

`coder` — code inlined in a \LaTeX document*

Jérôme LAURENS[†]

Released 2022/02/07

Abstract

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

This \LaTeX package requires $\text{Lua}\text{\TeX}$ and may use syntax coloring based on the `pygments`¹ package.

1 Package dependencies

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

2 Similar technologies

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

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

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

3 Known bugs and limitations

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

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

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

¹The `coder` package has been tested with `pygments` version 2.11.2

4 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 insert code snippets as blocks. Moreover, block code chunks can be exported to files, once declared with `\CDRExport` command. The `\CDRSet` command is used to set various parameters, including display engines declared with either `\CDRCodeEngineNew` or `\CDRBlockEngineNew`².

4.1 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 then 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.sty` is able to input 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 conjunction with `coder.sty`.

The package `coder.sty` only exchanges with `coder-util.lua` using `\directlua`, `tex.print` and `token.get_macro`. `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.

4.2 File exportation

1. The `\CDRExport` command declares a file path, a list of tags and other usefull informations 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.

²Work in progress

4.3 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`.

4.4 L^AT_EX user interface

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

4.5 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.

5 Namespace and conventions

L^AT_EX identifiers related to `coder` start with `CDR`, including both commands and environment. `expl3` identifiers also start with `CDR`, after and eventual leading `c_`, `l_` or `g_`. `l3keys` module path's first component is either `CDR` or starts with `CDR@`.

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

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

6 Options

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

6.1 fancyvrb

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

- **formatcom**=`<command>` execute before printing verbatim text. Initially empty. Ignored in `code` mode.
- **fontfamily**=`<family name>` font family to use. `tt`, `courier` and `helvetica` are pre-defined. Initially `tt`.

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

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

6.2 pygments options

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

- **style= $\langle name \rangle$** the **pygments** style to use. Initially **default**.
- ⊘ **full** Tells the formatter to output a **full** document, i.e. a complete self-contained document (default: **false**). Forbidden.
- ⊘ **title** If **full** is true, the title that should be used to caption the document (default empty). Forbidden.

- ⊘ **encoding** If given, must be an encoding name. This will be used to convert the Unicode token strings to byte strings in the output. If it is `None`, Unicode strings will be written to the output file, which most file-like objects do not support (default: `None`).
- ⊘ **outencoding** Overrides **encoding** if given.
- ⊘ **docclass** If the **full** option is enabled, this is the document class to use (default: `article`). Forbidden.
- ⊘ **preamble** If the **full** option is enabled, this can be further preamble commands, e.g. `"\usepackage"` (default `empty`). Forbidden.
- ⊘ **linenos**`[=true|false]` If set to `true`, output line numbers. Initially `false`: no numbering. Ignored in `code` mode.
- ⊘ **linenostart**`=<integer>` The line number for the first line. Initially 1: numbering starts from 1. Ignored in `code` mode.
- ⊘ **linenostep**`=<integer>` If set to a number $n > 1$, only every n th line number is printed. Ignored in `code` mode. Additional options given to the `Verbatim` environment (see the `fancyvrb` docs for possible values). Initially empty.
- ⊘ **verboptions** Forbidden.
- **commandprefix**`=<text>` The LaTeX commands used to produce colored output are constructed using this prefix and some letters. Initially `PY`.
- **texcomments**`[=true|false]` If set to `true`, enables LaTeX comment lines. That is, LaTeX markup in comment tokens is not escaped so that LaTeX can render it. Initially `false`. Ignored in `code` mode.
- **mathescape**`[=true|false]` If set to `true`, enables LaTeX math mode escape in comments. That is, `$...$` inside a comment will trigger math mode. Initially `false`.
- **escapeinside**`=<before><after>` If set to a string of length 2, enables escaping to LaTeX. Text delimited by these 2 characters is read as LaTeX code and typeset accordingly. It has no effect in string literals. It has no effect in comments if **texcomments** or **mathescape** is set. Initially empty.
- ⚙ **envname**`=<name>` Allows you to pick an alternative environment name replacing `Verbatim`. The alternate environment still has to support `Verbatim`'s option syntax. Initially `Verbatim`.

6.3 LaTeX

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

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

All the line templates below are L^AT_EX source text where `<placeholder:number>` should be replaced by a line number and `<placeholder:line>` should be replaced by the highlighted line code provided by `pygments`. They should not include a trailing newline char. The *<type>* is used to describe the line more precisely.

- **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-json')
```

3 General purpose material

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

```
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.

```

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* string for commands and execute them. The patterns being searched are enclosed within opening <<<<< and closing >>>>>, each containing 5 characters,

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

?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

highlight_set CDR:highlight_set(...)

Highlight the currently entered block. Build a configuration table with all data necessary for the processing, save it as a JSON file and launch coder-tool.py with the proper arguments.

```

131 local function highlight_set(self, key, value)
132   local args = self['.arguments']
133   local t = args
134   if t[key] == nil then
135     t = args.pygopts
136     if t[key] == nil then
137       t = args.texopts
138       if t[key] == nil then
139         t = args.fv_opts

```

```

140         assert(t[key] ~= nil)
141     end
142 end
143 end
144 t[key] = value
145 end
146
147 local function hilight_set_var(self, key, var)
148     self:hilight_set(key, assert(token.get_macro(var or 'l_CDR_tl'))))
149 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>.

```

150 local function hilight_source(self, sty, src)
151     local args = self['.arguments']
152     local texopts = args.texopts
153     local pygopts = args.pygopts
154     local inline = texopts.is_inline
155     local use_cache = self.is_truthy(args.cache)
156     local use_py = false
157     local cmd = self.PYTHON_PATH..' '..self.CDR_PY_PATH
158     local debug = args.debug
159     local pyg_sty_p
160     if sty then
161         pyg_sty_p = self.dir_p..pygopts.style..'pyg.sty'
162         token.set_macro('l_CDR_pyg_sty_tl', pyg_sty_p)
163         texopts.pyg_sty_p = pyg_sty_p
164         local mode,_,_ = lfs.attributes(pyg_sty_p, 'mode')
165         if not mode or not use_cache then
166             use_py = true
167             if debug then
168                 print('PYTHON STYLE:')
169             end
170             cmd = cmd..' --create_style'
171         end
172         self:cache_record(pyg_sty_p)
173     end
174     local pyg_tex_p
175     if src then
176         local source
177         if inline then
178             source = args.source
179         else
180             local ll = self['.lines']
181             source = table.concat(ll, '\n')
182         end
183         local hash = md5.sumhexa(('%s:%s:%s'

```

```

184         ):format(
185             source,
186             inline and 'code' or 'block',
187             pygopts.style
188         )
189     )
190     local base = self.dir_p..hash
191     pyg_tex_p = base..'pyg.tex'
192     token.set_macro('l_CDR_pyg_tex_tl', pyg_tex_p)
193     local mode,_,_ = lfs.attributes(pyg_tex_p,'mode')
194     if not mode or not use_cache then
195         use_py = true
196         if debug then
197             print('PYTHON SOURCE:', inline)
198         end
199         if not inline then
200             local tex_p = base..'tex'
201             local f = assert(io.open(tex_p, 'w'))
202             local ok, err = f:write(source)
203             f:close()
204             if not ok then
205                 print('File error('..tex_p..'): '..err)
206             end
207             if debug then
208                 print('OUTPUT: '..tex_p)
209             end
210         end
211         cmd = cmd..(' --base=%q'):format(base)
212     end
213 end
214 if use_py then
215     local json_p = self.json_p
216     local f = assert(io.open(json_p, 'w'))
217     local ok, err = f:write(json.tostring(args, true))
218     f:close()
219     if not ok then
220         print('File error('..json_p..'): '..err)
221     end
222     cmd = cmd..(' %q'):format(json_p)
223     if debug then
224         print('CDR>'..cmd)
225     end
226     local o = io.popen(cmd):read('a')
227     self:load_exec_output(o)
228     if debug then
229         print('PYTHON', o)
230     end
231 end
232 self:cache_record(
233     sty and pyg_sty_p or nil,
234     src and pyg_tex_p or nil
235 )
236 end

```

5.2 Code

5.3 Code

`highlight_code_setup` CDR:highlight_code_setup()

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

```
237 local function highlight_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__ = 'FV0pts',
256     }
257   }
258   self.highlight_json_written = false
259 end
260
```

5.4 Block

`highlight_block_setup` CDR:highlight_block_setup(`<tags clist var>`)

Records the contents of the `<tags clist var>` L^AT_EX variable to prepare block highlighting.

```
261 local function highlight_block_setup(self, tags_clist_var)
262   local tags_clist = assert(token.get_macro(assert(tags_clist_var)))
263   self['.tags clist'] = tags_clist
264   self['.lines'] = {}
265   self['.arguments'] = {
266     __cls__ = 'Arguments',
267     cache = false,
268     debug = false,
269     source = nil,
270     pygopts = {
271       __cls__ = 'PygOpts',
272       lang = 'tex',

```

```

273     style = 'default',
274     texcomments = false,
275     mathescape = false,
276     escapeinside = '',
277 },
278 texopts = {
279     __cls__ = 'TeXOpts',
280     tags = tags_clist,
281     is_inline = false,
282     pyg_sty_p = '',
283 },
284 fv_opts = {
285     __cls__ = 'FVOpts',
286     firstnumber = 1,
287     stepnumber = 1,
288 }
289 }
290 self.highlight_json_written = false
291 end

```

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

Store the content of the given named variable. It will be used for colorization and exportation.

```

292 local function record_line(self, line_variable_name)
293     local line = assert(token.get_macro(assert(line_variable_name)))
294     local ll = assert(self['.lines'])
295     ll[#ll+1] = line
296 end

```

highlight_block_teardown CDR:highlight_block_teardown()

Records the contents of the *<tags clist var>* L^AT_EX variable to prepare block highlighting.

```

297 local function highlight_block_teardown(self)
298     local ll = assert(self['.lines'])
299     if #ll > 0 then
300         local records = self['.records'] or {}
301         self['.records'] = records
302         local t = {
303             already = {},
304             code = table.concat(ll, '\n')
305         }
306         for tag in self['.tags clist']:gmatch('([^\,]+)') do
307             local tt = records[tag] or {}
308             records[tag] = tt
309             tt[#tt+1] = t
310         end
311     end
312 end

```

6 Exportation

For each file to be exported, `coder.sty` calls `export_file` to initialize 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.

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

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

```
313 local function export_file(self, file_name_var)
314   self['.name'] = assert(token.get_macro(assert(file_name_var)))
315   self['.export'] = {}
316 end
```

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

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

```
317 local function export_file_info(self, key, value)
318   local export = self['.export']
319   value = assert(token.get_macro(assert(value)))
320   export[key] = value
321 end
```

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

This is called at export time.

```
322 local function export_complete(self)
323   local name    = self['.name']
324   local export  = self['.export']
325   local records = self['.records']
326   local raw     = export.raw == 'true'
327   local tt      = {}
328   local s
329   if not raw then
330     s = export.preamble
331     if s and #s>0 then
332       tt[#tt+1] = s
333     end
334   end
335   for tag in string.gmatch(export.tags, '([^\,]+)') do
336     local Rs = records[tag]
337     if Rs then
338       for _,R in ipairs(Rs) do
339         if not R.already[name] or not once then
340           tt[#tt+1] = R.code
341         end
342         if once then
343           R.already[name] = true
344         end
345       end
346     end
347   end
```



```

344         end
345     end
346 end
347 end
348 if not raw then
349     s = export.postamble
350     if s and #s>0 then
351         tt[#tt+1] = s
352     end
353 end
354 if #tt>0 then
355     local fh = assert(io.open(name,'w'))
356     fh:write(table.concat(tt, '\n'))
357     fh:close()
358 end
359 self['.name'] = nil
360 self['.export'] = nil
361 end

```

7 Caching

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

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

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

```

362 local function cache_clean_all(self)
363     local to_remove = {}
364     for f in lfs.dir(self.dir_p) do
365         to_remove[f] = true
366     end
367     for k,_ in pairs(to_remove) do
368         os.remove(self.dir_p .. k)
369     end
370 end
371 local function cache_record(self, pyg_sty_p, pyg_tex_p)
372     if pyg_sty_p then
373         self['.style_set'] [pyg_sty_p] = true
374     end
375     if pyg_tex_p then

```

```

376     self['.colored_set'][pyg_tex_p] = true
377 end
378 end
379 local function cache_clean_unused(self)
380     local to_remove = {}
381     for f in lfs.dir(self.dir_p) do
382         f = self.dir_p .. f
383         if not self['.style_set'][f] and not self['.colored_set'][f] then
384             to_remove[f] = true
385         end
386     end
387     for f,_ in pairs(to_remove) do
388         os.remove(f)
389     end
390 end

```

_DESCRIPTION Short text description of the module.

```

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

```

8 Return the module

```

392 return {
    Known fields are

393     _DESCRIPTION      = _DESCRIPTION,

    _VERSION to store <version string>,

394     _VERSION          = token.get_macro('fileversion'),

    date to store <date string>,

395     date              = token.get_macro('filedate'),

    Various paths ,

396     CDR_PY_PATH       = CDR_PY_PATH,
397     PYTHON_PATH       = PYTHON_PATH,
398     set_python_path   = set_python_path,

    is_truthy

399     is_truthy         = is_truthy,

    escape

400     escape            = escape,

    make_directory

```

```

401     make_directory      = make_directory,

        load_exec

402     load_exec           = load_exec,

403     load_exec_output    = load_exec_output,

        record_line

404     record_line         = record_line,

        highlight common

405     highlight_set       = highlight_set,
406     highlight_set_var   = highlight_set_var,
407     highlight_source    = highlight_source,

        highlight code

408     highlight_code_setup = highlight_code_setup,

        highlight_block_setup

409     highlight_block_setup = highlight_block_setup,
410     highlight_block_teardown = highlight_block_teardown,

        cache

411     cache_clean_all     = cache_clean_all,
412     cache_record        = cache_record,
413     cache_clean_unused  = cache_clean_unused,

        Internals

414     ['.style_set']      = {},
415     ['.colored_set']    = {},
416     ['.options']        = {},
417     ['.export']         = {},
418     ['.name']           = nil,

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

419     already             = false,

        Other

420     dir_p               = dir_p,
421     json_p              = json_p,

        Exportation

```

```

422 export_file      = export_file,
423 export_file_info  = export_file_info,
424 export_complete   = export_complete,
425 }
426 %</lua>

```

File II

coder-tool.py implementation

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

```

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

```

1 Usage

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

2 Header and global declarations

```

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

```

3 Options classes

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

```

20 class BaseOpts(object):
21     @staticmethod
22     def ensure_bool(x):
23         if x == True or x == False: return x
24         x = x[0:1]
25         return x == 'T' or x == 't'

26     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

```

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

```

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

```

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

```

49 class PygOpts(BaseOpts):
50     style = 'default'
51     nobackground = False
52     linenos = False
53     linenostart = 1
54     linenostep = 1
55     commandprefix = 'Py'
56     texcomments = False
57     mathescape = False
58     escapeinside = ""

```

```

59 envname = 'Verbatim'
60 lang = 'tex'
61 def __init__(self, *args, **kwargs):
62     super().__init__(*args, **kwargs)
63     self.linenos = self.ensure_bool(self.linenos)
64     self.linenostart = abs(int(self.linenostart))
65     self.linenostep = abs(int(self.linenostep))
66     self.texcomments = self.ensure_bool(self.texcomments)
67     self.mathescape = self.ensure_bool(self.mathescape)

```

3.3 FVclass

```

68 class FVOpts(BaseOpts):
69     gobble = 0
70     tabsize = 4
71     linenosep = '0pt'
72     commentchar = ''
73     frame = 'none'
74     framerule = '0.4pt',
75     framesep = r'\fboxsep',
76     rulecolor = 'black',
77     fillcolor = '',
78     label = ''
79     labelposition = 'none'
80     numbers = 'left'
81     numbersep = '1ex'
82     firstnumber = 'auto'
83     stepnumber = 1
84     numberblanklines = True
85     firstline = ''
86     lastline = ''
87     baselinestretch = 'auto'
88     resetmargins = True
89     xleftmargin = '0pt'
90     xrightmargin = '0pt'
91     hfuzz = '2pt'
92     vspace = r'\topsep'
93     samepage = False
94     def __init__(self, *args, **kwargs):
95         super().__init__(*args, **kwargs)
96         self.gobble = abs(int(self.gobble))
97         self.tabsize = abs(int(self.tabsize))
98         if self.firstnumber != 'auto':
99             self.firstnumber = abs(int(self.firstnumber))
100         self.stepnumber = abs(int(self.stepnumber))
101         self.numberblanklines = self.ensure_bool(self.numberblanklines)
102         self.resetmargins = self.ensure_bool(self.resetmargins)
103         self.samepage = self.ensure_bool(self.samepage)

```

3.4 Argumentsclass

```

104 class Arguments(BaseOpts):
105     cache = False
106     debug = False

```

```

107 source = ""
108 style  = "default"
109 json   = ""
110 directory = "."
111 texopts = TeXOpts()
112 pygopts = PygOpts()
113 fv_opts = FVOpts()

```

4 Controller main class

```

114 class Controller:

```

4.1 Static methods

object_hook Helper for json parsing.

```

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

```

lua_command self.lua_command(*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 or executed synchronously.

```

126 @staticmethod
127 def lua_command(cmd):
128     print(f'<<<<<*LUA:{cmd}>>>>>')
129 @staticmethod
130 def lua_command_now(cmd):
131     print(f'<<<<<!LUA:{cmd}>>>>>')
132 @staticmethod
133 def lua_debug(msg):
134     print(f'<<<<<?LUA:{msg}>>>>>')

```

lua_text_escape self.lua_text_escape(*text*)

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

```

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

```

4.2 Computed properties

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

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

```

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

```

self.parser The correctly set up argparse instance.

