

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

6.3 LaTeX

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

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

All the line templates below are L^AT_EX source text where `<placeholder:number>` should be replaced by a line number and `<placeholder:line>` should be replaced by the highlighted line code provided by `pygments`. They should not include a trailing newline char. 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-jsn')
```

3 General purpose material

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

```

11 local CDR_PY_PATH = kpse.find_file('coder-tool.py')
```

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


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

```

12 local PYTHON_PATH = io.popen([[which python]]):read('a'):match("^%s*(.-)%s*$")
```

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

set_python_path CDR:set_python_path(<path var>)

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

```

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

```

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


Execute <true code> if <string> is the string "true", <false code> otherwise.

```

23 local function is_truthy(s)
24   return s == 'true'
25 end

```

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

 Escape the given string to be used by the shell.

```

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

```

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

Make a directory at the given path.

```

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>)

Highlight the currently entered block if <src> is true, build the style definitions if <sty> is true. Build a configuration table with all data necessary for the processing, save it as a JSON file and launch coder-tool.py with the proper arguments. 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 $\langle \text{code var name} \rangle$. 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($\langle \text{tags clist var} \rangle$)

Records the contents of the $\langle \text{tags clist var} \rangle$ 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.

<u><code>export_file</code></u>	<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
```

<u><code>export_file_info</code></u>	<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
```

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

```

3.1 TeXOpts class

```

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

```

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

```

40 sty_template=r'''% !TeX root=...
41 \makeatletter
42 \CDR@StyleDefine{<placeholder:style_name>} {%
43   <placeholder:style_defs>}%
44 \makeatother'''
45 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 = 'Opt'
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 = 'Opt'
90     xrightmargin = 'Opt'
91     hfuzz = '2pt'
92     samepage = False
93     def __init__(self, *args, **kwargs):
94         super().__init__(*args, **kwargs)
95         self.gobble = abs(int(self.gobble))
96         self.tabsize = abs(int(self.tabsize))
97         if self.firstnumber != 'auto':
98             self.firstnumber = abs(int(self.firstnumber))
99         self.stepnumber = abs(int(self.stepnumber))
100         self.numberblanklines = self.ensure_bool(self.numberblanklines)
101         self.resetmargins = self.ensure_bool(self.resetmargins)
102         self.samepage = self.ensure_bool(self.samepage)

```

3.4 Argumentsclass

```

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

```

```

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

```

4 Controller main class

```

113 class Controller:

```

4.1 Static methods

object_hook Helper for json parsing.

```

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

```

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

```

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

```

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

```

self.parser The correctly set up argparse instance.

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

```

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

```



```

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

```

4.3 Methods

4.3.1 __init__

__init__ Constructor. Reads the command line arguments.

```

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

```

```

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

```

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.

```

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

```

```

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

```

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

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

4.4 Main entry

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

```

\backslash CDR_tag_inherit:cn \backslash CDR_tag_inherit:cf	\backslash CDR_tag_inherit:cn { \langle child name \rangle } { \langle parent names comma list \rangle } Set the parents of \langle child name \rangle to the given list.
--	--

```

333 \cs_new:Npn \CDR_tag_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_tag_inherit:cf {
340   \exp_args:Nnf \CDR_tag_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 }

```

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.

```

349 \AddToHook { begindocument/before } {
350   \IfFileExists {./\jobname.aux} {} {
351     \lua_now:n {CDR:cache_clean_all()}
352   }
353 }

```

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

```

354 \AddToHook { enddocument/end } {
355   \lua_now:n {CDR:cache_clean_unused()}
356 }

```

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>*.

```

357 \cs_new:Npn \CDR_clist_map_inline:Nnn #1 #2 {
358   \clist_if_empty:NTF #1 {
359     #2
360     \use_none:n
361   } {
362     \clist_map_inline:Nn #1
363   }
364 }
```

\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.

```

365 \prg_new_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
366   \PackageError
367     { coder }
368     { Conditional~not~available }
369 }
```

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

```

370 \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 expanded when used as `n` argument of `!keys` functions.

```

371 \cs_set:Npn \CDR_tag_module:n #1 {
372   \str_if_eq:nnTF { #1 } { .. } {
373     \c_CDR_Tags
374   } {
375     \tl_if_empty:nnTF { #1 } { \c_CDR_Tags / tag } { \c_CDR_Tags / tag / #1 }
376   }
377 }
```

`\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`.

```

378 \cs_new:Npn \CDR_tag_keys_define:nn #1 {
379   \exp_args:Nf
380   \keys_define:nn { \CDR_tag_module:n { #1 } }
381 }
```

`\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.

```

382 \prg_new_conditional:Nnn \CDR_tag_keys_if_exist:nn { p, T, F, TF } {
383   \exp_args:Nf
384   \keys_if_exist:nnTF { \CDR_tag_module:n { #1 } } { #2 } {
385     \prg_return_true:
386   } {
387     \prg_return_false:
388   }
389 }
```

`\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`.

```

390 \cs_new_protected:Npn \CDR_tag_keys_set:nn #1 {
391   \exp_args:Nf
392   \keys_set:nn { \CDR_tag_module:n { #1 } }
393 }
394 \cs_generate_variant:Nn \CDR_tag_keys_set:nn { nV }
```

9.1.1 Handling unknown tags

While using `\keys_set:nn` and variants, each time a full key path matching the pattern `\c_CDR_tag/⟨tag name⟩/⟨relative key path⟩` is not recognized, we assume that

the client implicitly wants a tag with the given $\langle \text{tag name} \rangle$ to be defined. For that purpose, we collect unknown keys with $\backslash \text{keys_set_known:nnnN}$ then process them to find each $\langle \text{tag name} \rangle$ and define the new tag accordingly. A similar situation occurs for display engine options where the full key path reads $\backslash \text{c_CDR_tag} / \langle \text{tag name} \rangle / \langle \text{engine name} \rangle$ engine options where $\langle \text{engine name} \rangle$ is not known in advance.

$\backslash \text{CDR_tag_keys_inherit:nn}$ $\backslash \text{CDR_tag_keys_inherit:nn} \{ \langle \text{tag name} \rangle \} \{ \langle \text{parents comma list} \rangle \}$

Set the inheritance: $\langle \text{tag name} \rangle$ inherits from each parent, which is a tag name.

```

395 \cs_new:Npn \CDR_tag_keys_inherit__:nnn #1 #2 #3 {
396   \keys_define:nn { #1 } { #2 .inherit:n = { #1 / #3 } }
397 }
398 \cs_new:Npn \CDR_tag_keys_inherit_:nnn #1 #2 #3 {
399   \exp_args:Nnx
400   \use:n { \CDR_tag_keys_inherit__:nnn { #1 } { #2 } } {
401     \clist_use:nn { #3 } { ,#1/ }
402   }
403 }
404 \cs_new_protected:Npn \CDR_tag_keys_inherit:nn {
405   \exp_args:Nf
406   \CDR_tag_keys_inherit_:nnn { \CDR_tag_module:n { } }
407 }
```

$\backslash \text{CDR_tag_keys_set_known:nnN}$ $\backslash \text{CDR_tag_keys_set_known:nnN} \{ \langle \text{tag name} \rangle \} \{ \langle \text{key[=value] items} \rangle \} \langle \text{tl var} \rangle$

Wrappers over $\backslash \text{keys_set_known:nnnN}$ where the module is given by $\backslash \text{CDR_tag_module:n} \{ \langle \text{tag name} \rangle \}$. *Implementation detail* the remaining arguments are absorbed by the last macro.

```

408 \cs_new:Npn \CDR_tags_keys_set_known__:nnN #1 #2 {
409   \keys_set_known:nnnN { #1 } { #2 } { #1 }
410 }
411 \cs_new:Npn \CDR_tag_keys_set_known:nnN #1 {
412   \exp_args:Nf
413   \CDR_tags_keys_set_known__:nnN { \CDR_tag_module:n { #1 } }
414 }
415 \cs_generate_variant:Nn \CDR_tag_keys_set_known:nnN { nV }
```

$\backslash \text{c_CDR_provide_regex}$ To parse a $\backslash \text{keys}$ full key path.

```

416 \tl_set:Nn \l_CDR_tl { /([~/*])(?:/(.))*? $ } \use_none:n { $ }
417 \exp_args:NNf
418 \tl_put_left:Nn \l_CDR_tl { \CDR_tag_module:n { } }
419 \tl_put_left:Nn \l_CDR_tl { ^ }
420 \exp_args:NNV
421 \regex_const:Nn \c_CDR_provide_regex \l_CDR_tl
```

