsection
1471     \let\subsubsection\CDR@old@subsubsection
1472     \let\paragraph\CDR@old@paragraph
1473     \let\subparagraph\CDR@old@subparagraph
1474     \setcounter { page } { \value{ CDR@impl@page } }
1475   }
1476 }
1477 \cs_set_eq:NN \CDR_line_number: \prg_do_nothing:
```

19   Finale

```
1478 %\AddToHook { cmd/FancyVerbFormatLine/before } {
1479 %  \CDR_line_number:
1480 %}
1481 \AddToHook { shipout/before } {
1482   \tl_gclear:N \g_CDR_chunks_tl
1483 }

1484 % ========================================================
1485 % Auxiliary:
1486 %   finding the widest string in a comma
1487 %   separated list of strings delimited by parenthesis
1488 % ========================================================
1489
1490 % arguments:
1491 % #1) text: a comma separeted list of strings
1492 % #2) formatter: a macro to format each string
1493 % #3) dimension: will hold the result
1494
1495 \cs_new:Npn \CDRWidest (#1) #2 #3 {
1496   \group_begin:
1497   \dim_set:Nn #3 { 0pt }
```

```
1498    \clist_map_inline:nn { #1 } {
1499      \hbox_set:Nn \l_tmpa_box { #2{##1} }
1500      \dim_set:Nn \l_tmpa_dim { \dim_eval:n { \box_wd:N \l_tmpa_box } }
1501      \dim_compare:nNnT { #3 } < { \l_tmpa_dim } {
1502        \dim_set_eq:NN #3 \l_tm pa_dim
1503      }
1504    }
1505    \exp_args:NNNV
1506    \group_end:
1507    \dim_set:Nn #3 #3
1508 }
1509 \ExplSyntaxOff
1510
```

20 **pygmentex** implementation

```
1511 % ===========================================================
1512 % fancyvrb new commands to append to a file
1513 % ===========================================================
1514
1515 % See http://tex.stackexchange.com/questions/47462/inputenc-error-with-unicode-chars-and-verbati
1516
1517 \ExplSyntaxOn
1518
1519 \seq_new:N \l_CDR_records_seq
1520
1521 % ===========================================================
1522 % Main options
1523 % ===========================================================
1524
1525 \newif\ifCDR@left
1526 \newif\ifCDR@right
1527
1528
```

20.1 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:

21 Something else

```
1529
1530 % ===========================================================
1531 % pygmented commands and environments
1532 % ===========================================================
1533
1534
1535 \cs_generate_variant:Nn \exp_last_unbraced:NnNo { NxNo }
1536
1537
1538 % ERROR: JL undefined \CDR@alllinenos
```

```latex
\ProvideDocumentCommand\captionof{mm}{}
\def\CDR@alllinenos{(0)}

\def\FormatLineNumber#1{{\rmfamily\tiny#1}}

\newdimen\CDR@leftmargin
\newdimen\CDR@linenosep

%
%\newcommand\CDR@tcbox@more@options{%
%  nobeforeafter,%
%  tcbox~raise~base,%
%  left=0mm,%
%  right=0mm,%
%  top=0mm,%
%  bottom=0mm,%
%  boxsep=2pt,%
%  arc=1pt,%
%  boxrule=0pt,%
%  \CDR_options_if_in:nT {colback} {
%    colback=\CDR:n {colback}
%  }
%}
%
%\newcommand\CDR@mdframed@more@options{%
%  leftmargin=\CDR@leftmargin,%
%  frametitlerule=true,%
%  \CDR_if_in:nT {colback} {
%    backgroundcolor=\CDR:n {colback}
%  }
%}
%
%\newcommand\CDR@tcolorbox@more@options{%
%  grow~to~left~by=-\CDR@leftmargin,%
%  \CDR_if_in:nNT {colback} {
%    colback=\CDR:n {colback}
%  }
%}
%
%\newcommand\CDR@boite@more@options{%
%  leftmargin=\CDR@leftmargin,%
%  \ifcsname CDR@opt@colback\endcsname
%    colback=\CDR@opt@colback,%
%  \fi
%}
%
%\newcommand\CDR@mdframed@margin{%
%  \advance \CDR@linenosep \mdflength{outerlinewidth}%
%  \advance \CDR@linenosep \mdflength{middlelinewidth}%
%  \advance \CDR@linenosep \mdflength{innerlinewidth}%
%  \advance \CDR@linenosep \mdflength{innerleftmargin}%
%}
%
```

```
1593 %\newcommand\CDR@tcolorbox@margin{%
1594 %  \advance \CDR@linenosep \kvtcb@left@rule
1595 %  \advance \CDR@linenosep \kvtcb@leftupper
1596 %  \advance \CDR@linenosep \kvtcb@boxsep
1597 %}
1598 %
1599 %\newcommand\CDR@boite@margin{%
1600 %  \advance \CDR@linenosep \boite@leftrule
1601 %  \advance \CDR@linenosep \boite@boxsep
1602 %}
1603 %
1604 %\def\CDR@global@options{}
1605 %
1606 %\newcommand\setpygmented[1]{%
1607 %  \def\CDR@global@options{/CDR.cd,#1}%
1608 %}
1609

1610 \ExplSyntaxOff

1611 %</sty>
```