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

```

154     @property
155     def parser(self):
156         parser = argparse.ArgumentParser(
157             prog=sys.argv[0],
158             description='''
159 Writes to the output file a set of LaTeX macros describing
160 the syntax hilighting of the input file as given by pygments.
161 ''')
162     )
163     parser.add_argument(
164         "-v", "--version",
165         help="Print the version and exit",
166         action='version',
167         version=f'coder-tool version {__version__},
168         ' (c) {__YEAR__} by Jérôme LAURENS.'
169     )
170     parser.add_argument(
171         "--debug",
172         action='store_true',
173         default=None,
174         help="display informations useful for debugging"
175     )
176     parser.add_argument(
177         "--create_style",

```



```

178     action='store_true',
179     default=None,
180     help="create the style definitions"
181 )
182 parser.add_argument(
183     "--base",
184     action='store',
185     default=None,
186     help="the path of the file to be colored, with no extension"
187 )
188 parser.add_argument(
189     "json",
190     metavar="<json data file>",
191     help=""
192 file name with extension, contains processing information.
193 """
194 )
195 return parser
196

```

4.3 Methods

4.3.1 __init__

__init__ Constructor. Reads the command line arguments.

```

197 def __init__(self, argv = sys.argv):
198     argv = argv[1:] if re.match(".*coder\-tool\.py$", argv[0]) else argv
199     ns = self.parser.parse_args(
200         argv if len(argv) else ['-h']
201     )
202     with open(ns.json, 'r') as f:
203         self.arguments = json.load(
204             f,
205             object_hook = Controller.object_hook
206         )
207     args = self.arguments
208     args.json = ns.json
209     self.texopts = args.texopts
210     pygopts = self.pygopts = args.pygopts
211     fv_opts = self.fv_opts = args.fv_opts
212     self.formatter = LatexFormatter(
213         style = pygopts.style,
214         nobackground = pygopts.nobackground,
215         commandprefix = pygopts.commandprefix,
216         texcomments = pygopts.texcomments,
217         mathescape = pygopts.mathescape,
218         escapeinside = pygopts.escapeinside,
219         envname = 'CDR@Pyg@Verbatim',
220     )
221
222     try:

```

```

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

```

4.3.2 create_style

self.create_style self.create_style()

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

```

258 def create_style(self):
259     args = self.arguments
260     if not args.create_style:
261         return
262     texopts = args.texopts
263     pyg_sty_p = texopts.pyg_sty_p
264     if args.cache and pyg_sty_p.exists():
265         return
266     texopts = self.texopts
267     style = self.pygopts.style
268     formatter = self.formatter
269     style_defs = formatter.get_style_defs() \

```

```

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

```

4.3.3 pygmentize

```

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

```

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

```

293 def pygmentize(self, source):
294     source = highlight(source, self.lexer, self.formatter)
295     m = re.match(
296         r'\begin{CDR@Pyg@Verbatim}.*?\n(.*)\n\\end{CDR@Pyg@Verbatim}\s*\Z',
297         source,
298         flags=re.S
299     )
300     assert(m)
301     highlighted = m.group(1)
302     texopts = self.texopts
303     if texopts.is_inline:
304         return highlighted.replace(' ', r'\CDR@Sp ') + r'\ignorespaces'
305     lines = highlighted.split('\n')
306     ans_code = []
307     last = 1
308     for line in lines[1:]:
309         last += 1
310         ans_code.append(rf'''\CDR@Line{{{last}}}{{{{line}}}}''')
311     if len(lines):
312         ans_code.insert(0, rf'''\CDR@Line[last={last}]{{{{1}}}{{{{lines[0]}}}}''')
313     highlighted = '\n'.join(ans_code)
314     return highlighted

```

4.3.4 create_pygmented

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

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

```
315 def create_pygmented(self):
316     args = self.arguments
317     base = args.base
318     if not base:
319         return False
320     source = args.source
321     if not source:
322         tex_p = Path(base).with_suffix('.tex')
323         with open(tex_p, 'r') as f:
324             source = f.read()
325     pyg_tex_p = Path(base).with_suffix('.pyg.tex')
326     highlighted = self.pygmentize(source)
327     with pyg_tex_p.open(mode='w', encoding='utf-8') as f:
328         f.write(highlighted)
329     if args.debug:
330         print('HIGHLIGHTED', os.path.relpath(pyg_tex_p))
```

4.4 Main entry

```
331 if __name__ == '__main__':
332     try:
333         ctrl = Controller()
334         x = ctrl.create_style() or ctrl.create_pygmented()
335         print(f'{sys.argv[0]}: done')
336         sys.exit(x)
337     except KeyboardInterrupt:
338         sys.exit(1)
339 %</py>
```

File III

coder.sty implementation

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

1 Installation test

```
3 \NewDocumentCommand \CDRTest {} {
4   \sys_if_shell:TF {
5     \CDR_has_pygments:F {
6       \msg_warning:nnn
7       { coder }
8       { :n }
9       { No~"pygmentize"~found. }
```

```

10     }
11   } {
12     \msg_warning:nnn
13     { coder }
14     { :n }
15     { No~unrestricted~shell~escape~for~"pygmentize".}
16   }
17 }

```

2 Messages

```

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

```

3 Constants

`\c_CDR_tag` Paths of L3keys modules.

`\c_CDR_Tags` These are root path components used throughout the package. 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 package.

```

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.

Many functions have useful hooks for debugging or testing.

`\CDR@Debug` `\CDR@Debug {⟨argument⟩}`

The default implementation just gobbles its argument. During development or testing, this may call `\typeout`.

```

24 \cs_new:Npn \CDR@Debug { \use_none:n }

```

5 Variables

5.1 Internal scratch variables

These local variables are used in a very limited scope.

`\l_CDR_bool` Local scratch variable.

25 `\bool_new:N \l_CDR_bool`

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

`\l_CDR_tl` Local scratch variable.

26 `\tl_new:N \l_CDR_tl`

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

`\l_CDR_str` Local scratch variable.

27 `\str_new:N \l_CDR_str`

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

`\l_CDR_seq` Local scratch variable.

28 `\seq_new:N \l_CDR_seq`

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

`\l_CDR_prop` Local scratch variable.

29 `\prop_new:N \l_CDR_prop`

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

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

30 `\clist_new:N \l_CDR_clist`

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

5.2 Files

`\l_CDR_ior` Input file identifier

31 `\ior_new:N \l_CDR_ior`

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

`\l_CDR_iow` Output file identifier

32 `\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.

33 `\int_new:N \g_CDR_source_int`

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

`\g_CDR_source_prop` Global source property list.

34 `\prop_new:N \g_CDR_source_prop`

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

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

35 `\tl_new:N \g_CDR_chunks_tl`

36 `\tl_new:N \l_CDR_chunks_tl`

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

`\g_CDR_vars` Tree storage for global variables.

37 `\prop_new:N \g_CDR_vars`

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

`\g_CDR_hook_tl` Hook general purpose.

38 `\tl_new:N \g_CDR_hook_tl`

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

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

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

5.4 Local variables

`\l_CDR_kv_clist` keyval storage.

39 `\clist_new:N \l_CDR_kv_clist`

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

`\l_CDR_opts_tl` options storage.

40 `\tl_new:N \l_CDR_opts_tl`

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

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

41 `\tl_new:N \l_CDR_recorded_tl`

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

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

42 \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.

43 \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.

44 \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.

45 \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.

46 \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.

47 \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>.

```
48 \cs_new:Npn \CDR_int_new:cn #1 #2 {
49   \int_new:c { CDR@int.#1 }
50   \int_gset:cn { CDR@int.#1 } { #2 }
51 }
```

default Generic and named line number counter.

```
--line 52 \CDR_int_new:cn { default } { 1 }
53 \CDR_int_new:cn { __ } { 1 }
54 \CDR_int_new:cn { __line } { 1 }
```


(End definition for `default`, `__`, and `__line`. This variable is documented on page ??.)

`\CDR_int:c` ★ `\CDR_int:c {<tag name>}`
 Use the integer named after `<tag name>`.

```
55 \cs_new:Npn \CDR_int:c #1 {
56   \use:c { CDR@int.#1 }
57 }
```

`\CDR_int_use:c` ★ `\CDR_int_use:n {<tag name>}`
 Use the value of the integer named after `<tag name>`.

```
58 \cs_new:Npn \CDR_int_use:c #1 {
59   \int_use:c { CDR@int.#1 }
60 }
```

`\CDR_int_if_exist_p:c` ★ `\CDR_int_if_exist:cTF {<tag name>} {<true code>} {<false code>}`
`\CDR_int_if_exist:cTF` ★ Execute `<true code>` when an integer named after `<tag name>` exists, `<false code>` otherwise.

```
61 \prg_new_conditional:Nnn \CDR_int_if_exist:c { p, T, F, TF } {
62   \int_if_exist:cTF { CDR@int.#1 } {
63     \prg_return_true:
64   } {
65     \prg_return_false:
66   }
67 }
```

`\CDR_int_compare_p:cNn` ★ `\CDR_int_compare:cNnTF {<tag name>} <operator> {<intexpr2>} {<true code>} {<false code>}`
`\CDR_int_compare:cNnTF` ★ Forwards to `\int_compare...` with `\CDR_int_use:c { #1 }`.

```
68 \prg_new_conditional:Nnn \CDR_int_compare:cNn { p, T, F, TF } {
69   \int_compare:nNnTF { \CDR_int:c { #1 } } #2 { #3 } {
70     \prg_return_true:
71   } {
72     \prg_return_false:
73   }
74 }
```

<u>\CDR_int_set:cn</u>	\CDR_int_set:cn {<tag name>} {<value>}
<u>\CDR_int_gset:cn</u>	Set the integer named after <tag name> to the <value>. \CDR_int_gset:cn makes a global change.
<pre> 75 \cs_new:Npn \CDR_int_set:cn #1 #2 { 76 \int_set:cn { CDR@int.#1 } { #2 } 77 } 78 \cs_new:Npn \CDR_int_gset:cn #1 #2 { 79 \int_gset:cn { CDR@int.#1 } { #2 } 80 }</pre>	
<u>\CDR_int_set:cc</u>	\CDR_int_set:cc {<tag name>} {<other tag name>}
<u>\CDR_int_gset:cc</u>	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.
<pre> 81 \cs_new:Npn \CDR_int_set:cc #1 #2 { 82 \CDR_int_set:cn { #1 } { \CDR_int:c { #2 } } 83 } 84 \cs_new:Npn \CDR_int_gset:cc #1 #2 { 85 \CDR_int_gset:cn { #1 } { \CDR_int:c { #2 } } 86 }</pre>	
<u>\CDR_int_add:cn</u>	\CDR_int_add:cn {<tag name>} {<value>}
<u>\CDR_int_gadd:cn</u>	Add the <value> to the integer named after <tag name>. \CDR_int_gadd:cn makes a global change.
<pre> 87 \cs_new:Npn \CDR_int_add:cn #1 #2 { 88 \int_add:cn { CDR@int.#1 } { #2 } 89 } 90 \cs_new:Npn \CDR_int_gadd:cn #1 #2 { 91 \int_gadd:cn { CDR@int.#1 } { #2 } 92 }</pre>	
<u>\CDR_int_add:cc</u>	\CDR_int_add:cn {<tag name>} {<other tag name>}
<u>\CDR_int_gadd:cc</u>	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.
<pre> 93 \cs_new:Npn \CDR_int_add:cc #1 #2 { 94 \CDR_int_add:cn { #1 } { \CDR_int:c { #2 } } 95 } 96 \cs_new:Npn \CDR_int_gadd:cc #1 #2 { 97 \CDR_int_gadd:cn { #1 } { \CDR_int:c { #2 } } 98 }</pre>	
<u>\CDR_int_sub:cn</u>	\CDR_int_sub:cn {<tag name>} {<value>}
<u>\CDR_int_gsub:cn</u>	Subtract the <value> from the integer named after <tag name>. \CDR_int_gsub:cn makes a global change.

```

99 \cs_new:Npn \CDR_int_sub:cn #1 #2 {
100   \int_sub:cn { CDR@int.#1 } { #2 }
101 }
102 \cs_new:Npn \CDR_int_gsub:cn #1 #2 {
103   \int_gsub:cn { CDR@int.#1 } { #2 }
104 }

```

5.6 Utilities

`\g_CDR_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. The `\g_CDR_last_tags_clist` variable contains the last list of tags that was displayed.

```

105 \clist_new:N \g_CDR_tags_clist
106 \clist_new:N \g_CDR_all_tags_clist
107 \clist_new:N \g_CDR_last_tags_clist
108 \AddToHook { shipout/before } {
109   \clist_gclear:N \g_CDR_last_tags_clist
110 }

```

(End definition for `\g_CDR_tags_clist`, `\g_CDR_all_tags_clist`, and `\g_CDR_last_tags_clist`. These variables are documented on page ??.)

```

111 \prg_new_conditional:Nnn \CDR_clist_if_eq:NN { p, T, F, TF } {
112   \tl_if_eq:NNTF #1 #2 {
113     \prg_return_true:
114   } {
115     \prg_return_false:
116   }
117 }

```

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

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

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

```

118 \cs_new:Npn \CDR_tag_get_path:cc #1 #2 {
119   \c_CDR_tag_get @ #1 / #2
120 }
121 \cs_new:Npn \CDR_tag_get_path:c {
122   \CDR_tag_get_path:cc { __local }
123 }

```

6.2 Set

<code>\CDR_tag_set:ccn</code> <code>\CDR_tag_set:ccV</code>	<code>\CDR_tag_set:ccn {⟨tag name⟩} {⟨relative key path⟩} {⟨value⟩}</code> Store $\langle value \rangle$, which is further retrieved with the instruction <code>\CDR_tag_get:cc {⟨tag name⟩} {⟨relative key path⟩}</code> . Only $\langle tag name \rangle$ and $\langle relative key path \rangle$ containing no @ character are supported. All the affectations are made at the current T _E X group level. <i>Nota Bene:</i> <code>\cs_generate_variant:Nn</code> is buggy when there is a ‘c’ argument.
--	---

```

124 \cs_new_protected:Npn \CDR_tag_set:ccn #1 #2 #3 {
125   \cs_set:cpn { \CDR_tag_get_path:cc { #1 } { #2 } } { \exp_not:n { #3 } }
126 }
127 \cs_new_protected:Npn \CDR_tag_set:ccV #1 #2 #3 {
128   \exp_args:NnnV
129   \CDR_tag_set:ccn { #1 } { #2 } #3
130 }

```

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

```

131 \tl_set:Nn \l_CDR_tl { /([~/]*)/(.*)$ } \use_none:n { $ }
132 \tl_put_left:NV \l_CDR_tl \c_CDR_tag
133 \tl_put_left:Nn \l_CDR_tl { ^ }
134 \exp_args:NNV
135 \regex_const:Nn \c_CDR_tag_regex \l_CDR_tl

```

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

<code>\CDR_tag_set:n</code>	<code>\CDR_tag_set:n {⟨value⟩}</code> The value is provided but not the $\langle dir \rangle$ nor the $\langle relative key path \rangle$, both are guessed from <code>\l_keys_path_str</code> . More precisely, <code>\l_keys_path_str</code> is expected to read something like <code>\c_CDR_tag/⟨tag name⟩/⟨relative key path⟩</code> , an error is raised on the contrary. This is meant to be called from <code>\keys_define:nn</code> argument. Implementation detail: the last argument is parsed by the last command.
-----------------------------	---

```

136 \cs_new_protected:Npn \CDR_tag_set:n {
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     { \seq_item:Nn \l_CDR_seq 3 }
143   } {
144     \PackageWarning
145     { coder }
146     { Unexpected~key~path~‘\l_keys_path_str’ }
147     \use_none:n
148   }
149 }

```

<code>\CDR_tag_set:</code>	<code>\CDR_tag_set:</code> None of $\langle dir \rangle$, $\langle relative key path \rangle$ and $\langle value \rangle$ are provided. The latter is guessed from <code>\l_keys_value_tl</code> , and <code>\CDR_tag_set:n</code> is called. This is meant to be call from <code>\keys_define:nn</code> argument.
----------------------------	--

```

150 \cs_new_protected:Npn \CDR_tag_set: {
151   \exp_args:NV
152   \CDR_tag_set:n \l_keys_value_tl
153 }

```

\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.

```

154 \cs_new:Npn \CDR_tag_set:cn #1 {
155   \exp_args:NnV
156   \regex_extract_once:NnNTF \c_CDR_tag_regex
157   \l_keys_path_str \l_CDR_seq {
158     \CDR_tag_set:ccn
159     { \seq_item:Nn \l_CDR_seq 2 }
160     { #1 }
161   } {
162     \PackageWarning
163     { coder }
164     { Unexpected~key~path~‘\l_keys_path_str’ }
165     \use_none:n
166   }
167 }

```

\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.

```

168 \prg_generate_conditional_variant:Nnn \str_if_eq:nn { Vn } { p, T, F, TF }
169
170 \regex_const:Nn \c_CDR_root_regex { ^(.*)/.*$ } \use_none:n { $ }
171 \cs_new:Npn \CDR_tag_choices: {
172   \str_if_eq:nnT \l_keys_key_tl \l_keys_choice_tl {
173     \exp_args:NnV
174     \regex_extract_once:NnNT \c_CDR_root_regex
175     \l_keys_path_str \l_CDR_seq {
176       \str_set:Nx \l_keys_path_str {
177         \seq_item:Nn \l_CDR_seq 2
178       }
179     }
180   }
181 }

```

\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.