(End definition for $\backslash \text{c_CDR_provide_regex}$. 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*.

```
422 \regex_const:Nn \c_CDR_engine_regex { ^[~/]+\sengine\soptions$ } \use_none:n { $ }
423 \cs_new_protected_nopar:Npn \CDR_tag_provide:n #1 {
424 \CDR@Debug { \string\CDR_tag_provide:n: #1 }
425   \exp_args:Nnf
426   \regex_extract_once:NnNTF \c_CDR_provide_regex {
427     \CDR_tag_module:n { .. } / #1
428   } \l_CDR_seq {
429     \tl_set:Nx \l_CDR_tl { \seq_item:Nn \l_CDR_seq 3 }
430     \exp_args:Nx
431     \clist_map_inline:nn {
432       \seq_item:Nn \l_CDR_seq 2
433     } {
434       \CDR_tag_keys_if_exist:nnF { } { ##1 } {
435         \CDR_tag_keys_inherit:nn { ##1 } {
436           __pygments, __pygments.block,
437           default.block, default.code, default, __tags,
438           __fancyvrb, __fancyvrb.block, __fancyvrb.frame, __fancyvrb.number, __fancyvrb.all,
439         }
440         \CDR_tag_keys_define:nn { } {
441           ##1 .code:n = \CDR_tag_keys_set:nn { ##1 } { #####1 },
442           ##1 .value_required:n = true,
443         }
444 \CDR@Debug{\string\CDR_tag_provide:n \CDR_tag_module:n {##1} = ...}
445       }
446       \exp_args:NnV
447       \CDR_tag_keys_if_exist:nnF { ##1 } \l_CDR_tl {
448         \exp_args:NNV
449         \regex_match:NnT \c_CDR_engine_regex
450           \l_CDR_tl {
451             \exp_args:Nnf
452             \CDR_tag_keys_define:nn { ##1 } {
453               \use:n { \l_CDR_tl } .code:n = \CDR_tag_set:n { #####1 },
454             }
455             \exp_args:Nnf
456             \CDR_tag_keys_define:nn { ##1 } {
457               \use:n { \l_CDR_tl } .value_required:n = true,
458             }
459 \CDR@Debug{\string\CDR_tag_provide:n: \CDR_tag_module:n { ##1 } / \l_CDR_tl = ...}
460           }
461         }
462       }
463     } {
464       \regex_match:NnT \c_CDR_engine_regex { #1 } {
465         \CDR_tag_keys_define:nn { default } {
```

```

466     #1 .code:n = \CDR_tag_set:n { ##1 },
467     #1 .value_required:n = true,
468   }
469 \CDR@Debug{\string\CDR_tag_provide:n.C:\CDR_tag_module:n { default } / #1 = ...}
470   }
471 }
472 }
473 \cs_new:Npn \CDR_tag_provide:nn #1 #2 {
474   \CDR_tag_provide:n { #1 }
475 }
476 \cs_new:Npn \CDR_tag_provide_from_kv:n {
477   \keyval_parse:nnn {
478     \CDR_tag_provide:n
479   } {
480     \CDR_tag_provide:nn
481   }
482 }
483 \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 <code>pygments</code> is available, <code><false code></code> otherwise. <i>Implementation detail:</i> we define the conditionals and set them afterwards.
--	--

```

484 \sys_get_shell:nnN {which-pygmentize} {} \l_CDR_tl
485 \prg_new_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } { }
486 \tl_if_in:NnTF \l_CDR_tl { pygmentize } {
487   \prg_set_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } {
488     \prg_return_true:
489   }
490 } {
491   \prg_set_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } {
492     \prg_return_false:
493   }
494 }


```

9.2.2 __pygments l3keys module

```

495 \CDR_tag_keys_define:nn { __pygments } {

```

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

```

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

```


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

```
498 pygments .code:n = \CDR_tag_boolean_set:x { #1 },
499 pygments .default:n = true,
```

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

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

● **commandprefix**=*<text>* The L^AT_EX commands used to produce colored output are constructed using this prefix and some letters. Initially Py.

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

● **mathescape**[*=true|false*] If set to true, enables L^AT_EX math mode escape in comments. That is, \$...\$ inside a comment will trigger math mode. Initially false.

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

● **escapeinside**=*<before>**<after>* If set to a string of length 2, enables escaping to L^AT_EX. Text delimited by these 2 characters is read as L^AT_EX 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.

```
506 escapeinside .code:n = \CDR_tag_set:,
507 escapeinside .value_required:n = true,
```

● **__initialize** Initializer.

```
508 __initialize .meta:n = {
509   lang = tex,
510   pygments = \CDR_has_pygments:TF { true } { false },
511   style = default,
512   commandprefix = PY,
513   mathescape = false,
514   escapeinside = ,
515 },
516 __initialize .value_forbidden:n = true,

517 }
518 \AtBeginDocument{
519   \CDR_tag_keys_set:nn { __pygments } { __initialize }
520 }
```

9.2.3 `--pygments.block l3keys` module

```
521 \CDR_tag_keys_define:nn { --pygments.block } {
```

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

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

- **--initialize** Initializer.

```
524   --initialize .meta:n = {
525     texcomments = false,
526   },
527   --initialize .value_forbidden:n = true,

528 }
529 \AtBeginDocument{
530   \CDR_tag_keys_set:nn { --pygments.block } { --initialize }
531 }
```

9.3 Specific to coder

9.3.1 `default l3keys` module

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

Keys are:

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

```
533   format .code:n = \CDR_tag_set:,
534   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.

```
535   cache .code:n = \CDR_tag_boolean_set:x { #1 },
536   cache .default:n = true,
```

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

```
537   debug .code:n = \CDR_tag_boolean_set:x { #1 },
538   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.

```

539 post~processor .code:n = \CDR_tag_set:,
540 post~processor .value_required:n = true,

```

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

```

541 parskip .code:n = \CDR_tag_set:,
542 parskip .value_required:n = true,

```

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

```

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

```

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

```

545 default~engine~options .code:n = \CDR_tag_set:,
546 default~engine~options .value_required:n = true,

```

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

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

```

547 __initialize .meta:n = {
548   format = ,
549   cache = true,
550   debug = false,
551   post~processor = ,
552   parskip = \the\parskip,
553   engine = default,
554   default~engine~options = ,
555 },
556 __initialize .value_forbidden:n = true,

557 }
558 \AtBeginDocument{
559   \CDR_tag_keys_set:nn { default } { __initialize }
560 }

```

9.3.2 default.code l3keys module

Void for the moment.

```

561 \CDR_tag_keys_define:nn { default.code } {

```

Known keys include:

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

```

562 __initialize .meta:n = {
563 },
564 __initialize .value_forbidden:n = true,

565 }
566 \AtBeginDocument{
567   \CDR_tag_keys_set:nn { default.code } { __initialize }
568 }

```

9.3.3 __tags l3keys module

```

569 \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.

```

570 tags .code:n = {
571   \clist_set:Nn \l_CDR_clist { #1 }
572   \clist_remove_duplicates:N \l_CDR_clist
573   \exp_args:NV
574   \CDR_tag_set:n \l_CDR_clist
575 },
576 tags .value_required:n = true,

```

● **__initialize** Initialization.

```

577 __initialize .meta:n = {
578   tags = ,
579 },
580 __initialize .value_forbidden:n = true,

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

```

9.3.4 default.block l3keys module

```

585 \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.

```

586 tags~format .code:n = \CDR_tag_set:,
587 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).

```

588 numbers~format .code:n = \CDR_tag_set:,
589 numbers~format .value_required:n = true,

```

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

```

590 show~tags .choices:nn =
591   { none, left, right, numbers, mirror }
592   { \CDR_tag_choices_set: },
593 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.

```

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

```

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

```

596 use~margin .code:n = \CDR_tag_boolean_set:x { #1 },
597 use~margin .default:n = true,

```

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

```

598 blockskip .code:n = \CDR_tag_set:,
599 blockskip .value_required:n = true,

```

● **__initialize** Initialization.

```

600 __initialize .meta:n = {
601   show~tags = numbers,
602   only~top = true,
603   use~margin = true,
604   numbers~format = {
605     \sffamily
606     \scriptsize
607     \color{gray}
608   },
609   tags~format = {
610     \bfseries
611   },
612   blockskip = \topsep,
613 },
614 __initialize .value_forbidden:n = true,
615 }
616 \AtBeginDocument{
617   \CDR_tag_keys_set:nn { default.block } { __initialize }
618 }

```

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

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

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

```
620   formatcom .code:n = \CDR_tag_set:,
621   formatcom .value_required:n = true,
```

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

```
622   fontfamily .code:n = \CDR_tag_set:,
623   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.

```
624   fontsize .code:n = \CDR_tag_set:,
625   fontsize .value_required:n = true,
```

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

```
626   fontshape .code:n = \CDR_tag_set:,
627   fontshape .value_required:n = true,
```

● **fontseries**=*<series name>* \LaTeX font series to use. Initially `auto`: the same as the current font.

```
628   fontseries .code:n = \CDR_tag_set:,
629   fontseries .value_required:n = true,
```

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

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

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

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

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

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

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

```
636 tabsize .code:n = \CDR_tag_set:,
637 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.

```
638 defineactive .code:n = \CDR_tag_set:,
639 defineactive .value_required:n = true,
```

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

```
640 relabel .code:n = \CDR_tag_set:,
641 relabel .value_required:n = true,
```

✅ **__initialize** Initialization.