```

182 \cs_new_protected:Npn \CDR_tag_choices_set: {
183   \CDR_tag_choices:
184   \exp_args:NV
185   \CDR_tag_set:n \l_keys_choice_tl
186 }

```

<pre> \CDR_tag_if_truthy_p:cc * \CDR_tag_if_truthy:ccTF * \CDR_tag_if_truthy_p:c * \CDR_tag_if_truthy:cTF * </pre>	<pre> \CDR_tag_if_truthy:ccTF {<tag name>} {<relative key path>} {<true code>} {<false code>} \CDR_tag_if_truthy:cTF {<relative key path>} {<true code>} {<false code>} </pre> <p>Execute <i><true code></i> when the property for <i><tag name></i> and <i><relative key path></i> is a truthy value, <i><false code></i> otherwise. A truthy value is a text which is not “false” in a case insensitive comparison. In the second version, the <i><tag name></i> is not provided and set to <code>__local</code>.</p>
--	---

```

187 \prg_new_conditional:Nnn \CDR_tag_if_truthy:cc { p, T, F, TF } {
188   \exp_args:Ne
189   \str_compare:nNnTF {
190     \exp_args:Ne \str_lowercase:n { \CDR_tag_get:cc { #1 } { #2 } }
191   } = { true } {
192     \prg_return_true:
193   } {
194     \prg_return_false:
195   }
196 }
197 \prg_new_conditional:Nnn \CDR_tag_if_truthy:c { p, T, F, TF } {
198   \exp_args:Ne
199   \str_compare:nNnTF {
200     \exp_args:Ne \str_lowercase:n { \CDR_tag_get:c { #1 } }
201   } = { true } {
202     \prg_return_true:
203   } {
204     \prg_return_false:
205   }
206 }

```

<pre> \CDR_tag_if_eq_p:ccn * \CDR_tag_if_eq:ccnTF * \CDR_tag_if_eq_p:cn * \CDR_tag_if_eq:cnTF * </pre>	<pre> \CDR_tag_if_eq:ccnTF {<tag name>} {<relative key path>} {<value>} {<true code>} {<false code>} \CDR_tag_if_eq:cnTF {<relative key path>} {<value>} {<true code>} {<false code>} </pre> <p>Execute <i><true code></i> when the property for <i><tag name></i> and <i><relative key path></i> is equal to <i><value></i>, <i><false code></i> otherwise. The comparison is based on <code>\str_compare:....</code>. In the second version, the <i><tag name></i> is not provided and set to <code>__local</code>.</p>
--	---

```

207 \prg_new_conditional:Nnn \CDR_tag_if_eq:ccn { p, T, F, TF } {
208   \exp_args:Nf
209   \str_compare:nNnTF { \CDR_tag_get:cc { #1 } { #2 } } = { #3 } {
210     \prg_return_true:
211   } {
212     \prg_return_false:
213   }
214 }
215 \prg_new_conditional:Nnn \CDR_tag_if_eq:cn { p, T, F, TF } {

```

```

216 \exp_args:Nf
217 \str_compare:nNnTF { \CDR_tag_get:cc { __local } { #1 } } = { #2 } {
218   \prg_return_true:
219 } {
220   \prg_return_false:
221 }
222 }

```

`\CDR_if_truthy_p:n` ★ `\CDR_if_truthy:nTF` {*<token list>*} {*<true code>*} {*<false code>*}

`\CDR_if_truthy:nTF` ★ 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”.

```

223 \prg_new_conditional:Nnn \CDR_if_truthy:n { p, T, F, TF } {
224   \exp_args:Ne
225   \str_compare:nNnTF { \exp_args:Ne \str_lowercase:n { #1 } } = { true } {
226     \prg_return_true:
227   } {
228     \prg_return_false:
229   }
230 }

```

`\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.

```

231 \cs_new_protected:Npn \CDR_tag_boolean_set:n #1 {
232   \CDR_if_truthy:nTF { #1 } {
233     \CDR_tag_set:n { true }
234   } {
235     \CDR_tag_set:n { false }
236   }
237 }
238 \cs_generate_variant:Nn \CDR_tag_boolean_set:n { x }

```

6.3 Retrieving tag properties

Internally, all tag properties are collected with a full key path like `\c_CDR_tag_get/<tag name>/<relative key path>`. When typesetting some code with either the `\CDRCode` command or the `CDRBlock` environment, all properties defined locally are collected under the reserved `\c_CDR_tag_get/__local/<relative path>` full key paths. The `l3keys` module `\c_CDR_tag_get/__local` is modified in \TeX groups only. For running text code chunks, this module inherits from

1. `\c_CDR_tag_get/<tag name>` for the provided *<tag name>*,
2. `\c_CDR_tag_get/default.code`
3. `\c_CDR_tag_get/default`
4. `\c_CDR_tag_get/__pygments`
5. `\c_CDR_tag_get/__fancyvrb`

6. \c_CDR_tag_get/___fancyvrb.all when no using pygments

For text block code chunks, this module inherits from

1. \c_CDR_tag_get/⟨name₁⟩, ..., \c_CDR_tag_get/⟨name_n⟩ for each tag name of the ordered tags list
2. \c_CDR_tag_get/default.block
3. \c_CDR_tag_get/default
4. \c_CDR_tag_get/___pygments
5. \c_CDR_tag_get/___pygments.block
6. \c_CDR_tag_get/___fancyvrb
7. \c_CDR_tag_get/___fancyvrb.block
8. \c_CDR_tag_get/___fancyvrb.all when no using pygments

```
\CDR_tag_if_exist_here:p:cc * \CDR_tag_if_exist_here:ccTF {⟨tag name⟩} ⟨relative key path⟩ {⟨true
\CDR_tag_if_exist_here:ccTF * 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.

```
239 \prg_new_conditional:Nnn \CDR_tag_if_exist_here:cc { p, T, F, TF } {
240   \cs_if_exist:cTF { \CDR_tag_get_path:cc { #1 } { #2 } } {
241     \prg_return_true:
242   } {
243     \prg_return_false:
244   }
245 }
```

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

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`.

```
246 \prg_new_conditional:Nnn \CDR_tag_if_exist:cc { p, T, F, TF } {
247   \cs_if_exist:cTF { \CDR_tag_get_path:cc { #1 } { #2 } } {
248     \prg_return_true:
249   } {
250     \seq_if_exist:cTF { \CDR_tag_parent_seq:c { #1 } } {
251       \seq_map_tokens:cn
252         { \CDR_tag_parent_seq:c { #1 } }
253         { \CDR_tag_if_exist_f:cn { #2 } }
254     } {
255       \prg_return_false:
256     }
257 }
```



```

257 }
258 }
259 \prg_new_conditional:Nnn \CDR_tag_if_exist:c { p, T, F, TF } {
260   \cs_if_exist:cTF { \CDR_tag_get_path:c { #1 } } {
261     \prg_return_true:
262   } {
263     \seq_if_exist:cTF { \CDR_tag_parent_seq:c { __local } } {
264       \seq_map_tokens:cn
265         { \CDR_tag_parent_seq:c { __local } }
266         { \CDR_tag_if_exist_f:cn { #1 } }
267     } {
268       \prg_return_false:
269     }
270   }
271 }
272 \cs_new:Npn \CDR_tag_if_exist_f:cn #1 #2 {
273   \quark_if_no_value:nTF { #2 } {
274     \seq_map_break:n {
275       \prg_return_false:
276     }
277   } {
278     \CDR_tag_if_exist:ccT { #2 } { #1 } {
279       \seq_map_break:n {
280         \prg_return_true:
281       }
282     }
283   }
284 }

```

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

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

```

285 \cs_new:Npn \CDR_tag_get:cc #1 #2 {
286   \CDR_tag_if_exist_here:ccTF { #1 } { #2 } {
287     \use:c { \CDR_tag_get_path:cc { #1 } { #2 } }
288   } {
289     \seq_if_exist:cT { \CDR_tag_parent_seq:c { #1 } } {
290       \seq_map_tokens:cn
291         { \CDR_tag_parent_seq:c { #1 } }
292         { \CDR_tag_get_f:cn { #2 } }
293     }
294   }
295 }
296 \cs_new:Npn \CDR_tag_get_f:cn #1 #2 {
297   \quark_if_no_value:nF { #2 } {
298     \CDR_tag_if_exist_here:ccT { #2 } { #1 } {
299       \seq_map_break:n {
300         \use:c { \CDR_tag_get_path:cc { #2 } { #1 } }
301       }
302     }
303   }

```

```

304 }
305 \cs_new:Npn \CDR_tag_get:c {
306   \CDR_tag_get:cc { __local }
307 }

```

\CDR_tag_get:ccN	\CDR_tag_get:ccN {<tag name>} {<relative key path>} {<tl variable>}
\CDR_tag_get:cN	\CDR_tag_get:cN {<relative key path>} {<tl variable>}

Put in <tl variable> the property value stored for the __local <tag name> and <relative key path>. In the second version, the <tag name> is not provided an set to __local.

```

308 \cs_new_protected:Npn \CDR_tag_get:ccN #1 #2 #3 {
309   \tl_set:Nf #3 { \CDR_tag_get:cc { #1 } { #2 } }
310 }
311 \cs_new_protected:Npn \CDR_tag_get:cN {
312   \CDR_tag_get:ccN { __local }
313 }

```

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

Getter with branching. If the <relative key path> is known, save the value into <tl var> and execute <true code>. Otherwise, execute <false code>. In the second version, the <tag name> is not provided an set to __local.

```

314 \prg_new_protected_conditional:Nnn \CDR_tag_get:ccN { T, F, TF } {
315   \CDR_tag_if_exist:ccTF { #1 } { #2 } {
316     \CDR_tag_get:ccN { #1 } { #2 } #3
317     \prg_return_true:
318   } {
319     \prg_return_false:
320   }
321 }
322 \prg_new_protected_conditional:Nnn \CDR_tag_get:cN { T, F, TF } {
323   \CDR_tag_if_exist:cTF { #1 } {
324     \CDR_tag_get:cN { #1 } #2
325     \prg_return_true:
326   } {
327     \prg_return_false:
328   }
329 }

```

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 assigned locally.

```

330 \cs_new:Npn \CDR_tag_parent_seq:c #1 {
331   l_CDR:parent.tag @ #1 _seq
332 }

```

<pre> \CDR_get_inherit:cn \CDR_get_inherit:cf \CDR_get_inherit:n \CDR_get_inherit:f </pre>	<pre> \CDR_get_inherit:cn {{child name}} {{parent names comma list}} </pre> <p>Set the parents of $\langle child\ name \rangle$ to the given list. When the $\langle child\ name \rangle$ is not provided, it defaults to <code>__local</code>.</p>
--	---

```

333 \cs_new:Npn \CDR_get_inherit:cn #1 #2 {
334   \seq_set_from_clist:cn { \CDR_tag_parent_seq:c { #1 } } { #2 }
335   \seq_remove_duplicates:c \l_CDR_tl
336   \seq_remove_all:cn \l_CDR_tl {}
337   \seq_put_right:cn \l_CDR_tl { \q_no_value }
338 }
339 \cs_new:Npn \CDR_get_inherit:cf {
340   \exp_args:Nnf \CDR_get_inherit:cn
341 }
342 \cs_new:Npn \CDR_tag_parents:c #1 {
343   \seq_map_inline:cn { \CDR_tag_parent_seq:c { #1 } } {
344     \quark_if_no_value:nF { ##1 } {
345       ##1,
346     }
347   }
348 }
349 \cs_new:Npn \CDR_get_inherit:n {
350   \CDR_get_inherit:cn { __local }
351 }
352 \cs_new:Npn \CDR_get_inherit:f {
353   \CDR_get_inherit:cf { __local }
354 }

```

7 Cache management

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

```

355 \AddToHook { begindocument/before } {
356   \IfFileExists {./\jobname.aux} {} {
357     \lua_now:n {CDR:cache_clean_all()}
358   }
359 }

```

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

```

360 \AddToHook { enddocument/end } {
361   \lua_now:n {CDR:cache_clean_unused()}
362 }

```

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>.

```

363 \cs_new:Npn \CDR_clist_map_inline:Nnn #1 #2 {
364   \clist_if_empty:NTF #1 {
365     #2
366     \use_none:n
367   } {
368     \clist_map_inline:Nn #1
369   }
370 }
```

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

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

```

371 \prg_new_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
372   \PackageError
373     { coder }
374     { Conditional~not~available }
375     { Internal~error:~report~bug }
376 }
```

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

```

377 \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_module:n * \CDR_tag_module:n {<module base>}

The <module> is uniquely based on <module base>. This should be f expanded when used as n argument of l3keys functions.

```

378 \cs_set:Npn \CDR_tag_module:n #1 {
379   \str_if_eq:nnTF { #1 } { .. } {
380     \c_CDR_Tags
381   } {
382     \tl_if_empty:nTF { #1 } { \c_CDR_Tags / tag } { \c_CDR_Tags / tag / #1 }
383   }
384 }
```

\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.

```

385 \cs_new:Npn \CDR_tag_keys_define:nn #1 {
386   \exp_args:Nf
387   \keys_define:nn { \CDR_tag_module:n { #1 } }
388 }
```

\CDR_tag_keys_if_exist:nnTF * \CDR_tag_keys_if_exist:nnTF {<module base>} {<key>} {<true code>} {<false code>}

Execute <true code> if there is a <key> for the given <module base>, <false code> otherwise. If <module base> is empty, {<key>} is the module base used.

```

389 \prg_new_conditional:Nnn \CDR_tag_keys_if_exist:nn { p, T, F, TF } {
390   \exp_args:Nf
391   \keys_if_exist:nnTF { \CDR_tag_module:n { #1 } } { #2 } {
392     \prg_return_true:
393   } {
394     \prg_return_false:
395   }
396 }
```

\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.

```

397 \cs_new_protected:Npn \CDR_tag_keys_set:nn #1 {
398   \exp_args:Nf
399   \keys_set:nn { \CDR_tag_module:n { #1 } }
400 }
401 \cs_generate_variant:Nn \CDR_tag_keys_set:nn { nV }
```

```
\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*.

```
402 \cs_new_protected:Npn \CDR_local_set:n {
403   \CDR_tag_keys_set:nn { __local }
404 }
405 \cs_generate_variant:Nn \CDR_local_set:n { V }
```

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_tag_keys_inherit:nn \CDR_tag_keys_inherit:nn {<tag name>} {<parents comma list>}
```

Set the inheritance: *<tag name>* inherits from each parent, which is a tag name.

```
406 \cs_new_protected_nopar:Npn \CDR_tag_keys_inherit__:nnn #1 #2 #3 {
407   \keys_define:nn { #1 } { #2 .inherit:n = { #1 / #3 } }
408 }
409 \cs_new_protected_nopar:Npn \CDR_tag_keys_inherit_:nnn #1 #2 #3 {
410   \exp_args:Nnx
411   \use:n { \CDR_tag_keys_inherit__:nnn { #1 } { #2 } } {
412     \clist_use:nn { #3 } { ,#1/ }
413   }
414 }
415 \cs_new_protected_nopar:Npn \CDR_tag_keys_inherit:nn {
416   \exp_args:Nf
417   \CDR_tag_keys_inherit_:nnn { \CDR_tag_module:n { } }
418 }
```

```
\CDR_local_inherit:n Wrapper over \CDR_tag_keys_inherit:nn where <tag name> is
given by \CDR_tag_module:n{__local}.
```

Set the inheritance: *<tag name>* inherits from each parent, which is a tag name.

```
419 \cs_new_protected_nopar:Npn \CDR_local_inherit:n {
420   \CDR_tag_keys_inherit:nn { __local }
421 }
```

```
\CDR_tag_keys_set_known:nnN \CDR_tag_keys_set_known:nnN {<tag name>} {<key[=value] items>} <clist var>
\CDR_tag_keys_set_known:nVN \CDR_tag_keys_set_known:nN {<tag name>} <clist var>
\CDR_tag_keys_set_known:nN
\CDR_tag_keys_set_known:N
```

Wrappers over *\keys_set_known:nnnN* where the module is given by *\CDR_tag_module:n{<tag name>}*. *Implementation detail* the remaining arguments are absorbed by the last macro. When *<key[=value] items>* is omitted, it is the content of *<clist var>*.

```

422 \cs_new_protected_nopar:Npn \CDR_tag_keys_set_known__:nnN #1 #2 {
423   \keys_set_known:nnnN { #1 } { #2 } { #1 }
424 }
425 \cs_new_protected_nopar:Npn \CDR_tag_keys_set_known:nnN #1 {
426   \exp_args:Nf
427   \CDR_tag_keys_set_known__:nnN { \CDR_tag_module:n { #1 } }
428 }
429 \cs_generate_variant:Nn \CDR_tag_keys_set_known:nnN { nV }
430 \cs_new_protected_nopar:Npn \CDR_tag_keys_set_known:nN #1 #2 {
431   \CDR_tag_keys_set_known:nVN { #1 } #2 #2
432 }

```

```

\CDR_tag_keys_set_known:nnN   \CDR_local_set_known:nN {<key[=value] items>} <clist var>
\CDR_tag_keys_set_known:nVN   \CDR_local_set_known:N   <clist var>
\CDR_tag_keys_set_known:nN
\CDR_tag_keys_set_known:N

```

Wrappers over `\CDR_tag_keys_set_known:...` where the module is given by `\CDR_tag_module:n{__local}`. When `<key[=value] items>` is omitted, it is the content of `<clist var>`.

```

433 \cs_new_protected_nopar:Npn \CDR_local_set_known:nN {
434   \CDR_tag_keys_set_known:nnN { __local }
435 }
436 \cs_generate_variant:Nn \CDR_local_set_known:nN { V }
437 \cs_new_protected_nopar:Npn \CDR_local_set_known:N #1 {
438   \CDR_local_set_known:VN #1 #1
439 }

```

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

```

440 \tl_set:Nn \l_CDR_tl { /([~/*])(?:/(.))*?$ } \use_none:n { $ }
441 \exp_args:NNf
442 \tl_put_left:Nn \l_CDR_tl { \CDR_tag_module:n {} }
443 \tl_put_left:Nn \l_CDR_tl { ^ }
444 \exp_args:NNV
445 \regex_const:Nn \c_CDR_provide_regex \l_CDR_tl

```

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

```

\@CDR@TEST
\CDR_tag_provide_from_kv:n

```

```

\CDR_tag_provide:n {<deep comma list>}
\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 `/`. Implementation detail: uses `\l_CDR_tl`.

```

446 \regex_const:Nn \c_CDR_engine_regex { ^([~/*]+\sengine\soptions$ ) \use_none:n { $ }
447 \cs_new_protected_nopar:Npn \CDR_tag_provide:n #1 {
448   \CDR@Debug { \string\CDR_tag_provide:n: #1 }
449   \exp_args:NNf
450   \regex_extract_once:NnNTF \c_CDR_provide_regex {

```

```

451     \CDR_tag_module:n { .. } / #1
452 } \l_CDR_seq {
453     \tl_set:Nx \l_CDR_tl { \seq_item:Nn \l_CDR_seq 3 }
454     \exp_args:Nx
455     \clist_map_inline:nn {
456         \seq_item:Nn \l_CDR_seq 2
457     } {
458         \CDR_tag_keys_if_exist:nnF { } { ##1 } {
459             \CDR_tag_keys_inherit:nn { ##1 } {
460                 __pygments, __pygments.block,
461                 default.block, default.code, default, __tags, __engine,
462                 __fancyvrb, __fancyvrb.block, __fancyvrb.frame,
463                 __fancyvrb.number, __fancyvrb.all,
464             }
465             \CDR_tag_keys_define:nn { } {
466                 ##1 .code:n = \CDR_tag_keys_set:nn { ##1 } { #####1 },
467                 ##1 .value_required:n = true,
468             }
469 \CDR@Debug{\string\CDR_tag_provide:n \CDR_tag_module:n {##1} = ...}
470     }
471     \exp_args:NnV
472     \CDR_tag_keys_if_exist:nnF { ##1 } \l_CDR_tl {
473         \exp_args:NNV
474         \regex_match:NnT \c_CDR_engine_regex
475         \l_CDR_tl {
476             \exp_args:Nnf
477             \CDR_tag_keys_define:nn { ##1 } {
478                 \use:n { \l_CDR_tl } .code:n = \CDR_tag_set:n { #####1 },
479             }
480             \exp_args:Nnf
481             \CDR_tag_keys_define:nn { ##1 } {
482                 \use:n { \l_CDR_tl } .value_required:n = true,
483             }
484 \CDR@Debug{\string\CDR_tag_provide:n: \CDR_tag_module:n { ##1 } / \l_CDR_tl = ...}
485     }
486 }
487 }
488 } {
489     \regex_match:NnTF \c_CDR_engine_regex { #1 } {
490         \CDR_tag_keys_define:nn { default } {
491             #1 .code:n = \CDR_tag_set:n { ##1 },
492             #1 .value_required:n = true,
493         }
494 \CDR@Debug{\string\CDR_tag_provide:n:C:\CDR_tag_module:n { default } / #1 = ...}
495     } {
496 \CDR@Debug{\string\CDR_tag_provide:n\space did-nothing-new.}
497     }
498 }
499 }
500 \cs_new:Npn \CDR_tag_provide:nn #1 #2 {
501     \CDR_tag_provide:n { #1 }
502 }
503 \cs_new:Npn \CDR_tag_provide_from_kv:n {
504     \keyval_parse:nnn {

```



```

505 \CDR_tag_provide:n
506 } {
507 \CDR_tag_provide:nn
508 }
509 }
510 \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 not used.

9.2.1 Utilities

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

```

511 \sys_get_shell:nnN {which-pygmentize} {} \l_CDR_tl
512 \prg_new_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } { }
513 \tl_if_in:NnTF \l_CDR_tl { pygmentize } {
514 \prg_set_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } {
515 \prg_return_true:
516 }
517 } {
518 \prg_set_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } {
519 \prg_return_false:
520 }
521 }

```

9.2.2 __pygments l3keys module

```

522 \CDR_tag_keys_define:nn { __pygments } {

```

● **lang=<language name>** where `<language name>` is recognized by pygments, including a void string,

```

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

```

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

```

525 pygments .code:n = \CDR_tag_boolean_set:x { #1 },
526 pygments .default:n = true,

```

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

```

527 style .code:n = \CDR_tag_set:,
528 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`.

```
529 commandprefix .code:n = \CDR_tag_set:,
530 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`.

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

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

```
533 escapeinside .code:n = \CDR_tag_set:,
534 escapeinside .value_required:n = true,
```

🔴 **__initialize** Initializer.

```
535 __initialize .meta:n = {
536   lang = tex,
537   pygments = \CDR_has_pygments:TF { true } { false },
538   style = default,
539   commandprefix = PY,
540   mathescape = false,
541   escapeinside = ,
542 },
543 __initialize .value_forbidden:n = true,

544 }
545 \AtBeginDocument{
546   \CDR_tag_keys_set:nn { __pygments } { __initialize }
547 }
```

9.2.3 `__pygments.block l3keys` module

```
548 \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`.

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

🔴 **__initialize** Initializer.

```
551 __initialize .meta:n = {
552   texcomments = false,
553 },
554 __initialize .value_forbidden:n = true,
```

```

555 }
556 \AtBeginDocument{
557   \CDR_tag_keys_set:n { __pygments.block } { __initialize }
558 }

```

9.3 Specific to coder

9.3.1 default l3keys module

```

559 \CDR_tag_keys_define:n { 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.

```

560   format .code:n = \CDR_tag_set:,
561   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.

```

562   cache .code:n = \CDR_tag_boolean_set:x { #1 },
563   cache .default:n = true,

```

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

```

564   debug .code:n = \CDR_tag_boolean_set:x { #1 },
565   debug .default:n = true,

```

- **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.

```

566   post~processor .code:n = \CDR_tag_set:,
567   post~processor .value_required:n = true,

```

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

```

568   default~engine~options .code:n = \CDR_tag_set:,
569   default~engine~options .value_required:n = true,

```

- **default options**=*(default options)* to specify the coder options that should apply when the default engine is selected.setup_tags

```

570   default~options .code:n = \CDR_tag_set:,
571   default~options .value_required:n = true,

```

- *(engine name)* **engine options**=*(engine options)* to specify the options for the named engine,

● **<engine name> options=<coder options>** to specify the coder options that should apply when the named engine is selected.

● **__initialize** to initialize storage properly. We cannot use `.initial:n` actions because the `\l_keys_path_str` is not set up properly.

```

572 __initialize .meta:n = {
573     format = ,
574     cache = true,
575     debug = false,
576     post-processor = ,
577     default-engine-options = ,
578     default-options = ,
579 },
580 __initialize .value_forbidden:n = true,

581 }
582 \AtBeginDocument{
583     \CDR_tag_keys_set:nn { default } { __initialize }
584 }
```

9.3.2 default.code l3keys module

Void for the moment.

```

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

Known keys include:

● **__initialize** When set to `true`, put the argument inside a \LaTeX `mbox` to prevent the code chunk to spread over different lines. Initially `true`.

```

586     mbox .code:n = \CDR_tag_boolean_set:x { #1 },
587     mbox .default:n = true,
```

● **__initialize** to initialize storage properly. We cannot use `.initial:n` actions because the `\l_keys_path_str` is not set up properly.

```

588 __initialize .meta:n = {
589     mbox = true,
590 },
591 __initialize .value_forbidden:n = true,

592 }
593 \AtBeginDocument{
594     \CDR_tag_keys_set:nn { default.code } { __initialize }
595 }
```

9.3.3 __tags l3keys module

The only purpose is to catch only the `tags` key very early.

```

596 \CDR_tag_keys_define:nn { __tags } {
```

Known keys include:

🔴 **tags**=*<comma list of tag names>* to enable/disable the display of the code chunks tags. Initially empty.

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

```
597 tags .code:n = {
598   \clist_set:Nn \l_CDR_clist { #1 }
599   \clist_remove_duplicates:N \l_CDR_clist
600   \exp_args:NV
601   \CDR_tag_set:n \l_CDR_clist
602 },
603 tags .value_required:n = true,
```

🔴 **__initialize** Initialization.

```
604 __initialize .meta:n = {
605   tags = ,
606 },
607 __initialize .value_forbidden:n = true,

608 }
609 \AtBeginDocument{
610   \CDR_tag_keys_set:nn { __tags } { __initialize }
611 }
```

There is a companion module to catch unexpected **tags** key. Used for coder options when defining engines.

```
612 \CDR_tag_keys_define:nn { __no_tags } {
613   tags .code:n = {
614     \PackageError
615       { coder }
616       { Key~'tags'~is~forbidden~for~engines }
617       { See~the~coder~manual }
618   }
619 }
```

9.3.4 **__engine l3keys** module

The only purpose is to catch only the **engine** key very early, just after the **tags** key.

```
620 \CDR_tag_keys_define:nn { __engine } {
```

Known keys include:

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

```
621 engine .code:n = \CDR_tag_set:,
622 engine .value_required:n = true,
```

🔴 **__initialize** Initialization.

```

623   __initialize .meta:n = {
624     engine = default,
625   },
626   __initialize .value_forbidden:n = true,
627 }
628 \AtBeginDocument{
629   \CDR_tag_keys_set:nn { __engine } { __initialize }
630 }

```

There is a companion module to catch unexpected `tags` key. Used for `coder` options when defining engines.

```

631 \CDR_tag_keys_define:nn { __no_engine } {
632   engine .code:n = {
633     \PackageError
634       { coder }
635       { Key~'engine'~is~forbidden~for~engines }
636       { See~the~coder~manual }
637   }
638 }

```

9.3.5 default.block l3keys module

```

639 \CDR_tag_keys_define:nn { default.block } {

```

Known keys include:

- **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.

```

640   tags~format .code:n = \CDR_tag_set:,
641   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).

```

642   numbers~format .code:n = \CDR_tag_set:,
643   numbers~format .value_required:n = true,

```

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

```

644   show~tags .choices:nn =
645     { none, left, right, numbers, mirror }
646     { \CDR_tag_choices_set: },
647   show~tags .default:n = numbers,

```

- **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.

```

648   only~top .code:n = \CDR_tag_boolean_set:x { #1 },
649   only~top .default:n = true,

```

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

```
650 use~margin .code:n = \CDR_tag_boolean_set:x { #1 },
651 use~margin .default:n = true,
```

● **__initialize** Initialization.

```
652 __initialize .meta:n = {
653   show-tags = numbers,
654   only-top = true,
655   use~margin = true,
656   numbers~format = {
657     \sffamily
658     \scriptsize
659     \color{gray}
660   },
661   tags~format = {
662     \bfseries
663   },
664 },
665 __initialize .value_forbidden:n = true,

666 }
667 \AtBeginDocument{
668   \CDR_tag_keys_set:nn { default.block } { __initialize }
669 }
```

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

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

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

```
671 formatcom .code:n = \CDR_tag_set:,
672 formatcom .value_required:n = true,
```

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

```
673 fontfamily .code:n = \CDR_tag_set:,
674 fontfamily .value_required:n = true,
```

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

```
675 fontsize .code:n = \CDR_tag_set:,
676 fontsize .value_required:n = true,
```

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

```
677 fontshape .code:n = \CDR_tag_set:,
678 fontshape .value_required:n = true,
```

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

```
679 fontseries .code:n = \CDR_tag_set:,
680 fontseries .value_required:n = true,
```

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

```
681 showspaces .code:n = \CDR_tag_boolean_set:x { #1 },
682 showspaces .default:n = true,
```

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

```
683 showtabs .code:n = \CDR_tag_boolean_set:x { #1 },
684 showtabs .default:n = true,
```

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

```
685 obeytabs .code:n = \CDR_tag_boolean_set:x { #1 },
686 obeytabs .default:n = true,
```

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

```
687 tabsize .code:n = \CDR_tag_set:,
688 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.

```
689 defineactive .code:n = \CDR_tag_set:,
690 defineactive .value_required:n = true,
```

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

```
691 relabel .code:n = \CDR_tag_set:,
692 relabel .value_required:n = true,
```

✅ **__initialize** Initialization.


```

693 __initialize .meta:n = {
694     formatcom = ,
695     fontfamily = tt,
696     fontsize = auto,
697     fontseries = auto,
698     fontshape = auto,
699     showspace = false,
700     showtabs = false,
701     obeytabs = false,
702     tabsize = 2,
703     defineactive = ,
704     reflabel = ,
705 },
706 __initialize .value_forbidden:n = true,
707 }
708 \AtBeginDocument{
709     \CDR_tag_keys_set:nn { __fancyvrb } { __initialize }
710 }

```

9.4.2 __fancyvrb.frame l3keys module

Block specific options, frame related.

```

711 \CDR_tag_keys_define:nn { __fancyvrb.frame } {

```

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

```

712     frame .choices:nn =
713         { none, leftline, topline, bottomline, lines, single }
714         { \CDR_tag_choices_set: },

```

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

```

715     framerule .code:n = \CDR_tag_set:,
716     framerule .value_required:n = true,

```

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

```

717     framesep .code:n = \CDR_tag_set:,
718     framesep .value_required:n = true,

```

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

```

719     rulecolor .code:n = \CDR_tag_set:,
720     rulecolor .value_required:n = true,

```

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

```

721 fillcolor .code:n = \CDR_tag_set:,
722 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.

```

723 label .code:n = \CDR_tag_set:,
724 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.

```

725 labelposition .choices:nn =
726 { none, topline, bottomline, all }
727 { \CDR_tag_choices_set: },

```

- ✓ **__initialize** Initialization.

```

728 __initialize .meta:n = {
729   frame = none,
730   framerule = 0.4pt,
731   framesep = \fboxsep,
732   rulecolor = black,
733   fillcolor = ,
734   label = ,
735   labelposition = none,% auto?
736 },
737 __initialize .value_forbidden:n = true,

738 }
739 \AtBeginDocument{
740   \CDR_tag_keys_set:nn { __fancyvrb.frame } { __initialize }
741 }

```

9.4.3 __fancyvrb.block l3keys module

Block specific options, except numbering.

```

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

```

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

```

744 commentchar .code:n = \CDR_tag_set:,
745 commentchar .value_required:n = true,

```

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

```

746 gobble .choices:nn = {
747   0,1,2,3,4,5,6,7,8,9
748 } {
749   \CDR_tag_choices_set:
750 },

```

- **baselinestretch**=**auto**| $\langle dimension \rangle$ value to give to the usual $\backslash baselinestretch$ L^AT_EX parameter. Initially **auto**: its current value just before the verbatim command.

```

751 baselinestretch .code:n = \CDR_tag_set:,
752 baselinestretch .value_required:n = true,

```

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

- **xleftmargin**= $\langle dimension \rangle$ indentation to add at the start of each line. Initially **Opt**: no left margin.

```

753 xleftmargin .code:n = \CDR_tag_set:,
754 xleftmargin .value_required:n = true,

```

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

```

755 xrightmargin .code:n = \CDR_tag_set:,
756 xrightmargin .value_required:n = true,

```

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

```

757 resetmargins .code:n = \CDR_tag_boolean_set:x { #1 },
758 resetmargins .default:n = true,

```

- **hfuzz**= $\langle dimension \rangle$ value to give to the T_EX $\backslash hfuzz$ dimension for text to format. This can be used to avoid seeing some unimportant overfull box messages. Initially 2pt.

```

759 hfuzz .code:n = \CDR_tag_set:,
760 hfuzz .value_required:n = true,

```

- **vspace**= $\langle dimension \rangle$ the amount of vertical space added to $\backslash parskip$ before and after blocks. Initially $\backslash topsep$.

```

761 vspace .code:n = \CDR_tag_set:,
762 vspace .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**.

```

763 samepage .code:n = \CDR_tag_boolean_set:x { #1 },
764 samepage .default:n = true,

```

✓ **__initialize** Initialization.

```

765 __initialize .meta:n = {
766   commentchar = ,
767   gobble = 0,
768   baselinestretch = auto,
769   resetmargins = true,
770   xleftmargin = 0pt,
771   xrightmargin = 0pt,
772   hfuzz = 2pt,
773   vspace = \topset,
774   samepage = false,
775 },
776 __initialize .value_forbidden:n = true,

777 }
778 \AtBeginDocument{
779   \CDR_tag_keys_set:nn { __fancyvrb.block } { __initialize }
780 }

```

9.4.4 __fancyvrb.number l3keys module

Block line numbering.

```

781 \CDR_tag_keys_define:nn { __fancyvrb.number } {

```

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

```

782 numbers .choices:nn =
783   { none, left, right }
784   { \CDR_tag_choices_set: },

```

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

```

785 numbersep .code:n = \CDR_tag_set:,
786 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.

```


787 firstnumber .code:n = {
788   \regex_match:NnTF \c_CDR_integer_regex { #1 } {
789     \CDR_tag_set:
790   } {
791     \str_case:nnF { #1 } {
792       { auto } { \CDR_tag_set: }
793       { last } { \CDR_tag_set: }

```

```

794     } {
795         \PackageWarning
796         { CDR }
797         { Value~'#1'~not~in~auto,~last. }
798     }
799 }
800 },
801 firstnumber .value_required:n = true,


```

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

```

802 stepnumber .code:n = \CDR_tag_set:,
803 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.

```

804 numberblanklines .code:n = \CDR_tag_boolean_set:x { #1 },
805 numberblanklines .default:n = true,


```

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

```

806 firstline .code:n = \CDR_tag_set:,
807 firstline .value_required:n = true,


```

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

```

808 lastline .code:n = \CDR_tag_set:,
809 lastline .value_required:n = true,

```

 **__initialize** Initialization.

```

810 __initialize .meta:n = {
811     numbers = left,
812     numbersep = 1ex,
813     firstnumber = auto,
814     stepnumber = 1,
815     numberblanklines = true,
816     firstline = ,
817     lastline = ,
818 },
819 __initialize .value_forbidden:n = true,

820 }
821 \AtBeginDocument{
822     \CDR_tag_keys_set:nn { __fancyvrb.number } { __initialize }
823 }

```

9.4.5 `__fancyvrb.all` `l3keys` module

Options available when `pygments` is not used.

```
824 \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.

```
825   commandchars .code:n = \CDR_tag_set:,
826   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.

```
827   codes .code:n = \CDR_tag_set:,
828   codes .value_required:n = true,
```

- ✅ **__initialize** Initialization.