```
642 __initialize .meta:n = {
643   formatcom = ,
644   fontfamily = tt,
645   fontsize = auto,
646   fontseries = auto,
647   fontshape = auto,
648   showspaces = false,
649   showtabs = false,
650   obeytabs = false,
651   tabsize = 2,
652   defineactive = ,
653   relabel = ,
654 },
655 __initialize .value_forbidden:n = true,

656 }
657 \AtBeginDocument{
658   \CDR_tag_keys_set:nn { __fancyvrb } { __initialize }
659 }
```

9.4.2 __fancyvrb.frame l3keys module

Block specific options, frame related.

```
660 \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.

```
661 frame .choices:nn =
662   { none, leftline, topline, bottomline, lines, single }
663   { \CDR_tag_choices_set: },
```

🔴 **framerule**= $\langle dimension \rangle$ width of the rule of the frame if any. Initially 0.4pt.

```
664 framerule .code:n = \CDR_tag_set:,
665 framerule .value_required:n = true,
```

🔴 **framesep**= $\langle dimension \rangle$ width of the gap between the frame (if any) and the text. Initially `\fboxsep`.

```
666 framesep .code:n = \CDR_tag_set:,
667 framesep .value_required:n = true,
```

🔴 **rulecolor**= $\langle color command \rangle$ color of the frame rule, expressed in the standard L^AT_EX way. Initially black.

```
668 rulecolor .code:n = \CDR_tag_set:,
669 rulecolor .value_required:n = true,
```

🔴 **rulecolor**= $\langle color command \rangle$ color used to fill the space between the frame and the text (its thickness is given by **framesep**). Initially empty.

```
670 fillcolor .code:n = \CDR_tag_set:,
671 fillcolor .value_required:n = true,
```

🔴 **label**={ $[\langle top string \rangle] \langle string \rangle$ } 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.

```
672 label .code:n = \CDR_tag_set:,
673 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.

```
674 labelposition .choices:nn =
675   { none, topline, bottomline, all }
676   { \CDR_tag_choices_set: },
```

✅ **__initialize** Initialization.

```
677 __initialize .meta:n = {
678   frame = none,
679   framerule = 0.4pt,
680   framesep = \fboxsep,
681   rulecolor = black,
682   fillcolor = ,
683   label = ,
684   labelposition = none,% auto?
685 },
686 __initialize .value_forbidden:n = true,
```



```

687 }
688 \AtBeginDocument{
689   \CDR_tag_keys_set:nn { __fancyvrb.frame } { __initialize }
690 }

```

9.4.3 `__fancyvrb.block l3keys` module

Block specific options, except numbering.

```

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

```

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

```

693   commentchar .code:n = \CDR_tag_set:,
694   commentchar .value_required:n = true,

```

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

```

695   gobble .choices:nn = {
696     0,1,2,3,4,5,6,7,8,9
697   } {
698     \CDR_tag_choices_set:
699   },

```

- **baselinestretch**=`auto`|*<dimension>* value to give to the usual `\baselinestretch` L^AT_EX parameter. Initially `auto`: its current value just before the verbatim command.

```

700   baselinestretch .code:n = \CDR_tag_set:,
701   baselinestretch .value_required:n = true,

```

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

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

```

702   xleftmargin .code:n = \CDR_tag_set:,
703   xleftmargin .value_required:n = true,

```

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

```

704   xrightmargin .code:n = \CDR_tag_set:,
705   xrightmargin .value_required:n = true,

```

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

```

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

```

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

```

708 hfuzz .code:n = \CDR_tag_set:,
709 hfuzz .value_required:n = true,

```

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

```

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

```

✅ **__initialize** Initialization.

```

712 __initialize .meta:n = {
713   commentchar = ,
714   gobble = 0,
715   baselinestretch = auto,
716   resetmargins = true,
717   xleftmargin = 0pt,
718   xrightmargin = 0pt,
719   hfuzz = 2pt,
720   samepage = false,
721 },
722 __initialize .value_forbidden:n = true,
723 }
724 \AtBeginDocument{
725   \CDR_tag_keys_set:nn { __fancyvrb.block } { __initialize }
726 }

```

9.4.4 `__fancyvrb.number l3keys` module

Block line numbering.

```

727 \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.

```

728 numbers .choices:nn =
729   { none, left, right }
730   { \CDR_tag_choices_set: },

```

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

```

731 numbersep .code:n = \CDR_tag_set:,
732 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.

```
733 firstnumber .code:n = {
734   \regex_match:NnTF \c_CDR_integer_regex { #1 } {
735     \CDR_tag_set:
736   } {
737     \str_case:nnF { #1 } {
738       { auto } { \CDR_tag_set: }
739       { last } { \CDR_tag_set: }
740     } {
741       \PackageWarning
742         { CDR }
743         { Value~‘#1’~not~in~auto,~last. }
744     }
745   },
746   firstnumber .value_required:n = true,
```

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

```
748 stepnumber .code:n = \CDR_tag_set:,
749 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.

```
750 numberblanklines .code:n = \CDR_tag_boolean_set:x { #1 },
751 numberblanklines .default:n = true,
```

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

```
752 firstline .code:n = \CDR_tag_set:,
753 firstline .value_required:n = true,
```

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

```
754 lastline .code:n = \CDR_tag_set:,
755 lastline .value_required:n = true,
```

- ✓ **__initialize** Initialization.

```
756 __initialize .meta:n = {
757   numbers = left,
758   numbersep = 1ex,
759   firstnumber = auto,
760   stepnumber = 1,
761   numberblanklines = true,
```

```

762     firstline = ,
763     lastline = ,
764 },
765 __initialize .value_forbidden:n = true,

766 }
767 \AtBeginDocument{
768   \CDR_tag_keys_set:nn { __fancyvrb.number } { __initialize }
769 }

```


9.4.5 __fancyvrb.all l3keys module

Options available when pygments is not used.

```

770 \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.

```

771   commandchars .code:n = \CDR_tag_set:,
772   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.

```

773   codes .code:n = \CDR_tag_set:,
774   codes .value_required:n = true,

```

-  **__initialize** Initialization.

```

775   __initialize .meta:n = {
776     commandchars = ,
777     codes = ,
778   },
779   __initialize .value_forbidden:n = true,

780 }
781 \AtBeginDocument{
782   \CDR_tag_keys_set:nn { __fancyvrb.all } { __initialize }
783 }

```

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

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

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

```
785   only~description .choices:nn = { false, true, {} } {  
786     \int_compare:nNnTF \l_keys_choice_int = 1 {  
787       \prg_set_conditional:Nnn \CDR_if_only_description: { p, T, F, TF } { \prg_return_true: }  
788     } {  
789       \prg_set_conditional:Nnn \CDR_if_only_description: { p, T, F, TF } { \prg_return_false: }  
790     }  
791   },  
792   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.

```
793   python~path .code:n = {  
794     \str_set:Nn \l_CDR_str { #1 }  
795     \lua_now:n { CDR:set_python_path('l_CDR_str') }  
796   },  
  
797 }
```

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.

```
798 \cs_new:Npn \CDR_set_preflight:n #1 { }  
  
799 \NewDocumentCommand \CDRSet { m } {  
800   \CDR@Debug{\string\CDRSet}  
801   \CDR_set_preflight:n { #1 }  
802   \keys_set_known:nnnN { CDR@Set } { #1 } { CDR@Set } \l_CDR_kv_clist  
803   \clist_map_inline:nn {  
804     __pygments, __pygments.block,  
805     __tags, default.block, default.code, default,  
806     __fancyvrb, __fancyvrb.frame, __fancyvrb.block, __fancyvrb.number, __fancyvrb.all  
807   } {
```

```

808 \CDR_tag_keys_set_known:nVN { ##1 } \l_CDR_kv_clist \l_CDR_kv_clist
809 \CDR@Debug{ Debug.CDRSet.1:##1/\l_CDR_kv_clist/ }
810 }
811 \CDR_tag_keys_set_known:nVN { .. } \l_CDR_kv_clist \l_CDR_kv_clist
812 \CDR@Debug{ Debug.CDRSet.2:\CDR_tag_module:n { .. }//\l_CDR_kv_clist/ }
813 \CDR_tag_provide_from_kv:V \l_CDR_kv_clist
814 \CDR@Debug{ Debug.CDRSet.2a:\CDR_tag_module:n { .. }//\l_CDR_kv_clist/ }
815 \CDR_tag_keys_set_known:nVN { .. } \l_CDR_kv_clist \l_CDR_kv_clist
816 \CDR@Debug{ Debug.CDRSet.3:\CDR_tag_module:n { .. }//\l_CDR_kv_clist/ }
817 \CDR_tag_keys_set:nV { default } \l_CDR_kv_clist
818 \CDR@Debug{ Debug.CDRSet.4:\CDR_tag_module:n { default } /\l_CDR_kv_clist/ }
819 \keys_define:nn { CDR@Set@tags } {
820   tags .code:n = {
821     \clist_set:Nn \g_CDR_tags_clist { ##1 }
822     \clist_remove_duplicates:N \g_CDR_tags_clist
823   },
824 }
825 \keys_set_known:nn { CDR@Set@tags } { #1 }
826 }

```

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.