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

834 }
835 \AtBeginDocument{
836   \CDR_tag_keys_set:nn { __fancyvrb.all } { __initialize }
837 }
```

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


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

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

```

839 only~description .choices:nn = { false, true, {} } {
840   \int_compare:nNnTF \l_keys_choice_int = 1 {
841     \prg_set_conditional:Nnn \CDR_if_only_description: { p, T, F, TF } { \prg_return_true: }
842   } {
843     \prg_set_conditional:Nnn \CDR_if_only_description: { p, T, F, TF } { \prg_return_false: }
844   }
845 },
846 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.

```

847 python-path .code:n = {
848   \str_set:Nn \l_CDR_str { #1 }
849   \lua_now:n { CDR:set_python_path('l_CDR_str') }
850 },
851 }

```

10.2 Branching

```

\CDR_if_only_description_p: * \CDR_if_only_description:TF {<true code>} {<false code>}
\CDR_if_only_description:TF *

```

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

10.3 Implementation

```

\CDRBlock_preflight:n \CDR_set_preflight:n {<CDR@Set kv list>}

```

This is a preflight hook intended for testing. The default implementation does nothing.

```

852 \cs_new:Npn \CDR_set_preflight:n #1 { }

853 \NewDocumentCommand \CDRSet { m } {
854   \CDR@Debug{\string\CDRSet}
855   \CDR_set_preflight:n { #1 }
856   \keys_set_known:nnn { CDR@Set } { #1 } { CDR@Set } \l_CDR_kv_clist
857   \clist_map_inline:nn {
858     __pygments, __pygments.block,
859     __tags, __engine, default.block, default.code, default,
860     __fancyvrb, __fancyvrb.frame, __fancyvrb.block, __fancyvrb.number, __fancyvrb.all
861   } {
862     \CDR_tag_keys_set_known:nN { ##1 } \l_CDR_kv_clist
863   }
864   \CDR@Debug{ Debug.CDRSet.1:##1/\l_CDR_kv_clist/ }
865   \CDR_tag_keys_set_known:nN { .. } \l_CDR_kv_clist
866   \CDR@Debug{ Debug.CDRSet.2:\CDR_tag_module:n { .. }//\l_CDR_kv_clist/ }
867   \CDR_tag_provide_from_kv:V \l_CDR_kv_clist
868   \CDR@Debug{ Debug.CDRSet.2a:\CDR_tag_module:n { .. }//\l_CDR_kv_clist/ }

```

```

869 \CDR_tag_keys_set_known:nN { .. } \l_CDR_kv_clist
870 \CDR@Debug{ Debug.CDRSet.3:\CDR_tag_module:n { .. }//\l_CDR_kv_clist/ }
871 \CDR_tag_keys_set:nV { default } \l_CDR_kv_clist
872 \CDR@Debug{ Debug.CDRSet.4:\CDR_tag_module:n { default } /\l_CDR_kv_clist/ }
873 \keys_define:nn { CDR@Set@tags } {
874   tags .code:n = {
875     \clist_set:Nn \g_CDR_tags_clist { ##1 }
876     \clist_remove_duplicates:N \g_CDR_tags_clist
877   },
878 }
879 \keys_set_known:nn { CDR@Set@tags } { #1 }
880 \ignorespaces
881 }

```

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.

```

882 \cs_new:Npn \CDR_export_get_path:cc #1 #2 {
883   CDR @ export @ get @ #1 / #2
884 }

```

\CDR_export_set:ccn \CDR_export_set:ccn {<file name>} {<relative key path>} {<value>}

\CDR_export_set:Vcn Store <value>, which is further retrieved with the instruction \CDR_get_get:cc {<file name>} {<relative key path>}. All the affectations are made at the current T_EX group level.

\CDR_export_set:VcV

```

885 \cs_new_protected:Npn \CDR_export_set:ccn #1 #2 #3 {
886   \cs_set:cpn { \CDR_export_get_path:cc { #1 } { #2 } } { \exp_not:n { #3 } }
887 }
888 \cs_new_protected:Npn \CDR_export_set:Vcn #1 {
889   \exp_args:NV
890   \CDR_export_set:ccn { #1 }
891 }
892 \cs_new_protected:Npn \CDR_export_set:VcV #1 #2 #3 {
893   \exp_args:NnV
894   \use:n {
895     \exp_args:NV \CDR_export_set:ccn #1 { #2 }
896   } #3
897 }

```

```
\CDR_export_if_exist:ccTF * \CDR_export_if_exist:ccTF {<file name>} <relative key path> {<true code>}
{<false code>}
```

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

```
898 \prg_new_conditional:Nnn \CDR_export_if_exist:cc { p, T, F, TF } {
899   \cs_if_exist:cTF { \CDR_export_get_path:cc { #1 } { #2 } } {
900     \prg_return_true:
901   } {
902     \prg_return_false:
903   }
904 }
```

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

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

```
905 \cs_new:Npn \CDR_export_get:cc #1 #2 {
906   \CDR_export_if_exist:ccT { #1 } { #2 } {
907     \use:c { \CDR_export_get_path:cc { #1 } { #2 } }
908   }
909 }
```

```
\CDR_export_get:ccNTF \CDR_export_get:ccNTF {<file name>} {<relative key path>}
<t1 var> {<true code>} {<false code>}
```

Get the property value stored for <file name> and <relative key path>, copy it to <t1 var>. Execute <true code> on success, <false code> otherwise.

```
910 \prg_new_protected_conditional:Nnn \CDR_export_get:ccN { T, F, TF } {
911   \CDR_export_if_exist:ccTF { #1 } { #2 } {
912     \tl_set:Nx #3 { \CDR_export_get:cc { #1 } { #2 } }
913     \prg_return_true:
914   } {
915     \prg_return_false:
916   }
917 }
```

11.2 Storage

\g_CDR_export_seq Global list of all the files to be exported.

```
918 \seq_new:N \g_CDR_export_seq
```

(End definition for \g_CDR_export_seq. 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.

```
919 \tl_new:N \l_CDR_file_tl
```

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

\l_CDR_export_prop Used by CDR@Export l3keys module to temporarily store properties.

```
920 \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.

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

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

```
922   file .tl_set:N = \l_CDR_file_tl,
923   file .value_required:n = true,
```

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

```
924   tags .code:n = {
925     \clist_set:Nn \l_CDR_clist { #1 }
926     \clist_remove_duplicates:N \l_CDR_clist
927     \prop_put:NVV \l_CDR_export_prop \l_keys_key_str \l_CDR_clist
928   },
929   tags .value_required:n = true,
```

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

```
930   lang .code:n = {
931     \prop_put:NVn \l_CDR_export_prop \l_keys_key_str { #1 }
932   },
933   lang .value_required:n = true,
```

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

```
934   preamble .code:n = {
935     \prop_put:NVn \l_CDR_export_prop \l_keys_key_str { #1 }
936   },
937   preamble .value_required:n = true,
```

● **postamble** the added postamble. Initially empty.

```
938   postamble .code:n = {
939     \prop_put:NVn \l_CDR_export_prop \l_keys_key_str { #1 }
940   },
941   postamble .value_required:n = true,
```

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

```
942   raw .choices:nn = { false, true, {} } {
943     \prop_put:NVx \l_CDR_export_prop \l_keys_key_str {
944       \int_compare:nNnTF
945         \l_keys_choice_int = 1 { false } { true }
946     }
947   },
```

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

```

948   once .choices:nn = { false, true, {} } {
949     \prop_put:NVx \l_CDR_export_prop \l_keys_key_str {
950       \int_compare:nNnTF
951         \l_keys_choice_int = 1 { false } { true }
952     }
953 },

```

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

```

954   __initialize .meta:n = {
955     __initialize_prop = #1,
956     file =,
957     tags =,
958     lang = tex,
959     preamble =,
960     postamble =,
961     raw = false,
962     once = true,
963   },
964   __initialize .default:n = \l_CDR_export_prop,

```

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

```

965   __initialize_prop .code:n = \prop_clear:N #1,
966   __initialize_prop .value_required:n = true,
967 }

```

11.4 Implementation

```

968 \NewDocumentCommand \CDRExport { m } {
969   \keys_set:nn { CDR@Export } { __initialize }
970   \keys_set:nn { CDR@Export } { #1 }
971   \tl_if_empty:NTF \l_CDR_file_tl {
972     \PackageWarning
973       { coder }
974       { Missing~export~key~‘file’ }
975   } {
976     \CDR_export_set:VcV \l_CDR_file_tl { file } \l_CDR_file_tl
977     \prop_map_inline:Nn \l_CDR_export_prop {
978       \CDR_export_set:Vcn \l_CDR_file_tl { ##1 } { ##2 }
979     }

```

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.

```

980     \prop_get:NnNTF \l_CDR_export_prop { tags } \l_CDR_clist {
981       \tl_if_empty:NTF \l_CDR_clist {
982         \PackageWarning
983           { coder }
984           { Missing~export~key~‘tags’ }

```

```

985     } {
986       \clist_set_eq:NN \g_CDR_tags_clist \l_CDR_clist
987       \clist_remove_duplicates:N \g_CDR_tags_clist
988       \clist_put_left:NV \g_CDR_all_tags_clist \l_CDR_clist
989       \clist_remove_duplicates:N \g_CDR_all_tags_clist

```

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

```

990     \exp_args:NV
991     \CDR_export_get:ccNT \l_CDR_file_tl { lang } \l_CDR_tl {
992       \clist_map_inline:Nn \g_CDR_tags_clist {
993         \CDR_tag_set:ccV { ##1 } { lang } \l_CDR_tl
994       }
995     }
996   }
997   \seq_put_left:NV \g_CDR_export_seq \l_CDR_file_tl
998 } {
999   \PackageWarning
1000   { coder }
1001   { Missing~export~key~‘tags’ }
1002 }
1003 }
1004 \ignorespaces
1005 }

```

Files are created at the end of the typesetting process.

```

1006 \AddToHook { enddocument / end } {
1007   \seq_map_inline:Nn \g_CDR_export_seq {
1008     \str_set:Nx \l_CDR_str { #1 }
1009     \lua_now:n { CDR:export_file('l_CDR_str') }
1010     \clist_map_inline:nn {
1011       tags, raw, once, preamble, postamble
1012     } {
1013       \CDR_export_get:ccNT { #1 } { ##1 } \l_CDR_tl {
1014         \exp_args:NNx
1015         \str_set:Nn \l_CDR_str { \l_CDR_tl }
1016         \lua_now:n {
1017           CDR:export_file_info('##1', 'l_CDR_str')
1018         }
1019       }
1020     }
1021     \lua_now:n { CDR:export_complete() }
1022   }
1023 }

```

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>`.

```

1024 \cs_set:Npn \CDR@StyleDefine #1 {
1025   \tl_gset:cn { g_CDR@Style/#1 }
1026 }

```

\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.

```

1027 \cs_set:Npn \CDR@StyleUse #1 {
1028   \tl_use:c { g_CDR@Style/#1 }
1029 }
1030 \cs_set:Npn \CDR@StyleUseTag {
1031   \CDR@StyleUse { \CDR_tag_get:c { style } }
1032 }

```

\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.

```

1033 \prg_new_conditional:Nnn \CDR@StyleIfExist:c { TF } {
1034   \tl_if_exist:cTF { g_CDR@Style/#1 } {
1035     \prg_return_true:
1036   } {
1037     \prg_return_false:
1038   }
1039 }
1040 \cs_set_eq:NN \CDR@StyleIfExist \CDR@StyleIfExist:cTF

```

13 Creating display engines

13.1 Utilities

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

```

1041 \cs_new:Npn \CDRCode_engine:c #1 {
1042   CDR@colored/code/#1:nn
1043 }
1044 \cs_new:Npn \CDRBlock_engine:c #1 {
1045   CDR@colored/block/#1
1046 }
1047 \cs_new:Npn \CDRCode_engine:V {
1048   \exp_args:NV \CDRCode_engine:c
1049 }
1050 \cs_new:Npn \CDRBlock_engine:V {
1051   \exp_args:NV \CDRBlock_engine:c
1052 }

```

\CDRCode_options:c	★	\CDRCode_options:c {<engine name>}
\CDRCode_options:V	★	\CDRBlock_options:c {<engine name>}
\CDRBlock_options:c	★	\CDRCode_options:c builds a command sequence name based on <engine name> used
\CDRBlock_options:V	★	to store the comma list of key value options. \CDRBlock_options:c builds a command

sequence name based on <engine name> used to store the comma list of key value options.

```

1053 \cs_new:Npn \CDRCode_options:c #1 {
1054   CDR@colored/code~options/#1:nn
1055 }
1056 \cs_new:Npn \CDRBlock_options:c #1 {
1057   CDR@colored/block~options/#1
1058 }
1059 \cs_new:Npn \CDRCode_options:V {
1060   \exp_args:NV \CDRCode_options:c
1061 }
1062 \cs_new:Npn \CDRBlock_options:V {
1063   \exp_args:NV \CDRBlock_options:c
1064 }

```

\CDRCode_options_use:c	★	\CDRCode_options_use:c {<engine name>}
\CDRCode_options_use:V	★	\CDRBlock_options_use:c {<engine name>}
\CDRBlock_options_use:c	★	\CDRCode_options_use:c builds a command sequence name based on <engine name>
\CDRBlock_options_use:V	★	and use it. \CDRBlock_options:c builds a command sequence name based on <engine

name> and use it.

```

1065 \cs_new:Npn \CDRCode_options_use:c #1 {
1066   \CDRCode_if_options:cT { #1 } {
1067     \use:c { \CDRCode_options:c { #1 } }
1068   }
1069 }
1070 \cs_new:Npn \CDRBlock_options_use:c #1 {
1071   \CDRBlock_if_options:cT { #1 } {
1072     \use:c { \CDRBlock_options:c { #1 } }
1073   }
1074 }
1075 \cs_new:Npn \CDRCode_options_use:V {
1076   \exp_args:NV \CDRCode_options_use:c
1077 }
1078 \cs_new:Npn \CDRBlock_options_use:V {
1079   \exp_args:NV \CDRBlock_options_use:c
1080 }

```

\l_CDR_engine_tl Storage for an engine name.

```

1081 \tl_new:N \l_CDR_engine_tl

```

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

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

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

13.2 Implementation

<code>\CDRCodeEngineNew</code>	<code>\CDRCodeEngineNew {<engine name>}{<engine body>}</code>
<code>\CDRCodeEngineRenew</code>	<code>\CDRCodeEngineRenew{<engine name>}{<engine body>}</code>

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

```

1082 \NewDocumentCommand \CDRCodeEngineNew { mO{}m } {
1083   \exp_args:Nx
1084   \tl_if_empty:nTF { #1 } {
1085     \PackageWarning
1086       { coder }
1087       { The~engine~cannot~be~void. }
1088   } {
1089     \CDR_forbidden_keys:n { #2 }
1090     \cs_set:cpn { \CDRCode_options:c { #1 } } { \exp_not:n { #2 } }
1091     \cs_new:cpn { \CDRCode_engine:c {#1} } ##1 ##2 {
1092       \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
1093       #3
1094     }
1095     \ignorespaces
1096   }
1097 }
```

<code>\CDR_forbidden_keys:n</code>	<code>\CDR_forbidden_keys:n {<key[=value] items>}</code>
------------------------------------	--

Raise an error if one of tags and engine keys is provided in `<key[=value] items>`. These keys are forbidden for the coder options associate to an engine.

```

1098 \cs_new:Npn \CDR_forbidden_keys:n #1 {
1099   \group_begin:
1100   \CDR_local_inherit:n { __no_tags, __no_engine }
1101   \CDR_local_set_known:nN { #1 } \l_CDR_kv_clist
1102   \group_end:
1103 }

1104 \NewDocumentCommand \CDRCodeEngineRenew { mO{}m } {
1105   \exp_args:Nx
1106   \tl_if_empty:nTF { #1 } {
1107     \PackageWarning
1108       { coder }
1109       { The~engine~cannot~be~void. }
1110     \use_none:n
1111   } {
1112     \cs_if_exist:cTF { \CDRCode_engine:c { #1 } } {
1113       \CDR_forbidden_keys:n { #2 }
1114       \cs_set:cpn { \CDRCode_options:c { #1 } } { \exp_not:n { #2 } }
1115       \cs_set:cpn { \CDRCode_engine:c { #1 } } ##1 ##2 {
1116         \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
1117         #3

```

```

1118     }
1119   } {
1120     \PackageWarning
1121       { coder }
1122       { No~code~engine~#1.}
1123   }
1124   \ignorespaces
1125 }
1126 }

```

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

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

```

1127 \cs_new_protected:Npn \CDR@CodeEngineApply {
1128   \CDRCode_if_engine:cF { \CDR_tag_get:c { engine } } {
1129     \PackageError
1130       { coder }
1131       { \CDR_tag_get:c { engine }~code~engine~unknown,~replaced-by~‘default’ }
1132       { See~\CDRCodeEngineNew~in~the~coder~manual }
1133     \CDR_tag_set:cn { engine } { default }
1134   }
1135   \CDR_tag_get:c { format }
1136   \exp_args:Nnx
1137   \use:c { \CDRCode_engine:c { \CDR_tag_get:c { engine } } } {
1138     \CDR_tag_get:c { \CDR_tag_get:c { engine }~engine~options },
1139     \CDR_tag_get:c { engine~options }
1140   }
1141 }

```

\CDRBlockEngineNew \CDRBlockEngineNew {<engine name>} [<options>] {<begin instructions>} {<end instructions>}

\CDRBlockEngineRenew \CDRBlockEngineRenew {<engine name>} [<options>] {<begin instructions>} {<end instructions>}

Create a L^AT_EX environment uniquely named after <engine name>, which must be a non void string once expanded. The <begin instructions> and <end instructions> are lists of instructions which may refer to the name 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 fourth argument is parsed by \NewDocumentEnvironment.

```

1142 \NewDocumentCommand \CDRBlockEngineNew { mO{}m } {
1143   \CDR_forbidden_keys:n { #2 }
1144   \cs_set:cpn { \CDRBlock_options:c { #1 } } { \exp_not:n { #2 } }
1145   \NewDocumentEnvironment { \CDRBlock_engine:c { #1 } } { m } {
1146     \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
1147     #3
1148   }
1149 }

```



```

1150 \NewDocumentCommand \CDRBlockEngineRenew { mO{}m } {
1151   \tl_if_empty:nTF { #1 } {
1152     \PackageError
1153       { coder }
1154       { The~engine~cannot~be~void. }
1155       { See~\string\CDRBlockEngineNew~in~the~coder~manual }
1156     \use_none:n
1157   } {
1158     \cs_if_exist:cTF { \CDRBlock_engine:c { #1 } } {
1159       \CDR_forbidden_keys:n { #2 }
1160       \cs_set:cpn { \CDRBlock_options:c { #1 } } { \exp_not:n { #2 } }
1161       \RenewDocumentEnvironment { \CDRBlock_engine:c { #1 } } { m } {
1162         \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
1163         #3
1164       }
1165     } {
1166       \PackageError
1167         { coder }
1168         { No~block~engine~#1.}
1169         { See~\string\CDRBlockEngineNew~in~the~coder~manual }
1170     }
1171   }
1172 }

```

\CDRBlock_engine_begin:	\CDRBlock_engine_begin:
\CDR@Block_engine_end:	\CDRBlock_engine_end:

After some checking, begin the engine display environment with the proper options. The second command closes the environment. This does not start a new group.

```

1173 \cs_new:Npn \CDRBlock_engine_begin: {
1174   \CDRBlock_if_engine:cF { \CDR_tag_get:c { engine } } {
1175     \PackageError
1176       { coder }
1177       { \CDR_tag_get:c { engine }~block~engine~unknown,~replaced~by~‘default’ }
1178       {See~\CDRBlockEngineNew~in~the~coder~manual}
1179     \CDR_tag_set:cn { engine } { default }
1180   }
1181   \exp_args:Nnx
1182   \use:c { \CDRBlock_engine:c \CDR_tag_get:c { engine } } {
1183     \CDR_tag_get:c { \CDR_tag_get:c { engine }~engine~options },
1184     \CDR_tag_get:c { engine~options },
1185   }
1186 }
1187 \cs_new:Npn \CDRBlock_engine_end: {
1188   \use:c { end \CDRBlock_engine:c \CDR_tag_get:c { engine } }
1189 }
1190 % \begin{MacroCode}
1191 %
1192 % \subsection{Conditionals}
1193 %
1194 % \begin{function}[EXP,TF]{\CDRCode_if_engine:c}
1195 % \begin{syntax}
1196 % \cs{CDRCode_if_engine:cTF} \Arg{engine name} \Arg{true code} \Arg{false code}

```

```

1197 % \end{syntax}
1198 % If there exists a code engine with the given \metatt{engine name},
1199 % execute \metatt{true code}.
1200 % Otherwise, execute \metatt{false code}.
1201 % \end{function}
1202 % \begin{MacroCode}[OK]
1203 \prg_new_conditional:Nnn \CDRCode_if_engine:c { p, T, F, TF } {
1204   \cs_if_exist:cTF { \CDRCode_engine:c { #1 } } {
1205     \prg_return_true:
1206   } {
1207     \prg_return_false:
1208   }
1209 }
1210 \prg_new_conditional:Nnn \CDRCode_if_engine:V { p, T, F, TF } {
1211   \cs_if_exist:cTF { \CDRCode_engine:V #1 } {
1212     \prg_return_true:
1213   } {
1214     \prg_return_false:
1215   }
1216 }

```

\CDRBlock_if_engine:cTF ★ \CDRBlock_if_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>.

```

1217 \prg_new_conditional:Nnn \CDRBlock_if_engine:c { p, T, F, TF } {
1218   \cs_if_exist:cTF { \CDRBlock_engine:c { #1 } } {
1219     \prg_return_true:
1220   } {
1221     \prg_return_false:
1222   }
1223 }
1224 \prg_new_conditional:Nnn \CDRBlock_if_engine:V { p, T, F, TF } {
1225   \cs_if_exist:cTF { \CDRBlock_engine:V #1 } {
1226     \prg_return_true:
1227   } {
1228     \prg_return_false:
1229   }
1230 }

```

\CDRCode_if_options:cTF ★ \CDRCode_if_options:cTF {<engine name>} {<true code>} {<false code>}

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

```

1231 \prg_new_conditional:Nnn \CDRCode_if_options:c { p, T, F, TF } {
1232   \cs_if_exist:cTF { \CDRCode_options:c { #1 } } {
1233     \prg_return_true:
1234   } {
1235     \prg_return_false:
1236   }
1237 }

```

```

1238 \prg_new_conditional:Nnn \CDRCode_if_options:V { p, T, F, TF } {
1239   \cs_if_exist:cTF { \CDRCode_options:V #1 } {
1240     \prg_return_true:
1241   } {
1242     \prg_return_false:
1243   }
1244 }

```

```

\CDRBlock_if_options:cTF ★ \CDRBlock_if_options:c {⟨engine name⟩} {⟨true code⟩} {⟨false code⟩}

```

If there exists a block options with the given *⟨engine name⟩*, execute *⟨true code⟩*, otherwise, execute *⟨false code⟩*.

```

1245 \prg_new_conditional:Nnn \CDRBlock_if_options:c { p, T, F, TF } {
1246   \cs_if_exist:cTF { \CDRBlock_options:c { #1 } } {
1247     \prg_return_true:
1248   } {
1249     \prg_return_false:
1250   }
1251 }
1252 \prg_new_conditional:Nnn \CDRBlock_if_options:V { p, T, F, TF } {
1253   \cs_if_exist:cTF { \CDRBlock_options:V #1 } {
1254     \prg_return_true:
1255   } {
1256     \prg_return_false:
1257   }
1258 }

```

13.3 Default code engine

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

```

1259 \CDRCodeEngineNew { default } { #2 }

```

13.4 efbox code engine

```

1260 \AtBeginDocument {
1261   \@ifpackageloaded{efbox} {
1262     \CDRCodeEngineNew {efbox} {
1263       \efbox[#1]{#2}
1264     }
1265   } {}
1266 }

```

13.5 Block mode default engine

```

1267 \CDRBlockEngineNew {default} {
1268 } {
1269 }

```

13.6 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.

```
1270 \cs_new:Npn \CDR@DefinePygSp {
1271   \CDR_tag_if_truthy:cTF { showspaces } {
1272     \cs_set:Npn \CDR@Sp {{\FancyVerbSpace}}
1273   } {
1274     \cs_set_eq:NN \CDR@Sp \space
1275   }
1276 }
```

\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.


```
1277 \tl_new:N \l_CDR_tag_tl
```

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


14.3 __code l3keys module

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


```
1278 \CDR_tag_keys_define:nn { __code } {
```

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

```
1279   tag .tl_set:N = \l_CDR_tag_tl,
1280   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.

```
1281   engine~options .code:n = \CDR_tag_set:,
1282   engine~options .value_required:n = true,
```

 **__initialize** initialize

```
1283   __initialize .meta:n = {
1284     tag = default,
1285     engine~options = ,
1286   },
1287   __initialize .value_forbidden:n = true,
1288 }
```

14.4 Implementation

`\CDRCodeformat:` `\CDRCodeformat:`

Private utility to setup the formatting.

```

1289 \cs_new:Npn \CDR_brace_if_contains_comma:n #1 {
1290   \tl_if_in:nnTF { #1 } { , } { { #1 } } { #1 }
1291 }
1292 \cs_generate_variant:Nn \CDR_brace_if_contains_comma:n { V }
1293 \cs_new:Npn \CDRCodeformat: {
1294   \frenchspacing
1295   \CDR_tag_get:cN { baselinestretch } \l_CDR_tl
1296   \str_if_eq:VnF \l_CDR_tl { auto } {
1297     \exp_args:NNV
1298     \def \baselinestretch \l_CDR_tl
1299   }
1300   \CDR_tag_get:cN { fontfamily } \l_CDR_tl
1301   \str_if_eq:VnT \l_CDR_tl { tt } { \tl_set:Nn \l_CDR_tl { lmtt } }
1302   \exp_args:NV
1303   \fontfamily \l_CDR_tl
1304   \clist_map_inline:nn { series, shape } {
1305     \CDR_tag_get:cN { font##1 } \l_CDR_tl
1306     \str_if_eq:VnF \l_CDR_tl { auto } {
1307       \exp_args:NnV
1308       \use:c { font##1 } \l_CDR_tl
1309     }
1310   }
1311   \CDR_tag_get:cN { fontsize } \l_CDR_tl
1312   \str_if_eq:VnF \l_CDR_tl { auto } {
1313     \tl_use:N \l_CDR_tl
1314   }
1315   \selectfont
1316   % \@noligs ?? this is in fancyvrb but does not work here as is
1317 }

1318 \NewDocumentCommand \CDRCode { O{} } {
1319   \group_begin:
1320   \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
1321     \prg_return_false:
1322   }
1323   \clist_set:Nn \l_CDR_kv_clist { #1 }
1324   \CDRCode_setup_tags_and_engine:N \l_CDR_kv_clist
1325   \CDR_local_inherit:n {
1326     __code, default.code, __pygments, default,
1327   }
1328   \CDR_local_set_known:N \l_CDR_kv_clist
1329   \CDR_tag_provide_from_kv:V \l_CDR_kv_clist
1330   \CDR_local_set_known:N \l_CDR_kv_clist
1331   \CDR_local_inherit:n {
1332     __fancyvrb,
1333   }
1334   \CDR_local_set:V \l_CDR_kv_clist
1335   \CDRCode:n
1336 }

```

\CDRBlock_setup_tags_and_engine:

Utility to setup the tags and the tag inheritance tree. When not provided explicitly with the `tags=...` user interface, a code chunk will have the list of tags stored in `\g_CDR_tags_clist` by last `\CDRExport`, `\CDRSet` or `\CDRBlock` environment. At least one tag must be provided, either implicitly or explicitly.

```
1337 \cs_new_protected_nopar:Npn \CDRCode_setup_tags_and_engine:N #1 {
1338   \CDR_local_inherit:n { __tags }
1339   \CDR_local_set_known:N #1
1340   \CDR_tag_if_exist_here:ccT { __local } { tags } {
1341     \CDR_tag_get:cN { tags } \l_CDR_clist
1342     \clist_if_empty:NF \l_CDR_clist {
1343       \clist_gset_eq:NN \g_CDR_tags_clist \l_CDR_clist
1344     }
1345   }
1346   \clist_if_empty:NT \g_CDR_tags_clist {
1347     \PackageWarning
1348       { coder }
1349     { No~(default)~tags~provided. }
1350   }
1351   \CDR@Debug {CDRCode_setup_tags_and_engine:\space\g_CDR_tags_clist}
```

Setup the inheritance tree for the `\CDR_tag_get:...` related functions.

```
1352   \CDR_get_inherit:f {
1353     \g_CDR_tags_clist,
1354     __tags, __engine, __code, default.code, __pygments, default,
1355   }
```

Now setup the engine options if any.

```
1356   \CDR_local_inherit:n { __engine }
1357   \CDR_local_set_known:N #1
1358   \CDR_tag_get:cNT { engine } \l_CDR_tl {
1359     \clist_put_left:Nx #1 { \CDRCode_options_use:V \l_CDR_tl }
1360   }
1361 }
```

\CDRCode:n `\CDRCode:n <delimiter>`

Main utility used by `\CDRCode`. The main tricky part is that we must collect the `<key[=value]>` items and feed `\FV@KeyValues` with them in the `aftersave` handler.

```
1362 \cs_new_protected_nopar:Npn \CDRCode:n #1 {
1363   \CDR_tag_if_truthy:cTF {pygments} {
1364     \cs_set:Npn \CDR@StyleUseTag {
1365       \CDR@StyleUse { \CDR_tag_get:c { style } }
1366       \cs_set_eq:NN \CDR@StyleUseTag \prg_do_nothing:
1367     }
1368     \DefineShortVerb { #1 }
1369     \SaveVerb [
1370       aftersave = {
1371         \exp_args:Nx \UndefineShortVerb { #1 }
1372         \lua_now:n { CDR:highlight_code_setup() }

```

```

1373 \CDR_tag_get:cN {lang} \l_CDR_tl
1374 \lua_now:n { CDR:highlight_set_var('lang') }
1375 \CDR_tag_get:cN {cache} \l_CDR_tl
1376 \lua_now:n { CDR:highlight_set_var('cache') }
1377 \CDR_tag_get:cN {debug} \l_CDR_tl
1378 \lua_now:n { CDR:highlight_set_var('debug') }
1379 \CDR_tag_get:cN {style} \l_CDR_tl
1380 \lua_now:n { CDR:highlight_set_var('style') }
1381 \lua_now:n { CDR:highlight_set_var('source', 'FV@SV@CDR@Source') }
1382 \clist_set_eq:NN \FV@KeyValues \l_CDR_kv_clist
1383 \FV@UseKeyValues
1384 \FV@FormattingPrep
1385 \CDR@DefinePygSp
1386 \CDR_tag_get:c { format }
1387 \CDR@CodeEngineApply {
1388   \CDR@StyleIfExist { \CDR_tag_get:c { style } } {
1389     \CDR@StyleUseTag
1390     \lua_now:n { CDR:highlight_source(false, true) }
1391   } {
1392     \lua_now:n { CDR:highlight_source(true, true) }
1393     \input { \l_CDR_pyg_sty_tl }
1394     \CDR@StyleUseTag
1395   }
1396   \makeatletter
1397   \lua_now:n {
1398     CDR.synctex_tag = tex.get_synctex_tag();
1399     CDR.synctex_line = tex.inputlineno;
1400     tex.set_synctex_mode(1)
1401   }
1402   \CDR_tag_if_truthy:cT { mbox } { \mbox } {
1403     \input { \l_CDR_pyg_tex_tl } \ignorespaces
1404   }
1405   \lua_now:n {
1406     tex.set_synctex_mode(0)
1407   }
1408   \makeatother
1409 }
1410 \group_end:
1411
1412 }
1413 ] { CDR@Source } #1
1414 } {
1415   \DefineShortVerb { #1 }
1416   \SaveVerb [
1417     aftersave = {
1418       \UndefineShortVerb { #1 }
1419       \cs_set_eq:NN \CDR@FormattingPrep \FV@FormattingPrep
1420       \cs_set:Npn \FV@FormattingPrep {
1421         \CDR@FormattingPrep
1422         \CDR_tag_get:c { format }
1423       }
1424       \CDR@CodeEngineApply {
1425         \CDR_tag_if_truthy:cT { mbox } { \mbox } {
1426           \clist_set_eq:NN \FV@KeyValues \l_CDR_kv_clist

```

```

1427         \FV@UseKeyValues
1428         \FV@FormattingPrep
1429         \FV@SV@CDR@Code
1430     }
1431 }
1432 \group_end:
1433 }
1434 ] { CDR@Code } #1
1435 }
1436 }

```

15 CDRBlock environment

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

15.1 `__block l3keys` module

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

```

1437 \CDR_tag_keys_define:nn { __block } {


```

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

```

1438   no-export .code:n = \CDR_tag_boolean_set:x { #1 },
1439   no-export .default:n = true,


```

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

```

1440   no-export~format .code:n = \CDR_tag_set:,


```

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

```

1441   dry~numbers .code:n = \CDR_tag_set:,
1442   dry~numbers .default:n = true,


```

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

```

1443   test .code:n = \CDR_tag_boolean_set:x { #1 },
1444   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.

```

1445   engine-options .code:n = \CDR_tag_set:,
1446   engine-options .value_required:n = true,

```

 **__initialize** initialize


```

1447 __initialize .meta:n = {
1448     no-export = false,
1449     no-export-format = ,
1450     dry-numbers = false,
1451     test = false,
1452     engine-options = ,
1453 },
1454 __initialize .value_forbidden:n = true,
1455 }

```

15.2 Implementation

15.2.1 Storage

`__start` For the line numbering, these are loop integer controls.

`__step`

`__last` `__start` for the first index

`__step` for the step, defaults to 1

`__last` for the last index, included

```

1456 \CDR_int_new:cn { __start } { 0 }
1457 \CDR_int_new:cn { __step } { 0 }
1458 \CDR_int_new:cn { __last } { 0 }

```

(End definition for `__start`, `__step`, and `__last`.)

15.2.2 Preparation

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.

```

1459 \clist_map_inline:nn { i, ii, iii, iv } {
1460     \cs_set_eq:cc { CDR@ListProcessLine@ #1 } { FV@ListProcessLine@ #1 }
1461 }

```

`\CDRBlock_preflight:n` `\CDRBlock_preflight:n {<CDR@Block kv list>}`

This is a preflight hook intended for testing. The default implementation does nothing.

```

1462 \cs_new:Npn \CDRBlock_preflight:n #1 { }

```

15.2.3 Main environment

`\l_CDR_vrb_seq` All the lines are scanned and recorded before they are processed.

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

```

1463 \seq_new:N \l_CDR_vrb_seq

```

`\FVB@CDRBlock` `fancyvrb` helper to begin the `CDRBlock` environment.

```

1464 \cs_new:Npn \FVB@CDRBlock {
1465   \@bsphack
1466   \exp_args:NV \CDRBlock_preflight:n \FV@KeyValues
1467   \begingroup
1468   \lua_now:n {
1469     CDR.synctex_tag = tex.get_synctex_tag();
1470     CDR.synctex_line = tex.inputlineno;
1471     tex.set_synctex_mode(1)
1472   }
1473   \seq_clear:N \l_CDR_vrb_seq
1474   \cs_set_protected_nopar:Npn \FV@ProcessLine ##1 {
1475     \seq_put_right:Nn \l_CDR_vrb_seq { ##1 }
1476   }
1477   \FV@Scan
1478 }

```

\FVE@CDRBlock fancyvrb helper to end the CDRBlock environment.

```

1479 \cs_new:Npn \FVE@CDRBlock {%
1480   \CDRBlock_setup:
1481   \CDR_if_no_export:F {
1482     \seq_map_inline:Nn \l_CDR_vrb_seq {
1483       \tl_set:Nn \l_CDR_tl { ##1 }
1484       \lua_now:n { CDR:record_line('l_CDR_tl') }
1485     }
1486   }
1487   \CDRBlock_engine_begin:
1488   \CDR_if_pygments:TF {
1489     \CDRBlock@Pyg
1490   } {
1491     \CDRBlock@FV
1492   }
1493   \lua_now:n {
1494     tex.set_synctex_mode(0);
1495     CDR.synctex_line = 0;
1496   }
1497   \CDRBlock_engine_end:
1498   \CDRBlock_teardown:
1499   \endgroup
1500   \@esphack
1501 }
1502 \DefineVerbatimEnvironment{CDRBlock}{CDRBlock}{}
1503 % \begin{MacroCode}
1504 \cs_new_protected_nopar:Npn \CDRBlock_setup: {
1505   \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
1506     \prg_return_true:
1507   }
1508   \CDR_tag_keys_set:nn { __block } { __initialize }

```

Read and catch the key value arguments, except the ones related to fancyvrb. Then build the dynamic keys matching *<engine name>* engine options for appropriate engine names.