```

827 \cs_new:Npn \CDR_export_get_path:cc #1 #2 {
828   CDR @ export @ get @ #1 / #2
829 }

```

`\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 `TEX` group level.

```

830 \cs_new_protected:Npn \CDR_export_set:ccn #1 #2 #3 {
831   \cs_set:cpn { \CDR_export_get_path:cc { #1 } { #2 } } { \exp_not:n { #3 } }
832 }
833 \cs_new_protected:Npn \CDR_export_set:Vcn #1 {
834   \exp_args:NV
835   \CDR_export_set:ccn { #1 }
836 }
837 \cs_new_protected:Npn \CDR_export_set:VcV #1 #2 #3 {
838   \exp_args:NnV

```

```

839 \use:n {
840   \exp_args:NV \CDR_export_set:ccn #1 { #2 }
841 } #3
842 }

```

\CDR_export_if_exist:ccTF \star \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.

```

843 \prg_new_conditional:Nnn \CDR_export_if_exist:cc { p, T, F, TF } {
844   \cs_if_exist:ctf { \CDR_export_get_path:cc { #1 } { #2 } } {
845     \prg_return_true:
846   } {
847     \prg_return_false:
848   }
849 }

```

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

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

```

850 \cs_new:Npn \CDR_export_get:cc #1 #2 {
851   \CDR_export_if_exist:ccT { #1 } { #2 } {
852     \use:c { \CDR_export_get_path:cc { #1 } { #2 } }
853   }
854 }

```

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

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

```

855 \prg_new_protected_conditional:Nnn \CDR_export_get:ccN { T, F, TF } {
856   \CDR_export_if_exist:ccTF { #1 } { #2 } {
857     \tl_set:Nx #3 { \CDR_export_get:cc { #1 } { #2 } }
858     \prg_return_true:
859   } {
860     \prg_return_false:
861   }
862 }

```

11.2 Storage

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

```

863 \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.

864 \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.

865 \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.

866 \keys_define:nn { CDR@Export } {

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

867 file .tl_set:N = \l_CDR_file_tl,

868 file .value_required:n = true,

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

869 tags .code:n = {

870 \clist_set:Nn \l_CDR_clist { #1 }

871 \clist_remove_duplicates:N \l_CDR_clist

872 \prop_put:NVV \l_CDR_export_prop \l_keys_key_str \l_CDR_clist

873 },

874 tags .value_required:n = true,

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

875 lang .code:n = {

876 \prop_put:NVN \l_CDR_export_prop \l_keys_key_str { #1 }

877 },

878 lang .value_required:n = true,

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

879 preamble .code:n = {

880 \prop_put:NVN \l_CDR_export_prop \l_keys_key_str { #1 }

881 },

882 preamble .value_required:n = true,

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

883 postamble .code:n = {

884 \prop_put:NVN \l_CDR_export_prop \l_keys_key_str { #1 }

885 },


886 postamble .value_required:n = true,

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


```

887   raw .choices:nn = { false, true, {} } {
888     \prop_put:NVx \l_CDR_export_prop \l_keys_key_str {
889       \int_compare:nNnTF
890         \l_keys_choice_int = 1 { false } { true }
891     }
892 },


```

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

```

893   once .choices:nn = { false, true, {} } {
894     \prop_put:NVx \l_CDR_export_prop \l_keys_key_str {
895       \int_compare:nNnTF
896         \l_keys_choice_int = 1 { false } { true }
897     }
898 },

```

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

```

899   __initialize .meta:n = {
900     __initialize_prop = #1,
901     file =,
902     tags =,
903     lang = tex,
904     preamble =,
905     postamble =,
906     raw = false,
907     once = true,
908   },
909   __initialize .default:n = \l_CDR_export_prop,

```

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

```

910   __initialize_prop .code:n = \prop_clear:N #1,
911   __initialize_prop .value_required:n = true,
912 }

```

11.4 Implementation

```

913 \NewDocumentCommand \CDRExport { m } {
914   \keys_set:nn { CDR@Export } { __initialize }
915   \keys_set:nn { CDR@Export } { #1 }
916   \tl_if_empty:NTF \l_CDR_file_tl {
917     \PackageWarning
918       { coder }
919       { Missing~export~key~‘file’ }
920   } {
921     \CDR_export_set:VcV \l_CDR_file_tl { file } \l_CDR_file_tl
922     \prop_map_inline:Nn \l_CDR_export_prop {
923       \CDR_export_set:Vcn \l_CDR_file_tl { ##1 } { ##2 }
924     }

```

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.

```

925 \prop_get:NnNTF \l_CDR_export_prop { tags } \l_CDR_clist {
926   \tl_if_empty:NTF \l_CDR_clist {
927     \PackageWarning
928       { coder }
929     { Missing~export~key~‘tags’ }
930   } {
931     \clist_set_eq:NN \g_CDR_tags_clist \l_CDR_clist
932     \clist_remove_duplicates:N \g_CDR_tags_clist
933     \clist_put_left:NV \g_CDR_all_tags_clist \l_CDR_clist
934     \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`.

```

935   \exp_args:NV
936   \CDR_export_get:ccNT \l_CDR_file_tl { lang } \l_CDR_tl {
937     \clist_map_inline:Nn \g_CDR_tags_clist {
938       \CDR_tag_set:ccV { ##1 } { lang } \l_CDR_tl
939     }
940   }
941 }
942 \seq_put_left:NV \g_CDR_export_seq \l_CDR_file_tl
943 } {
944   \PackageWarning
945     { coder }
946   { Missing~export~key~‘tags’ }
947 }
948 }
949 }

```

Files are created at the end of the typesetting process.

```

950 \AddToHook { enddocument / end } {
951   \seq_map_inline:Nn \g_CDR_export_seq {
952     \str_set:Nx \l_CDR_str { #1 }
953     \lua_now:n { CDR:export_file('l_CDR_str') }
954     \clist_map_inline:nn {
955       tags, raw, once, preamble, postamble
956     } {
957       \CDR_export_get:ccNT { #1 } { ##1 } \l_CDR_tl {
958         \exp_args:NNx
959         \str_set:Nn \l_CDR_str { \l_CDR_tl }
960         \lua_now:n {
961           CDR:export_file_info('##1','l_CDR_str')
962         }
963       }
964     }
965     \lua_now:n { CDR:export_complete() }
966   }
967 }

```

12 Style

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

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

Define the definitions for the given *⟨pygments style name⟩*.

```

968 \cs_set:Npn \CDR@StyleDefine #1 {
969   \tl_gset:cn { g_CDR@Style/#1 }
970 }
```

<code>\CDR@StyleUse</code>	<code>\CDR@StyleUse {⟨<i>pygments style name</i>⟩}</code>
<code>CDR@StyleUseTag</code>	<code>\CDR@StyleUseTag</code>

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.

```

971 \cs_set:Npn \CDR@StyleUse #1 {
972   \tl_use:c { g_CDR@Style/#1 }
973 }
974 \cs_set:Npn \CDR@StyleUseTag {
975   \CDR@StyleUse { \CDR_tag_get:c { style } }
976 }
```

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

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

```

977 \prg_new_conditional:Nnn \CDR@StyleIfExist:c { TF } {
978   \tl_if_exist:cTF { g_CDR@Style/#1 } {
979     \prg_return_true:
980   } {
981     \prg_return_false:
982   }
983 }
984 \cs_set_eq:NN \CDR@StyleIfExist \CDR@StyleIfExist:cTF
```

13 Creating display engines

13.1 Utilities

<code>\CDRCode_engine:c</code>	★	<code>\CDRCodeengine:c {⟨<i>engine name</i>⟩}</code>
<code>\CDRCode_engine:V</code>	★	<code>\CDRBlock_engine:c {⟨<i>engine name</i>⟩}</code>
<code>\CDRBlock_engine:c</code>	★	<code>\CDRCode_engine:c</code> builds a command sequence name based on <i>⟨engine name⟩</i> . <code>\CDRBlock_engine:c</code> builds an environment name based on <i>⟨engine name⟩</i> .
<code>\CDRBlock_engine:V</code>	★	

```

985 \cs_new:Npn \CDRCode_engine:c #1 {
986   CDR@colored/code/#1:nn
987 }
988 \cs_new:Npn \CDRBlock_engine:c #1 {
989   CDR@colored/block/#1
990 }
991 \cs_new:Npn \CDRCode_engine:V {
992   \exp_args:NV \CDRCode_engine:c
993 }
994 \cs_new:Npn \CDRBlock_engine:V {
995   \exp_args:NV \CDRBlock_engine:c
996 }

```

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

```

997 \cs_new:Npn \CDRCode_options:c #1 {
998   CDR@colored/code~options/#1:nn
999 }
1000 \cs_new:Npn \CDRBlock_options:c #1 {
1001   CDR@colored/block~options/#1
1002 }
1003 \cs_new:Npn \CDRCode_options:V {
1004   \exp_args:NV \CDRCode_options:c
1005 }
1006 \cs_new:Npn \CDRBlock_options:V {
1007   \exp_args:NV \CDRBlock_options:c
1008 }

```

\l_CDR_engine_tl Storage for an engine name.

```
1009 \tl_new:N \l_CDR_engine_tl
```

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

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

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

13.2 Implementation

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

<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.

```

1010 \NewDocumentCommand \CDRCodeEngineNew { mm } {
1011   \exp_args:Nx
1012   \tl_if_empty:nTF { #1 } {
1013     \PackageWarning
1014       { coder }
1015       { The~engine~cannot~be~void. }
1016   } {
1017     \cs_new:cpn { \CDRCode_engine:c {#1} } ##1 ##2 {
1018       \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
1019       #2
1020     }
1021     \ignorespaces
1022   }
1023 }

1024 \NewDocumentCommand \CDRCodeEngineRenew { mm } {
1025   \exp_args:Nx
1026   \tl_if_empty:nTF { #1 } {
1027     \PackageWarning
1028       { coder }
1029       { The~engine~cannot~be~void. }
1030     \use_none:n
1031   } {
1032     \cs_if_exist:cTF { \CDRCode_engine:c { #1 } } {
1033       \cs_set:cpn { \CDRCode_engine:c { #1 } } ##1 ##2 {
1034         \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
1035         #2
1036       }
1037     } {
1038       \PackageWarning
1039         { coder }
1040         { No~code~engine~#1.}
1041     }
1042     \ignorespaces
1043   }
1044 }

```

\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.

```

1045 \cs_new_protected:Npn \CDR@CodeEngineApply {
1046   \CDR_if_code_engine:cF { \CDR_tag_get:c { engine } } {
1047     \PackageError
1048       { coder }
1049       { \CDR_tag_get:c { engine }~code~engine~unknown,~replaced~by~‘default’ }
1050       {See~\CDRCodeEngineNew~in~the~coder~manual}
1051     \CDR_tag_set:cn { engine } { default }
1052   }
1053   \CDR_tag_get:c { format }
1054   \exp_args:Nnx
1055   \use:c { \CDRCode_engine:c { \CDR_tag_get:c { engine } } } {

```

```

1056 \CDR_tag_get:c { \CDR_tag_get:c { engine }~engine~options },
1057 \CDR_tag_get:c { engine~options }
1058 }
1059 }

```

\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 third argument is parsed by \NewDocumentEnvironment.

```

1060 \NewDocumentCommand \CDRBlockEngineNew { mO{}m } {
1061   \cs_set:cpn { \CDRBlock_options:c { #1 } } { #2 }
1062   \NewDocumentEnvironment { \CDRBlock_engine:c { #1 } } { m } {
1063     \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
1064     #3
1065   }
1066 }

1067 \NewDocumentCommand \CDRBlockEngineRenew { mm } {
1068   \tl_if_empty:nTF { #1 } {
1069     \PackageWarning
1070       { coder }
1071       { The~engine~cannot~be~void. }
1072     \use_none:n
1073   } {
1074     \RenewDocumentEnvironment { \CDRBlock_engine:c { #1 } } { m } {
1075       \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
1076       #2
1077     }
1078   }
1079 }

```

\CDRBlock_engine_begin:	\CDR@BlockEngineBegin
\CDR@BlockEngineEnd	\CDR@BlockEngineEnd

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.

```

1080 \cs_new:Npn \CDRBlock_engine_begin: {
1081   \CDR_if_block_engine:cF { \CDR_tag_get:c { engine } } {
1082     \PackageError
1083       { coder }
1084       { \CDR_tag_get:c { engine }~block~engine~unknown,~replaced~by~‘default’ }
1085       {See~\CDRBlockEngineNew~in~the~coder~manual}
1086     \CDR_tag_set:cn { engine } { default }
1087   }

```

```

1088 \exp_args:Nnx
1089 \use:c { \CDRBlock_engine:c \CDR_tag_get:c { engine } } {
1090   \CDR_tag_get:c { \CDR_tag_get:c { engine }~engine~options },
1091   \CDR_tag_get:c { engine~options },
1092 }
1093 }
1094 \cs_new:Npn \CDRBlock_engine_end: {
1095   \use:c { end \CDRBlock_engine:c \CDR_tag_get:c { engine } }
1096 }
1097 % \begin{MacroCode}
1098 %
1099 % \subsection{Conditionals}
1100 %
1101 % \begin{function}[EXP,TF]{\CDR_if_code_engine:c}
1102 % \begin{syntax}
1103 % \cs{CDR_if_code_engine:cTF} \Arg{engine name} \Arg{true code} \Arg{false code}
1104 % \end{syntax}
1105 % If there exists a code engine with the given \metatt{engine name},
1106 % execute \metatt{true code}.
1107 % Otherwise, execute \metatt{false code}.
1108 % \end{function}
1109 % \begin{MacroCode}[OK]
1110 \prg_new_conditional:Nnn \CDR_if_code_engine:c { p, T, F, TF } {
1111   \cs_if_exist:cTF { \CDRCode_engine:c { #1 } } {
1112     \prg_return_true:
1113   } {
1114     \prg_return_false:
1115   }
1116 }
1117 \prg_new_conditional:Nnn \CDR_if_code_engine:V { p, T, F, TF } {
1118   \cs_if_exist:cTF { \CDRCode_engine:V #1 } {
1119     \prg_return_true:
1120   } {
1121     \prg_return_false:
1122   }
1123 }

```

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

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

```

1124 \prg_new_conditional:Nnn \CDR_if_block_engine:c { p, T, F, TF } {
1125   \cs_if_exist:cTF { \CDRBlock_engine:c { #1 } } {
1126     \prg_return_true:
1127   } {
1128     \prg_return_false:
1129   }
1130 }
1131 \prg_new_conditional:Nnn \CDR_if_block_engine:V { p, T, F, TF } {
1132   \cs_if_exist:cTF { \CDRBlock_engine:V #1 } {
1133     \prg_return_true:
1134   } {
1135     \prg_return_false:

```

```

1136 }
1137 }

```

13.3 Default code engine

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

```

1138 \CDRCodeEngineNew { default } { #2 }

```

13.4 efbox code engine

```

1139 \AtBeginDocument {
1140   \ifpackageloaded{efbox} {
1141     \CDRCodeEngineNew {efbox} {
1142       \efbox[#1]{#2}
1143     }
1144   } {}
1145 }

```

13.5 Block mode default engine

```

1146 \CDRBlockEngineNew {default} {
1147 } {
1148 }

```

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.

```

1149 \cs_new:Npn \CDR@DefineSp {
1150   \CDR_tag_if_truthy:cTF { showspaces } {
1151     \cs_set:Npn \CDR@Sp {{\FancyVerbSpace}}
1152   } {
1153     \cs_set_eq:NN \CDR@Sp \space
1154   }
1155 }

```

\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.

```
1156 \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.

```
1157 \CDR_tag_keys_define:nn { __code } {
```

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

```
1158   tag .tl_set:N = \l_CDR_tag_tl,
1159   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.

```
1160   engine~options .code:n = \CDR_tag_set:,
1161   engine~options .value_required:n = true,
```

🔴 **__initialize** initialize

```
1162   __initialize .meta:n = {
1163     tag = default,
1164     engine~options = ,
1165   },
1166   __initialize .value_forbidden:n = true,
1167 }
```

14.4 Implementation

`\CDRCodeformat:` `\CDRCodeformat:`

Private utility to setup the formatting.

```
1168 \cs_new:Npn \CDR_brace_if_contains_comma:n #1 {
1169   \tl_if_in:nnTF { #1 } { , } { { #1 } } { #1 }
1170 }
1171 \cs_generate_variant:Nn \CDR_brace_if_contains_comma:n { V }
1172 \cs_new:Npn \CDRCodeformat: {
1173   \frenchspacing
1174   \CDR_tag_get:cN { baselinestretch } \l_CDR_tl
1175   \str_if_eq:VnF \l_CDR_tl { auto } {
1176     \exp_args:NNV
1177     \def \baselinestretch \l_CDR_tl
1178   }
1179   \CDR_tag_get:cN { fontfamily } \l_CDR_tl
1180   \str_if_eq:VnT \l_CDR_tl { tt } { \tl_set:Nn \l_CDR_tl { lmtt } }
```

```

1181 \exp_args:NV
1182 \fontfamily \l_CDR_tl
1183 \clist_map_inline:nn { series, shape } {
1184   \CDR_tag_get:cN { font##1 } \l_CDR_tl
1185   \str_if_eq:VnF \l_CDR_tl { auto } {
1186     \exp_args:NnV
1187     \use:c { font##1 } \l_CDR_tl
1188   }
1189 }
1190 \CDR_tag_get:cN { fontsize } \l_CDR_tl
1191 \str_if_eq:VnF \l_CDR_tl { auto } {
1192   \tl_use:N \l_CDR_tl
1193 }
1194 \selectfont
1195 % \@noligs ?? this is in fancyvrb but does not work here as is
1196 }

```

\CDRCode:n \CDRCode:n <delimiter>

Main utility used by \CDRCode.