```

1509 \CDRBlock_setup_tags_and_engine:N \FV@KeyValues
1510 \CDR_local_inherit:n {
1511     __block, __pygments.block, default.block,
1512     __pygments, default
1513 }
1514 \CDR_local_set_known:N \FV@KeyValues
1515 \CDR_tag_provide_from_kv:V \FV@KeyValues
1516 \CDR_local_set_known:N \FV@KeyValues
1517 \CDR@Debug{CDRBlock.KV1:\l_CDR_kv_clist}

```

Now `\FV@KeyValues` is meant to contains only keys related to `fancyvrb` but we still need to filter them out. If the display engine is not the default one, we catch any key related to framing. Anyways, we catch keys related to numbering because line numbering is completely performed by `coder`.

```

1518 \CDR_local_inherit:n {
1519     \CDR_tag_if_eq:cnF { engine } { default } {
1520         __fancyvrb.frame,
1521     },
1522     __fancyvrb.number,
1523 }
1524 \CDR_local_set_known:N \FV@KeyValues

```

These keys are read without removing them later and eventually forwarded to `fancyvrb` through its natural `\FV@UseKeyValues` mechanism.

```

1525 \CDR_local_inherit:n {
1526     __fancyvrb.block,
1527     __fancyvrb,
1528 }
1529 \CDR_local_set_known:VN \FV@KeyValues \l_CDR_kv_clist
1530 \lua_now:n {
1531     CDR:highlight_block_setup('g_CDR_tags_clist')
1532 }
1533 \CDR_set_conditional:Nn \CDR_if_pygments:
1534 { \CDR_tag_if_truthy_p:c { pygments } }
1535 \CDR_set_conditional:Nn \CDR_if_no_export:
1536 { \CDR_tag_if_truthy_p:c { no-export } }
1537 \CDR_set_conditional:Nn \CDR_if_dry_numbers:
1538 { \CDR_tag_if_truthy_p:c { dry-numbers } }
1539 \CDR_set_conditional:Nn \CDR_if_number_on:
1540 { ! \CDR_tag_if_eq_p:cn { numbers } { none } }
1541 \CDR_set_conditional:Nn \CDR_tags_if_already: {
1542     \CDR_tag_if_truthy_p:c { only-top } &&
1543     \CDR_clist_if_eq_p:NN \g_CDR_tags_clist \g_CDR_last_tags_clist
1544 }
1545 \CDR_if_number_on:T {
1546     \clist_map_inline:Nn \g_CDR_tags_clist {
1547         \CDR_int_if_exist:cF { ##1 } {
1548             \CDR_int_new:cn { ##1 } { 1 }
1549         }
1550     }
1551 }
1552 }

```

```

1553 \cs_new_protected_nopar:Npn \CDRBlock_teardown: {
1554   \CDR_if_dry_numbers:F {
1555     \tl_set:Nx \l_CDR_tl { \seq_count:N \l_CDR_vrb_seq }
1556     \clist_map_inline:Nn \g_CDR_tags_clist {
1557       \CDR_int_gadd:cn { ##1 } { \l_CDR_tl }
1558     }
1559   }
1560   \lua_now:n {
1561     CDR:highlight_block_teardown()
1562   }
1563 }

```

15.2.4 pygments only

Parts of CDRBlock environment specific to pygments.

\CDRBlock@Pyg	\CDRBlock@Pyg
---------------	---------------

The code chunk is stored line by line in `\l_CDR_vrb_seq`. Use `pygments` to colorize the code, and use `fancyvrb` once more to display the colored code.

```

1564 \cs_set_protected:Npn \CDRBlock@Pyg {
1565   \CDR@Debug { \string\CDRBlock@Pyg/\the\inputlineno }
1566   \CDR_tag_get:cN {lang} \l_CDR_tl
1567   \lua_now:n { CDR:highlight_set_var('lang') }
1568   \CDR_tag_get:cN {cache} \l_CDR_tl
1569   \lua_now:n { CDR:highlight_set_var('cache') }
1570   \CDR_tag_get:cN {debug} \l_CDR_tl
1571   \lua_now:n { CDR:highlight_set_var('debug') }
1572   \CDR_tag_get:cN {texcomments} \l_CDR_tl
1573   \lua_now:n { CDR:highlight_set_var('texcomments') }
1574   \CDR_tag_get:cN {escapeinside} \l_CDR_tl
1575   \lua_now:n { CDR:highlight_set_var('escapeinside') }
1576   \CDR_tag_get:cN {mathescape} \l_CDR_tl
1577   \lua_now:n { CDR:highlight_set_var('mathescape') }
1578   \CDR_tag_get:cN {style} \l_CDR_tl
1579   \lua_now:n { CDR:highlight_set_var('style') }
1580   \CDR@StyleIfExist { \l_CDR_tl } { } {
1581     \lua_now:n { CDR:highlight_source(true, false) }
1582     \input { \l_CDR_pyg_sty_tl }
1583   }
1584   \CDR@StyleUseTag
1585   \CDR@DefinePygSp
1586   \lua_now:n { CDR:highlight_source(false, true) }
1587   \fvset{ commandchars=\\{\} }
1588   \FV@UseVerbatim {
1589     \CDR_tag_get:c { format }
1590     \CDR_if_no_export:T {
1591       \CDR_tag_get:c { no~export~format }
1592     }
1593     \makeatletter
1594     \input{ \l_CDR_pyg_tex_tl }
1595     \makeatother
1596     \def \FV@ProcessLine {}

```

```

1597 }
1598 \CDR_if_number_on:T {
1599   \CDR_int_add:cn { __last } { 1 }
1600   \clist_map_inline:Nn \g_CDR_tags_clist {
1601     \CDR_int_gset:cc { ##1 } { __last }
1602 \CDR@Debug {DEBUG.CDRBlock.LAST: ##1 -> \CDR_int_use:c { ##1 } }
1603   }
1604 }
1605 }

```

Info

```

1606 \cs_new:Npn \CDR@NumberFormat {
1607   \CDR_tag_get:c { numbers~format }
1608 }
1609 \cs_new:Npn \CDR@NumberSep {
1610   \hspace{ \CDR_tag_get:c { numbersep } }
1611 }
1612 \cs_new:Npn \CDR@TagsFormat {
1613   \CDR_tag_get:c { tags~format }
1614 }

```

<code>\CDR_info_N_L:n</code>	<code>\CDR_info_N_L:n {<line number>}</code>
<code>\CDR_info_N_R:n</code>	<code>\CDR_info_T_L:n {<line number>}</code>
<code>\CDR_info_T_L:n</code>	Core methods to display the left and right information. The T variants contain tags informations, they are only used on the first line eventually. The N variants are for line numbers only.
<code>\CDR_info_T_R:n</code>	

```

1615 \cs_new:Npn \CDR_info_N_L:n #1 {
1616   \hbox_overlap_left:n {
1617     \cs_set:Npn \baselinestretch { 1 }
1618     { \CDR@NumberFormat
1619       #1
1620     }
1621     \CDR@NumberSep
1622   }
1623 }
1624 \cs_new:Npn \CDR_info_T_L:n #1 {
1625   \hbox_overlap_left:n {
1626     \cs_set:Npn \baselinestretch { 1 }
1627     \CDR@NumberFormat
1628     \smash{
1629       \parbox[b]{\marginparwidth}{
1630         \raggedleft
1631         { \CDR@TagsFormat \g_CDR_tags_clist :}
1632       }
1633       #1
1634     }
1635     \CDR@NumberSep
1636   }
1637 }
1638 \cs_new:Npn \CDR_info_N_R:n #1 {
1639   \hbox_overlap_right:n {

```

```

1640 \CDR@NumberSep
1641 \cs_set:Npn \baselinestretch { 1 }
1642 \CDR@NumberFormat
1643 #1
1644 }
1645 }
1646 \cs_new:Npn \CDR_info_T_R:n #1 {
1647 \hbox_overlap_right:n {
1648 \cs_set:Npn \baselinestretch { 1 }
1649 \CDR@NumberSep
1650 \CDR@NumberFormat
1651 \smash {
1652 \parbox[b]{\marginparwidth}{
1653 \raggedright
1654 #1:
1655 {\CDR@TagsFormat \space \g_CDR_tags_clist}
1656 }
1657 }
1658 }
1659 }

```

`\CDR_number_alt:n` First line.

```

1660 \cs_set:Npn \CDR_number_alt:n #1 {
1661 \use:c { CDRNumber
1662 \CDR_if_number_visible:nTF { #1 } { Main } { Other }
1663 } { #1 }
1664 }
1665 \cs_set:Npn \CDR_number_alt: {
1666 \CDR@Debug{ALT: \CDR_int_use:c { __ } }
1667 \CDR_number_alt:n { \CDR_int_use:c { __ } }
1668 }

```

<code>\CDRNumberMain</code>	<code>\CDRNumberMain {⟨integer expression⟩}</code>
<code>\CDRNumberOther</code>	<code>\CDRNumberOther {⟨integer expression⟩}</code>

This is used when typesetting line numbers. The default `...Other` function just gobble one argument. The `⟨integer expression⟩` is exactly what will be displayed.

```

1669 \cs_new:Npn \CDRNumberMain {
1670 }
1671 \cs_new:Npn \CDRNumberOther {
1672 \use_none:n
1673 }

```

<code>\CDR@NumberMain</code>	<code>\CDR@NumberMain</code>
<code>\CDR@NumberOther</code>	<code>\CDR@NumberOther</code>

Respectively apply `\CDR@NumberMain` or `\CDR@NumberOther` on `\CDR_int_use:c { __ }`

```

1674 \cs_new:Npn \CDR@NumberMain {
1675     \CDRNumberMain { \CDR_int_use:c { __ } }
1676 }
1677 \cs_new:Npn \CDR@NumberOther {
1678     \CDRNumberOther { \CDR_int_use:c { __ } }
1679 }

```

Boxes for lines The first index is for the tags (L, R, N, A, M), the second for the numbers (L, R, N). L stands for left, R stands for right, N stands for nothing, S stands for same side as numbers, O stands for opposite side of numbers.

<code>\CDR_line_[LRNSO]_[LRN]:nn</code>	<code>\CDR_line_[LRNSO]_[LRN]:nn {<line number>} {<line content>}</code>
---	--

These functions may be called by `\CDR_line:nnn` on each block. LRNSO corresponds to the `show tags` options whereas LRN corresponds to the `numbers` options. These functions display the first line and setup the next one.

```

1680 \cs_new:Npn \CDR_line_N_N:n {
1681     \CDR@Debug {Debug.CDR_line_N_N:n}
1682     \CDR_line_box_N:n
1683 }
1684
1685 \cs_new:Npn \CDR_line_L_N:n #1 {
1686     \CDR@Debug {Debug.CDR_line_L_N:n}
1687     \CDR_line_box:nnn { \CDR_info_T_L:n { } } { #1 } { }
1688 }
1689
1690 \cs_new:Npn \CDR_line_R_N:n #1 {
1691     \CDR@Debug {Debug.CDR_line_R_N:n}
1692     \CDR_line_box:nnn { } { #1 } { \CDR_info_T_R:n { } }
1693 }
1694
1695 \cs_new:Npn \CDR_line_S_N:n {
1696     \CDR@Debug {Debug.CDR_line_S_N:n}
1697     \CDR_line_box_N:n
1698 }
1699
1700 \cs_new:Npn \CDR_line_O_N:n {
1701     \CDR@Debug {STEP:CDR_line_O_N:n}
1702     \CDR_line_box_N:n
1703 }
1704
1705 \cs_new:Npn \CDR_line_N_L:n #1 {
1706     \CDR@Debug {STEP:CDR_line_N_L:n}
1707     \CDR_if_no_number:TF {
1708         \CDR_line_box:nnn {
1709             \CDR_info_N_L:n { \CDR@NumberMain }
1710         } { #1 } {}
1711     } {
1712         \CDR_line_box_L:n { #1 }
1713     }
1714 }
1715
1716 \cs_new:Npn \CDR_line_L_L:n #1 {

```

```

1717 \CDR@Debug {STEP:CDR_line_L_L:n}
1718 \CDR_if_number_single:TF {
1719 \CDR_line_box:nnn {
1720 \CDR_info_T_L:n { \space \CDR@NumberMain }
1721 } { #1 } {}
1722 } {
1723 \CDR_if_no_number:TF {
1724 \cs_set:Npn \CDR@@Line {
1725 \cs_set:Npn \CDR@@Line {
1726 \CDR_line_box_L:nn { \CDR_info_N_L:n { \CDR@NumberOther } }
1727 }
1728 \CDR_line_box_L:nn { \CDR_info_N_L:n { \CDR@NumberMain } }
1729 }
1730 } {
1731 \cs_set:Npn \CDR@@Line {
1732 \CDR_line_box_L:nn { \CDR_info_N_L:n { \CDR_number_alt: } }
1733 }
1734 }
1735 \CDR_line_box:nnn { \CDR_info_T_L:n { } } { #1 } { }
1736 }
1737 }
1738
1739 \cs_new:Npn \CDR_line_R_R:n #1 {
1740 \CDR@Debug {STEP:CDR_line_R_R:n}
1741 \CDR_if_number_single:TF {
1742 \CDR_line_box:nnn { } { #1 } {
1743 \CDR_info_T_R:n { \CDR@NumberMain }
1744 }
1745 } {
1746 \CDR_if_no_number:TF {
1747 \cs_set:Npn \CDR@@Line {
1748 \cs_set:Npn \CDR@@Line {
1749 \CDR_line_box_R:nn { \CDR_info_N_R:n { \CDR@NumberOther } }
1750 }
1751 \CDR_line_box_R:nn { \CDR_info_N_R:n { \CDR@NumberMain } }
1752 }
1753 } {
1754 \cs_set:Npn \CDR@@Line {
1755 \CDR_line_box_R:nn { \CDR_info_N_R:n { \CDR_number_alt: } }
1756 }
1757 }
1758 \CDR_line_box:nnn { } { #1 } { \CDR_info_T_R:n { } }
1759 }
1760 }
1761
1762 \cs_new:Npn \CDR_line_R_L:n #1 {
1763 \CDR@Debug {STEP:CDR_line_R_L:n}
1764 \CDR_line_box:nnn {
1765 \CDR_if_no_number:TF {
1766 \CDR_info_N_L:n { \CDR@NumberMain }
1767 } {
1768 \CDR_info_N_L:n { \CDR_number_alt: }
1769 }
1770 } { #1 } {

```



```

1771     \CDR_info_T_R:n { }
1772   }
1773 }
1774
1775 \cs_set_eq:NN \CDR_line_S_L:n \CDR_line_L_L:n
1776 \cs_set_eq:NN \CDR_line_O_L:n \CDR_line_R_L:n
1777
1778 \cs_new:Npn \CDR_line_N_R:n {
1779   \typeout {STEP:CDR_line_N_R:n}
1780   \CDR_line_box_R:n
1781 }
1782
1783 \cs_new:Npn \CDR_line_L_R:n #1 {
1784   \CDR@Debug {STEP:CDR_line_L_R:n}
1785   \CDR_line_box:nnn {
1786     \CDR_info_T_L:n { }
1787   } { #1 } {
1788     \CDR_if_no_number:TF {
1789       \CDR_info_N_R:n { \CDR@NumberMain }
1790     } {
1791       \CDR_info_N_R:n { \CDR_number_alt: }
1792     }
1793   }
1794 }
1795
1796 \cs_set_eq:NN \CDR_line_S_R:n \CDR_line_R_R:n
1797 \cs_set_eq:NN \CDR_line_O_R:n \CDR_line_L_R:n
1798
1799
1800 \cs_new:Npn \CDR_line_box_N:n #1 {
1801   \CDR@Debug {STEP:CDR_line_box_N:n}
1802   \CDR_line_box:nnn { } { #1 } {}
1803 }
1804
1805 \cs_new:Npn \CDR_line_box_L:n #1 {
1806   \CDR@Debug {STEP:CDR_line_box_L:n}
1807   \CDR_line_box:nnn {
1808     \CDR_info_N_L:n { \CDR_number_alt: }
1809   } { #1 } {}
1810 }
1811
1812 \cs_new:Npn \CDR_line_box_R:n #1 {
1813   \CDR@Debug {STEP:CDR_line_box_R:n}
1814   \CDR_line_box:nnn { } { #1 } {
1815     \CDR_info_N_R:n { \CDR_number_alt: }
1816   }
1817 }

```

<code>\CDR_line_box:nnn</code>	<code>\CDR_line_box:nnn {<left info>} {<line content>} {<right info>}</code>
<code>\CDR_line_box_L:nn</code>	<code>\CDR_line_box_L:nn {<left info>} {<line content>}</code>
<code>\CDR_line_box_R:nn</code>	<code>\CDR_line_box_R:nn {<right info>} {<line content>}</code>
<code>\CDR_line_box:nn</code>	

Returns an hbox with the given material. The first LR command is the reference, from which are derived the L, R and N commands. At run time the `\CDR_line_box:nn` is defined to call one of the above commands (with the same signature).