```

1197 \cs_new:Npn \CDRCode:n #1 {
1198   \CDR_tag_if_truthy:cTF {pygments} {
1199     \cs_set:Npn \CDR@StyleUseTag {
1200       \CDR@StyleUse { \CDR_tag_get:c { style } }
1201       \cs_set_eq:NN \CDR@StyleUseTag \prg_do_nothing:
1202     }
1203     \CDR_tag_keys_inherit:nn { __local } {
1204       __fancyvrb,
1205     }
1206     \CDR_tag_keys_set:nV { __local } \l_CDR_kv_clist
1207     \DefineShortVerb { #1 }
1208     \SaveVerb [
1209       aftersave = {
1210         \exp_args:Nx \UndefineShortVerb { #1 }
1211         \lua_now:n { CDR:highlight_code_setup() }
1212         \CDR_tag_get:cN {lang} \l_CDR_tl
1213         \lua_now:n { CDR:highlight_set_var('lang') }
1214         \CDR_tag_get:cN {cache} \l_CDR_tl
1215         \lua_now:n { CDR:highlight_set_var('cache') }
1216         \CDR_tag_get:cN {debug} \l_CDR_tl
1217         \lua_now:n { CDR:highlight_set_var('debug') }
1218         \CDR_tag_get:cN {style} \l_CDR_tl
1219         \lua_now:n { CDR:highlight_set_var('style') }
1220         \lua_now:n { CDR:highlight_set_var('source', 'FV@SV@CDR@Source') }
1221         \FV@UseKeyValues
1222         \frenchspacing
1223         % \FV@SetupFont Break
1224         \FV@DefineWhiteSpace
1225         \FancyVerbDefineActive
1226         \FancyVerbFormatCom
1227         \CDRCodeformat:
1228         \CDR@DefineSp
1229         \CDR_tag_get:c { format }

```

```

1230     \CDR@CodeEngineApply {
1231         \CDR@StyleIfExist { \l_CDR_tl } {
1232             \CDR@StyleUseTag
1233             \lua_now:n { CDR:highlight_source(false, true) }
1234         } {
1235             \lua_now:n { CDR:highlight_source(true, true) }
1236             \input { \l_CDR_pyg_sty_tl }
1237             \CDR@StyleUseTag
1238         }
1239         \makeatletter
1240         \input { \l_CDR_pyg_tex_tl }\ignorespaces
1241         \makeatother
1242     }
1243     \group_end:
1244 }
1245 ] { CDR@Source } #1
1246 { {
1247     \exp_args:NV \fvset \l_CDR_kv_clist
1248     \DefineShortVerb { #1 }
1249     \SaveVerb [
1250         aftersave = {
1251             \UndefineShortVerb { #1 }
1252             \cs_set_eq:NN \CDR@FormattingPrep \FV@FormattingPrep
1253             \cs_set:Npn \FV@FormattingPrep {
1254                 \CDR@FormattingPrep
1255                 \CDR_tag_get:c { format }
1256             }
1257             \CDR@CodeEngineApply { \mbox {
1258                 \FV@UseKeyValues
1259                 \FV@FormattingPrep
1260                 \FV@SV@CDR@Code
1261             } }
1262             \group_end:
1263         }
1264     ] { CDR@Code } #1
1265 }
1266 }

1267 \NewDocumentCommand \CDRCode { 0{ } } {
1268     \group_begin:
1269     \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
1270         \prg_return_false:
1271     }
1272     \CDR_tag_keys_inherit:nn { __local } {
1273         __code, default.code, __pygments, default,
1274     }
1275     \CDR_tag_keys_set_known:nnN { __local } { #1 } \l_CDR_kv_clist
1276     \CDR_tag_provide_from_kv:V \l_CDR_kv_clist
1277     \CDR_tag_keys_set_known:nVN { __local } \l_CDR_kv_clist \l_CDR_kv_clist
1278     \exp_args:NNV
1279     \def \FV@KeyValues \l_CDR_kv_clist
1280     \CDR_tag_keys_inherit:nn { __local } {
1281         __fancyvrb,
1282     }
1283     \CDR_tag_keys_set:nV { __local } \l_CDR_kv_clist

```

```

1284 \CDR_tag_inherit:cf { __local } {
1285   \tl_if_empty:NF \l_CDR_tag_tl { \l_CDR_tag_tl, }
1286   __code, default.code, __pygments, default, __fancyvrb,
1287 }
1288 \CDRCode:n
1289 }
1290 \cs_set:Npn \CDRCode:n #1 {
1291   \CDR_tag_if_truthy:cTF {pygments} {
1292     \CDR_tag_keys_inherit:nn { __local } {
1293       __fancyvrb,
1294     }
1295     \CDR_tag_keys_set:nV { __local } \l_CDR_kv_clist
1296     \DefineShortVerb { #1 }
1297     \SaveVerb [
1298       aftersave = {
1299         \exp_args:Nx \UndefineShortVerb { #1 }
1300         \lua_now:n { CDR:highlight_code_setup() }
1301         \CDR_tag_get:cN {lang} \l_CDR_tl
1302         \lua_now:n { CDR:highlight_set_var('lang') }
1303         \CDR_tag_get:cN {cache} \l_CDR_tl
1304         \lua_now:n { CDR:highlight_set_var('cache') }
1305         \CDR_tag_get:cN {debug} \l_CDR_tl
1306         \lua_now:n { CDR:highlight_set_var('debug') }
1307         \CDR_tag_get:cN {style} \l_CDR_tl
1308         \lua_now:n { CDR:highlight_set_var('style') }
1309         \lua_now:n { CDR:highlight_set_var('source', 'FV@SV@CDR@Source') }
1310         \exp_args:NNV
1311         \def \FV@KeyValues \l_CDR_kv_clist
1312         \FV@UseKeyValues
1313         \frenchspacing
1314         % \FV@SetupFont Break
1315         \FV@DefineWhiteSpace
1316         \FancyVerbDefineActive
1317         \FancyVerbFormatCom
1318         \CDR@DefineSp
1319         \CDRCodeformat:
1320         \CDR_tag_get:c { format }
1321         \CDR@CodeEngineApply {
1322           \CDR@StyleIfExist { \CDR_tag_get:c {style} } {
1323             \CDR@StyleUseTag
1324             \lua_now:n { CDR:highlight_source(false, true) }
1325           } {
1326             \lua_now:n { CDR:highlight_source(true, true) }
1327             \input { \l_CDR_pyg_sty_tl }
1328             \CDR@StyleUseTag
1329           }
1330           \makeatletter
1331           \input { \l_CDR_pyg_tex_tl }
1332           \makeatother
1333         }
1334         \group_end:
1335       }
1336     ] { CDR@Source } #1
1337   } {

```

```

1338 \DefineShortVerb { #1 }
1339 \SaveVerb [
1340   aftersave = {
1341     \UndefinedShortVerb { #1 }
1342     \cs_set_eq:NN \CDR@FormattingPrep \FV@FormattingPrep
1343     \cs_set:Npn \FV@FormattingPrep {
1344       \CDR@FormattingPrep
1345       \CDR_tag_get:c { format }
1346     }
1347     \CDR@CodeEngineApply { \mbox {
1348       \exp_args:NNV
1349       \def \FV@KeyValues \l_CDR_kv_clist
1350       \FV@UseKeyValues
1351       \FV@FormattingPrep
1352       \@nameuse{FV@SV@CDR@Code}
1353     } }
1354     \group_end:
1355   }
1356 ] { CDR@Code } #1
1357 }
1358 }

1359 \RenewDocumentCommand \CDRCode { 0{} } {
1360   \group_begin:
1361   \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
1362     \prg_return_false:
1363   }
1364   \CDR_tag_keys_inherit:nn { __local } {
1365     __code, default.code, __pygments, default,
1366   }
1367   \CDR_tag_keys_set_known:nnN { __local } { #1 } \l_CDR_kv_clist
1368   \CDR_tag_provide_from_kv:V \l_CDR_kv_clist
1369   \CDR_tag_keys_set_known:nVN { __local } \l_CDR_kv_clist \l_CDR_kv_clist
1370   \CDR_tag_keys_inherit:nn { __local } {
1371     __fancyvrb,
1372   }
1373   \CDR_tag_keys_set:nV { __local } \l_CDR_kv_clist
1374   \CDR_tag_inherit:cf { __local } {
1375     \tl_if_empty:NF \l_CDR_tag_tl { \l_CDR_tag_tl, }
1376     __code, default.code, __pygments, default, __fancyvrb,
1377   }
1378   \fvset{showspaces}
1379   \CDRCode:n
1380 }

```

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.

```

1381 \CDR_tag_keys_define:nn { __block } {

```

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

```
1382 no~export .code:n = \CDR_tag_boolean_set:x { #1 },
1383 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.

```
1384 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.

```
1385 dry~numbers .code:n = \CDR_tag_set:,
1386 dry~numbers .default:n = true,
```

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

```
1387 test .code:n = \CDR_tag_boolean_set:x { #1 },
1388 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.

```
1389 engine~options .code:n = \CDR_tag_set:,
1390 engine~options .value_required:n = true,
```

🔴 **__initialize** initialize

```
1391 __initialize .meta:n = {
1392     no~export = false,
1393     no~export~format = ,
1394     dry~numbers = false,
1395     test = false,
1396     engine~options = ,
1397 },
1398 __initialize .value_forbidden:n = true,
1399 }
```

15.2 Implementation

15.2.1 Storage

__start For the line numbering, these are loop integer controls.
__step **__start** for the first index
__last

__step for the step, defaults to 1
__last for the last index, included

```
1400 \CDR_int_new:cn { __start } { 0 }
1401 \CDR_int_new:cn { __step } { 0 }
1402 \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.

```
1403 \clist_map_inline:nn { i, ii, iii, iv } {
1404   \cs_set_eq:cc { CDR@ListProcessLine@ #1 } { FV@ListProcessLine@ #1 }
1405 }
```

`\CDRBlock_preflight:n` `\CDRBlock_preflight:n {(CDR@Block kv list)}`

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

```
1406 \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 ??.)

```
1407 \seq_new:N \l_CDR_vrb_seq
```

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

```
1408 \cs_new:Npn \FVB@CDRBlock {
1409   \@bsphack
1410   \exp_args:NV \CDRBlock_preflight:n \FV@KeyValues
1411   \begingroup
1412   \lua_now:n {
1413     CDR.synctex_tag = tex.get_synctex_tag();
1414     CDR.synctex_line = tex.inputlineno;
1415     tex.set_synctex_mode(1)
1416   }
1417   \seq_clear:N \l_CDR_vrb_seq
1418   \cs_set_protected_nopar:Npn \FV@ProcessLine ##1 {
1419     \seq_put_right:Nn \l_CDR_vrb_seq { ##1 }
1420   }
1421   \FV@Scan
1422 }
```

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

```
1423 \cs_new:Npn \FVE@CDRBlock {%
1424   \CDRBlock_setup:
1425   \CDR_if_no_export:F {
1426     \seq_map_inline:Nn \l_CDR_vrb_seq {
1427       \tl_set:Nn \l_CDR_tl { ##1 }
1428       \lua_now:n { CDR:record_line('l_CDR_tl') }
1429     }
1430   }
```

```

1430 }
1431 \CDRBlock_engine_begin:
1432 \CDR_if_pygments:TF {
1433   \CDRBlock@Pyg
1434 } {
1435   \CDRBlock@FV
1436 }
1437 \lua_now:n {
1438   tex.set_synctex_mode(0);
1439   CDR.synctex_line = 0;
1440 }
1441 \CDRBlock_engine_end:
1442 \CDRBlock_teardown:
1443 \endgroup
1444 \@esphack
1445 }
1446 \DefineVerbatimEnvironment{CDRBlock}{CDRBlock}{}
1447 % \begin{MacroCode}
1448 \cs_new_protected_nopar:Npn \CDRBlock_setup: {
1449   \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
1450     \prg_return_true:
1451   }
1452   \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.

```

1453 \CDRBlock_setup_tags:
1454 \CDR_tag_keys_inherit:nn { __local } {
1455   __block, __pygments.block, default.block,
1456   __pygments, default
1457 }
1458 \CDR_tag_keys_set_known:nVN { __local } \FV@KeyValues \FV@KeyValues
1459 \CDR_tag_provide_from_kv:V \FV@KeyValues
1460 \CDR_tag_keys_set_known:nVN { __local } \FV@KeyValues \FV@KeyValues
1461 \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.

```

1462 \CDR_tag_keys_inherit:nn { __local } {
1463   \CDR_tag_if_eq:cnF { engine } { default } {
1464     __fancyvrb.frame,
1465   },
1466   __fancyvrb.number,
1467 }
1468 \CDR_tag_keys_set_known:nVN { __local } \FV@KeyValues \FV@KeyValues

```

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


```

1469 \CDR_tag_keys_inherit:nn { __local } {
1470   __fancyvrb.block,
1471   __fancyvrb,
1472 }
1473 \CDR_tag_keys_set_known:nVN { __local } \FV@KeyValues \l_CDR_kv_clist
1474 \lua_now:n {
1475   CDR:highlight_block_setup('g_CDR_tags_clist')
1476 }
1477 \CDR_set_conditional:Nn \CDR_if_pygments:
1478   { \CDR_tag_if_truthy_p:c { pygments } }
1479 \CDR_set_conditional:Nn \CDR_if_no_export:
1480   { \CDR_tag_if_truthy_p:c { no~export } }
1481 \CDR_set_conditional:Nn \CDR_if_dry_numbers:
1482   { \CDR_tag_if_truthy_p:c { dry~numbers } }
1483 \CDR_set_conditional:Nn \CDR_if_number_on:
1484   { ! \CDR_tag_if_eq_p:cn { numbers } { none } }
1485 \CDR_set_conditional:Nn \CDR_tags_if_already: {
1486   \CDR_tag_if_truthy_p:c { only~top } &&
1487   \CDR_clist_if_eq_p:NN \g_CDR_tags_clist \g_CDR_last_tags_clist
1488 }
1489 \CDR_if_number_on:T {
1490   \clist_map_inline:Nn \g_CDR_tags_clist {
1491     \CDR_int_if_exist:cF { ##1 } {
1492       \CDR_int_new:cn { ##1 } { 1 }
1493     }
1494   }
1495 }
1496 }

1497 \cs_new_protected_nopar:Npn \CDRBlock_tear_down: {
1498   \CDR_if_dry_numbers:F {
1499     \tl_set:Nx \l_CDR_tl { \seq_count:N \l_CDR_vrb_seq }
1500     \clist_map_inline:Nn \g_CDR_tags_clist {
1501       \CDR_int_gadd:cn { ##1 } { \l_CDR_tl }
1502     }
1503   }
1504   \lua_now:n {
1505     CDR:highlight_block_tear_down()
1506   }
1507 }

```

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.

```

1508 \cs_set_protected:Npn \CDRBlock@Pyg {
1509 \CDR@Debug { \string\CDRBlock@Pyg/\the\inputlineno }
1510 \CDR_tag_get:cN {lang} \l_CDR_tl
1511 \lua_now:n { CDR:highlight_set_var('lang') }
1512 \CDR_tag_get:cN {cache} \l_CDR_tl

```

```

1513 \lua_now:n { CDR:highlight_set_var('cache') }
1514 \CDR_tag_get:cN {debug} \l_CDR_tl
1515 \lua_now:n { CDR:highlight_set_var('debug') }
1516 \CDR_tag_get:cN {texcomments} \l_CDR_tl
1517 \lua_now:n { CDR:highlight_set_var('texcomments') }
1518 \CDR_tag_get:cN {escapeinside} \l_CDR_tl
1519 \lua_now:n { CDR:highlight_set_var('escapeinside') }
1520 \CDR_tag_get:cN {mathescape} \l_CDR_tl
1521 \lua_now:n { CDR:highlight_set_var('mathescape') }
1522 \CDR_tag_get:cN {style} \l_CDR_tl
1523 \lua_now:n { CDR:highlight_set_var('style') }
1524 \CDR@StyleIfExist { \l_CDR_tl } { } {
1525   \lua_now:n { CDR:highlight_source(true, false) }
1526   \input { \l_CDR_pyg_sty_tl }
1527 }
1528 \CDR@StyleUseTag
1529 % \CDR_tag_get:c { format }
1530 % \cs_set:Npn \CDR:nn ##1 ##2 {
1531 %\CDR@Debug{Debug.CDRBlock.FV.KEYS:\tl_to_str:n{##1}->\tl_to_str:n{##2}}
1532 %   \fvset{ ##1 = ##2, }
1533 % }
1534 % \clist_map_inline:Nn \c_CDRBlock@Pyg_clist {
1535 %   \exp_args:Nnx
1536 %   \CDR:nn { ##1 } { \CDR_tag_get:c { ##1 } }
1537 % }
1538 \CDR@DefineSp
1539 \lua_now:n { CDR:highlight_source(false, true) }
1540 \fvset{ commandchars=\\{\} }
1541 \FV@UseVerbatim {
1542   \CDR_tag_get:c { format }
1543   \CDR_if_no_export:T {
1544     \CDR_tag_get:c { no-export-format }
1545   }
1546   \makeatletter
1547   \input{ \l_CDR_pyg_tex_tl }
1548   \makeatother
1549   \def \FV@ProcessLine {}
1550 }
1551 \CDR_if_number_on:T {
1552   \CDR_int_add:cn { __last } { 1 }
1553   \clist_map_inline:Nn \g_CDR_tags_clist {
1554     \CDR_int_gset:cc { ##1 } { __last }
1555 \CDR@Debug {DEBUG.CDRBlock.LAST: ##1 -> \CDR_int_use:c { ##1 } }
1556   }
1557 }
1558 }

```

\c_CDRBlock@Pyg_clist

(End definition for \c_CDRBlock@Pyg_clist. This variable is documented on page ??.)

```

1559 \clist_const:Nn \c_CDRBlock@Pyg_clist {
1560 % __fancyvrb:
1561   formatcom,% = ,
1562   fontfamily,% = tt,