```

1818 \cs_new:Npn \CDR_line_box:nnn #1 #2 #3 {
1819 \CDR@Debug {\string\CDR_line_box:nnn/\tl_to_str:n{#1}/.../\tl_to_str:n{#3}/}
1820 \directlua {
1821     tex.set_synctex_tag( CDR.synctex_tag )
1822 }
1823 \lua_now:e {
1824     tex.set_synctex_line(CDR.synctex_line+( \CDR_int_use:c { __ } ) )
1825 }
1826 \hbox to \hsize {
1827     \kern \leftmargin
1828     #1
1829     \hbox to \linewidth {
1830         \FV@LeftListFrame
1831         #2
1832         \hss
1833         \FV@RightListFrame
1834     }
1835     #3
1836 } \ignorespaces
1837 }
1838 \cs_new:Npn \CDR_line_box_L:nn #1 #2 {
1839 \CDR_line_box:nnn { #1 } { #2 } {}
1840 }
1841 \cs_new:Npn \CDR_line_box_R:nn #1 #2 {
1842 \CDR@Debug {STEP:CDR_line_box_R:nn}
1843 \CDR_line_box:nnn { } {#2} { #1 }
1844 }
1845 \cs_new:Npn \CDR_line_box_N:nn #1 #2 {
1846 \CDR@Debug {STEP:CDR_line_box_N:nn}
1847 \CDR_line_box:nnn { } { #2 } {}
1848 }

```

Lines

```

1849 \cs_new:Npn \CDR@Line {
1850 \CDR@Debug {\string\CDR@Line}
1851 \peek_meaning_ignore_spaces:NTF [%]
1852 { \CDR_line:nnn } {
1853     \PackageError
1854     { coder }
1855     { Missing~'[~'%]
1856     ~at~first~\string\CDR@Line~call }
1857     { See~the~coder~developer~manual }
1858 }
1859 }

```

```
\CDR_line:nnn {<CDR@Line kv list>} {<line number>} {<line content>}
```

This is the very first command called when typesetting. Some setup are made for line numbering, in particular the `\CDR_if_visible_at_index:n...` family is set here. The first line must read `\CDR@Line[last=...]{1}{...}`, be it input from any `...pyg.tex` files or directly, like for `fancyvrb` usage.

```
1860 \keys_define:nn { CDR@Line } {
1861   last .code:n = \CDR_int_set:cn { __last } { #1 },
1862 }
1863 \cs_new:Npn \CDR_line:nnn [ #1 ] #2 {
1864   \CDR@Debug {\string\CDR_line:nnn}
1865   \keys_set:nn { CDR@Line } { #1 }
1866   \CDR_int_set:cn { __ } { 0 }
1867   \CDR_if_number_on:TF {
1868     \CDR_tag_if_eq:cnTF { firstnumber } { last } {
1869       \clist_map_inline:Nn \g_CDR_tags_clist {
1870         \clist_map_break:n {
1871           \CDR_int_set:cc { __start } { ##1 }
1872         }
1873       }
1874     }
1875   } {
1876     \CDR_tag_if_eq:cnTF { firstnumber } { auto } {
1877       \CDR_int_set:cn { __start } { 1 }
1878     } {
1879       \CDR_int_set:cn { __start } { \CDR_tag_get:c { firstnumber } }
1880     }
1881   }
```

Make `__last` absolute only after defining the `\CDR_if_number_single...` conditionals.

```
1882   \CDR_set_conditional:Nn \CDR_if_number_single: {
1883     \CDR_int_compare_p:cNn { __last } = 1
1884   }
1885   \CDR@Debug{***** TEST: \CDR_if_number_single:TF { SINGLE } { MULTI } }
1886   \CDR_int_add:cn { __last } { \CDR_int:c { __start } - 1 }
1887   \CDR_int_set:cn { __step } { \CDR_tag_get:c { stepnumber } }
1888   \CDR@Debug {CDR_line:nnn:START/STEP/LAST=\CDR_int_use:c { __start }/\CDR_int_use:c { __step } /\
```

```
\CDR_if_visible_at_index_p:n * \CDR_if_visible_at_index:nTF {<relative line number>} {<true code>}
\CDR_if_visible_at_index:nTF * {<false code>}
```

The `<relative line number>` is the first braced token after `\CDR@Line` in the various colored `...pyg.tex` files. Execute `<true code>` if the `<relative line number>` is visible, `<false code>` otherwise. The `<relative line number>` visibility depends on the value relative to first number and the step. This is relevant only when line numbering is enabled. Some setup are made for line numbering, in particular the `\CDR_if_visible_at_index:n...` family is set here.

```
1889   \CDR_int_compare:cNnTF { __step } < 2 {
1890     \CDR_int_set:cn { __step } { 1 }
1891     \CDR_set_conditional_alt:Nn \CDR_if_visible_at_index:n {
1892       ! \CDR_int_compare_p:cNn { __last } < { ##1 + \CDR_int:c { __start } - 1 }
```

```

1893     }
1894     \CDR_set_conditional_alt:Nn \CDR_if_number_visible:n {
1895       ! \CDR_int_compare_p:cNn { __last } < { ##1 }
1896     }
1897   } {
1898     \CDR_set_conditional_alt:Nn \CDR_if_visible_at_index:n {
1899       ! \CDR_int_compare_p:cNn { __last } < { ##1 + \CDR_int:c { __start } - 1 }
1900     }
1901     \CDR_set_conditional_alt:Nn \CDR_if_number_visible:n {
1902       \int_compare_p:nNn {
1903         ( ##1 + \CDR_int:c { __start } - 1 )
1904         / \CDR_int:c { __step } * \CDR_int:c { __step }
1905         - \CDR_int:c { __start } + 1
1906       } = { ##1 }
1907       && \CDR_if_visible_at_index_p:n { ##1 }
1908     }
1909   }
1910 \CDR@Debug {CDR_line:nnn:1}

1911   \CDR_set_conditional:Nn \CDR_if_no_number: {
1912     \CDR_int_compare_p:cNn { __start } > {
1913       \CDR_int:c { __last } / \CDR_int:c { __step } * \CDR_int:c { __step }
1914     }
1915   }
1916   \cs_set:Npn \CDR@Line ##1 {
1917 \CDR@Debug {\string\CDR@Line(A), \the\inputlineno}
1918     \CDR_int_set:cn { __ } { ##1 + \CDR_int:c { __start } - #2 }
1919     \CDR@@Line
1920   }
1921   \CDR_int_set:cn { __ } { \CDR_int:c { __start } + 1 - #2 }
1922 } {
1923 \CDR@Debug {NUMBER-OFF}
1924   \cs_set:Npn \CDR@Line ##1 {
1925 \CDR@Debug {\string\CDR@Line(B), \the\inputlineno}
1926     \CDR@@Line
1927   }
1928 }
1929 \CDR@Debug {STEP_S, \CDR_int_use:c {__step}, \CDR_int_use:c {__last} }

```

Convenient method to branch whether one line number will be displayed or not, considering the stepping. When numbering is on, each code chunk must have at least one number. One solution is to allways display the first one but it is not satisfying when lines are numbered stepwise, moreover when the tags should be displayed.

```

1930 \tl_clear:N \l_CDR_tl
1931 \CDR_tags_if_already:TF {
1932   \tl_put_right:Nn \l_CDR_tl { _N }
1933 } {
1934   \exp_args:Nx
1935   \str_case:nnF { \CDR_tag_get:c { show-tags } } {
1936     { left } { \tl_put_right:Nn \l_CDR_tl { _L } }
1937     { right } { \tl_put_right:Nn \l_CDR_tl { _R } }
1938     { none } { \tl_put_right:Nn \l_CDR_tl { _N } }
1939     { numbers } { \tl_put_right:Nn \l_CDR_tl { _S } }
1940     { mirror } { \tl_put_right:Nn \l_CDR_tl { _O } }

```

```

1941 } { \PackageError
1942     { coder }
1943     { Unknown~show~tags~options~::~ \CDR_tag_get:c { show~tags } }
1944     { See~the~coder~manual }
1945 }
1946 }

```

By default, the next line is displayed with no tag, but the real content may change to save space.

```

1947 \exp_args:Nx
1948 \str_case:nnF { \CDR_tag_get:c { numbers } } {
1949     { left } {
1950         \tl_put_right:Nn \l_CDR_tl { _L }
1951         \cs_set:Npn \CDR@@Line { \CDR_line_box_L:n }
1952     }
1953     { right } {
1954         \tl_put_right:Nn \l_CDR_tl { _R }
1955         \cs_set:Npn \CDR@@Line { \CDR_line_box_R:n }
1956     }
1957     { none } {
1958         \tl_put_right:Nn \l_CDR_tl { _N }
1959         \cs_set:Npn \CDR@@Line { \CDR_line_box_N:n }
1960     }
1961 } { \PackageError
1962     { coder }
1963     { Unknown~numbers~options~::~ \CDR_tag_get:c { numbers } }
1964     { See~the~coder~manual }
1965 }
1966 \CDR@Debug {BRANCH:CDR_line \l_CDR_tl :n}
1967 \use:c { CDR_line \l_CDR_tl :n }
1968 }

```

15.2.5 fancyvrb only

pygments is not used, fall back to fancyvrb features.

```
CDRBlock@FV \CDRBlock@Fv
```

```

1969 \cs_new_protected:Npn \CDRBlock@FV {
1970 \CDR@Debug {DEBUG.Block.FV}
1971 \FV@UseKeyValues
1972 \FV@UseVerbatim {
1973     \CDR_tag_get:c { format }
1974     \CDR_if_no_export:T {
1975         \CDR_tag_get:c { no~export~format }
1976     }
1977     \tl_set:Nx \l_CDR_tl { [ last=%]
1978         \seq_count:N \l_CDR_vrb_seq %[
1979     ] }
1980     \seq_map_indexed_inline:Nn \l_CDR_vrb_seq {
1981         \exp_last_unbraced:NV \CDR@Line \l_CDR_tl { ##1 } { ##2 }
1982         \tl_clear:N \l_CDR_tl
1983     }

```

```

1984 \tl_clear:N \FV@ProcessLine
1985 }
1986 \CDR_if_number_on:T {
1987   \CDR_int_compare:cNnTF { __ } > 0 {
1988     \CDR_int_set:cn { __ } {
1989       \value{FancyVerbLine} - \CDR_int_use:c { __ } + 1
1990     }
1991     \clist_map_inline:Nn \g_CDR_tags_clist {
1992       \CDR_int_gadd:cc { ##1 } { __ }
1993     }
1994   } {
1995     \CDR_int_set:cn { __ } { \value{FancyVerbLine} + 1 }
1996     \clist_map_inline:Nn \g_CDR_tags_clist {
1997       \CDR_int_gset:cc { ##1 } { __ }
1998     }
1999   }
2000 }
2001 }
2002 }

```

15.2.6 Utilities

This is put aside for better clarity.

<code>\CDR_set_conditional:Nn</code>	<code>\CDR_set_conditional:Nn <core name> {<condition>}</code>
--------------------------------------	--

Wrapper over `\prg_set_conditional:Nnn`.

```

2003 \cs_new:Npn \CDR_set_conditional:Nn #1 #2 {
2004   \bool_if:nTF { #2 } {
2005     \prg_set_conditional:Nnn #1 { p, T, F, TF } { \prg_return_true: }
2006   } {
2007     \prg_set_conditional:Nnn #1 { p, T, F, TF } { \prg_return_false: }
2008   }
2009 }

```

<code>\CDR_set_conditional_alt:Nn</code>	<code>\CDR_set_conditional_alt:Nnnn <core name> {<condition>}</code>
--	--

Wrapper over `\prg_set_conditional:Nnn`.

```

2010 \cs_new:Npn \CDR_set_conditional_alt:Nn #1 #2 {
2011   \prg_set_conditional:Nnn #1 { p, T, F, TF } {
2012     \bool_if:nTF { #2 } { \prg_return_true: } { \prg_return_false: }
2013   }
2014 }

```

<code>\CDR_if_middle_column:</code>	<code>\CDR_int_if_middle_column:TF {<true code>} {<false code>}</code>
-------------------------------------	--

<code>\CDR_if_right_column:</code>	<code>\CDR_int_if_right_column:TF {<true code>} {<false code>}</code>
------------------------------------	---

Execute `<true code>` when in the middle or right column, `<false code>` otherwise.

```

2015 \prg_set_conditional:Nnn \CDR_if_middle_column: { p, T, F, TF } { \prg_return_false: }
2016 \prg_set_conditional:Nnn \CDR_if_right_column: { p, T, F, TF } { \prg_return_false: }

```

Various utility conditionals: their purpose is to clarify the code. They are available in the CDRBlock environment only.

```
\CDR_tags_if_visible_p:n ★ \CDR_tags_if_visible:nTF {\left|right}\{<true code>\} {\<false code>\}
\CDR_tags_if_visible:nTF ★
```

Whether the tags should be visible, at the left or at the right.

```
2017 \prg_set_conditional:Nnn \CDR_tags_if_visible:n { p, T, F, TF } {
2018   \bool_if:nTF {
2019     ( \CDR_tag_if_eq_p:cn { show-tags } { ##1 } ||
2020       \CDR_tag_if_eq_p:cn { show-tags } { numbers } &&
2021       \CDR_tag_if_eq_p:cn { numbers } { ##1 }
2022     ) && ! \CDR_tags_if_already_p:
2023   } {
2024     \prg_return_true:
2025   } {
2026     \prg_return_false:
2027   }
2028 }
```

```
\CDRBlock_setup_tags_and_engine:N \CDRBlock_setup_tags_and_engine:N {\<list var>}
```

Utility to setup the tags and the tag inheritance tree. When not provided explicitly with the `tags=...` user interface, a code chunk will have the list of tags stored in `\g_CDR_tags_clist` by last `\CDRExport`, `\CDRSet` or `\CDRBlock` environment. At least one tag must be provided, either implicitly or explicitly.

```
2029 \cs_new_protected_nopar:Npn \CDRBlock_setup_tags_and_engine:N #1 {
2030   \CDR_local_inherit:n { __tags }
2031   \CDR_local_set_known:N #1
2032   \CDR_tag_if_exist_here:ccT { __local } { tags } {
2033     \CDR_tag_get:cN { tags } \l_CDR_clist
2034     \clist_if_empty:NF \l_CDR_clist {
2035       \clist_gset_eq:NN \g_CDR_tags_clist \l_CDR_clist
2036     }
2037   }
2038   \clist_if_empty:NT \g_CDR_tags_clist {
2039     \PackageWarning
2040       { coder }
2041       { No~(default)~tags~provided. }
2042   }
2043   \CDR@Debug {CDRBlock_setup_tags:\space\g_CDR_tags_clist}
```

Setup the inheritance tree for the `\CDR_tag_get:...` related functions.

```
2044 \CDR_get_inherit:f {
2045   \g_CDR_tags_clist,
2046   __block, __tags, __engine, default.block, __pygments.block,
2047   __fancyvrb.block __fancyvrb.frame, __fancyvrb.number,
2048   __pygments, default, __fancyvrb,
2049 }
```

For each `<tag name>`, create an `\int` variable and initialize it to 1.

```

2050 \clist_map_inline:Nn \g_CDR_tags_clist {
2051   \CDR_int_if_exist:cF { ##1 } {
2052     \CDR_int_new:cn { ##1 } { 1 }
2053   }
2054 }

```

Now setup the engine options if any.

```

2055 \CDR_local_inherit:n { __engine }
2056 \CDR_local_set_known:N #1
2057 \CDR_tag_get:cNT { engine } \l_CDR_tl {
2058   \clist_put_left:Nx #1 { \CDRBlock_options_use:V \l_CDR_tl }
2059 }
2060 }

```

16 Management

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

```

2061 \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_p: ★ \CDR_if_show_code:TF {⟨true code⟩} {⟨false code⟩}

```

```

\CDR_if_show_code:TF ★

```

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

```

2062 \prg_new_conditional:Nnn \CDR_if_show_code: { p, T, F, TF } {
2063   \bool_if:nTF {
2064     \g_CDR_in_impl_bool && !\g_CDR_with_impl_bool
2065   } {
2066     \prg_return_false:
2067   } {
2068     \prg_return_true:
2069   }
2070 }

```

`\g_CDR_with_impl_bool`

```

2071 \bool_new:N \g_CDR_with_impl_bool

```

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

```

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

```

Store the content of *⟨file name⟩* into the variable *⟨variable⟩*. This is currently unstable.

```

2072 \DeclareDocumentCommand \CDRPreamble { m m } {
2073   \msg_info:nnn
2074     { coder }
2075     { :n }
2076     { Reading-preamble-from-file-"#2". }
2077   \tl_set:Nn \l_CDR_tl { #2 }
2078   \exp_args:NNx
2079   \tl_set:Nx #1 { \lua_now:n {CDR.print_file_content('l_CDR_tl')} }
2080 }

```


17 Section separators

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

18 Finale

```

2081 \newcounter{CDR@impl@page}
2082 \DeclareDocumentCommand \CDRImplementation {} {
2083   \bool_if:NF \g_CDR_with_impl_bool {
2084     \clearpage
2085     \bool_gset_true:N \g_CDR_in_impl_bool
2086     \let\CDR@old@part\part
2087     \DeclareDocumentCommand\part{som}{}
2088     \let\CDR@old@section\section
2089     \DeclareDocumentCommand\section{som}{}
2090     \let\CDR@old@subsection\subsection
2091     \DeclareDocumentCommand\subsection{som}{}
2092     \let\CDR@old@subsubsection\subsubsection
2093     \DeclareDocumentCommand\subsubsection{som}{}
2094     \let\CDR@old@paragraph\paragraph
2095     \DeclareDocumentCommand\paragraph{som}{}
2096     \let\CDR@old@subparagraph\subparagraph
2097     \DeclareDocumentCommand\subparagraph{som}{}
2098     \cs_if_exist:NT \refsection{ \refsection }
2099     \setcounter{ CDR@impl@page }{ \value{page} }
2100   }
2101 }
2102 \DeclareDocumentCommand\CDRFinale {} {
2103   \bool_if:NF \g_CDR_with_impl_bool {
2104     \clearpage
2105     \bool_gset_false:N \g_CDR_in_impl_bool
2106     \let\part\CDR@old@part
2107     \let\section\CDR@old@section
2108     \let\subsection\CDR@old@subsection
2109     \let\subsubsection\CDR@old@subsubsection
2110     \let\paragraph\CDR@old@paragraph
2111     \let\subparagraph\CDR@old@subparagraph
2112     \setcounter { page } { \value{ CDR@impl@page } }
2113   }
2114 }
2115 %\cs_set_eq:NN \CDR_line_number: \prg_do_nothing:

```

19 Finale

```

2116 %\AddToHook { cmd/FancyVerbFormatLine/before } {
2117 %   \CDR_line_number:
2118 %}

```

```
2119
2120 \ExplSyntaxOff
2121
```

Input a configuration file named `coder.cfg`, if any.

```
2122 \AtBeginDocument{
2123   \InputIfFileExists{coder.cfg}{}{}
2124 }
2125 %</sty>
```