```

```

1563 fontsize,% = auto,
1564 fontseries,% = auto,
1565 fontshape,% = auto,
1566 showspaces,% = false,
1567 showtabs,% = false,
1568 obeytabs,% = false,
1569 tabsize,% = 2,
1570 defineactive,% = ,
1571 reflabel,% = ,
1572 % __fancyvrb.frame:
1573 frame,% = none,
1574 framerule,% = 0.4pt,
1575 framesep,% = \fboxsep,
1576 rulecolor,% = black,
1577 fillcolor,% = ,
1578 label,% = ,
1579 labelposition,% = none,% auto?
1580 % __fancyvrb.block:
1581 % commentchar,% = ,
1582 % gobble,% = 0,
1583 baselinestretch,% = auto,
1584 resetmargins,% = true,
1585 xleftmargin,% = 0pt,
1586 xrightmargin,% = 0pt,
1587 hfuzz,% = 2pt,
1588 samepage,% = false,
1589 % __fancyvrb.number
1590 % numbers,% = left,
1591 % numbersep,% = 1ex,
1592 % firstnumber,% = auto,
1593 % stepnumber,% = 1,
1594 % numberblanklines,% = true,
1595 % firstline,% = ,
1596 % lastline,% = ,
1597 }

```

Info

```

1598 \cs_new:Npn \CDR@NumberFormat {
1599   \CDR_tag_get:c { numbers~format }
1600 }
1601 \cs_new:Npn \CDR@NumberSep {
1602   \hspace{ \CDR_tag_get:c { numbersep } }
1603 }
1604 \cs_new:Npn \CDR@TagsFormat {
1605   \CDR_tag_get:c { tags~format }
1606 }

```

<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>	

```

1607 \cs_new:Npn \CDR_info_N_L:n #1 {
1608   \hbox_overlap_left:n {
1609     \cs_set:Npn \baselinestretch { 1 }
1610     { \CDR@NumberFormat
1611       #1
1612     }
1613     \CDR@NumberSep
1614   }
1615 }
1616 \cs_new:Npn \CDR_info_T_L:n #1 {
1617   \hbox_overlap_left:n {
1618     \cs_set:Npn \baselinestretch { 1 }
1619     \CDR@NumberFormat
1620     \smash{
1621       \parbox[b]{\marginparwidth}{
1622         \raggedleft
1623         { \CDR@TagsFormat \g_CDR_tags_clist :}
1624       }
1625       #1
1626     }
1627     \CDR@NumberSep
1628   }
1629 }
1630 \cs_new:Npn \CDR_info_N_R:n #1 {
1631   \hbox_overlap_right:n {
1632     \CDR@NumberSep
1633     \cs_set:Npn \baselinestretch { 1 }
1634     \CDR@NumberFormat
1635     #1
1636   }
1637 }
1638 \cs_new:Npn \CDR_info_T_R:n #1 {
1639   \hbox_overlap_right:n {
1640     \cs_set:Npn \baselinestretch { 1 }
1641     \CDR@NumberSep
1642     \CDR@NumberFormat
1643     \smash {
1644       \parbox[b]{\marginparwidth}{
1645         \raggedright
1646         #1:
1647         { \CDR@TagsFormat \space \g_CDR_tags_clist}
1648       }
1649     }
1650   }
1651 }

```

\CDR_number_alt:n First line.

```

1652 \cs_set:Npn \CDR_number_alt:n #1 {
1653   \use:c { CDRNumber
1654     \CDR_if_number_visible:nTF { #1 } { Main } { Other }
1655   } { #1 }

```

```

1656 }
1657 \cs_set:Npn \CDR_number_alt: {
1658 \CDR@Debug{ALT: \CDR_int_use:c { __ } }
1659 \CDR_number_alt:n { \CDR_int_use:c { __ } }
1660 }

```

<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.

```

1661 \cs_new:Npn \CDRNumberMain {
1662 }
1663 \cs_new:Npn \CDRNumberOther {
1664     \use_none:n
1665 }

```

<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 { __ }`

```

1666 \cs_new:Npn \CDR@NumberMain {
1667     \CDRNumberMain { \CDR_int_use:c { __ } }
1668 }
1669 \cs_new:Npn \CDR@NumberOther {
1670     \CDRNumberOther { \CDR_int_use:c { __ } }
1671 }

```

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.

```

1672 \cs_new:Npn \CDR_line_N_N:n {
1673 \CDR@Debug {Debug.CDR_line_N_N:n}
1674 \CDR_line_box_N:n
1675 }
1676
1677 \cs_new:Npn \CDR_line_L_N:n #1 {
1678 \CDR@Debug {Debug.CDR_line_L_N:n}
1679 \CDR_line_box:nnn { \CDR_info_T_L:n { } } { #1 } { }
1680 }
1681
1682 \cs_new:Npn \CDR_line_R_N:n #1 {
1683 \CDR@Debug {Debug.CDR_line_R_N:n}
1684 \CDR_line_box:nnn { } { #1 } { \CDR_info_T_R:n { } }

```

```

1685 }
1686
1687 \cs_new:Npn \CDR_line_S_N:n {
1688 \CDR@Debug {Debug.CDR_line_S_N:n}
1689 \CDR_line_box_N:n
1690 }
1691
1692 \cs_new:Npn \CDR_line_O_N:n {
1693 \CDR@Debug {STEP:CDR_line_O_N:n}
1694 \CDR_line_box_N:n
1695 }
1696
1697 \cs_new:Npn \CDR_line_N_L:n #1 {
1698 \CDR@Debug {STEP:CDR_line_N_L:n}
1699 \CDR_if_no_number:TF {
1700 \CDR_line_box:nnn {
1701 \CDR_info_N_L:n { \CDR@NumberMain }
1702 } { #1 } {}
1703 } {
1704 \CDR_line_box_L:n { #1 }
1705 }
1706 }
1707
1708 \cs_new:Npn \CDR_line_L_L:n #1 {
1709 \CDR@Debug {STEP:CDR_line_L_L:n}
1710 \CDR_if_number_single:TF {
1711 \CDR_line_box:nnn {
1712 \CDR_info_T_L:n { \space \CDR@NumberMain }
1713 } { #1 } {}
1714 } {
1715 \CDR_if_no_number:TF {
1716 \cs_set:Npn \CDR@@Line {
1717 \cs_set:Npn \CDR@@Line {
1718 \CDR_line_box_L:nn { \CDR_info_N_L:n { \CDR@NumberOther } }
1719 }
1720 \CDR_line_box_L:nn { \CDR_info_N_L:n { \CDR@NumberMain } }
1721 }
1722 } {
1723 \cs_set:Npn \CDR@@Line {
1724 \CDR_line_box_L:nn { \CDR_info_N_L:n { \CDR_number_alt: } }
1725 }
1726 }
1727 \CDR_line_box:nnn { \CDR_info_T_L:n { } } { #1 } { }
1728 }
1729 }
1730
1731 \cs_new:Npn \CDR_line_R_R:n #1 {
1732 \CDR@Debug {STEP:CDR_line_R_R:n}
1733 \CDR_if_number_single:TF {
1734 \CDR_line_box:nnn { } { #1 } {
1735 \CDR_info_T_R:n { \CDR@NumberMain }
1736 }
1737 } {
1738 \CDR_if_no_number:TF {

```

```

1739     \cs_set:Npn \CDR@@Line {
1740         \cs_set:Npn \CDR@@Line {
1741             \CDR_line_box_R:nn { \CDR_info_N_R:n { \CDR@NumberOther } }
1742         }
1743         \CDR_line_box_R:nn { \CDR_info_N_R:n { \CDR@NumberMain } }
1744     }
1745     } {
1746         \cs_set:Npn \CDR@@Line {
1747             \CDR_line_box_R:nn { \CDR_info_N_R:n { \CDR_number_alt: } }
1748         }
1749     }
1750     \CDR_line_box:nnn { } { #1 } { \CDR_info_T_R:n { } }
1751 }
1752 }
1753
1754 \cs_new:Npn \CDR_line_R_L:n #1 {
1755 \CDR@Debug {STEP:CDR_line_R_L:n}
1756 \CDR_line_box:nnn {
1757     \CDR_if_no_number:TF {
1758         \CDR_info_N_L:n { \CDR@NumberMain }
1759     } {
1760         \CDR_info_N_L:n { \CDR_number_alt: }
1761     }
1762 } { #1 } {
1763     \CDR_info_T_R:n { }
1764 }
1765 }
1766
1767 \cs_set_eq:NN \CDR_line_S_L:n \CDR_line_L_L:n
1768 \cs_set_eq:NN \CDR_line_O_L:n \CDR_line_R_L:n
1769
1770 \cs_new:Npn \CDR_line_N_R:n {
1771 \typeout {STEP:CDR_line_N_R:n}
1772 \CDR_line_box_R:n
1773 }
1774
1775 \cs_new:Npn \CDR_line_L_R:n #1 {
1776 \CDR@Debug {STEP:CDR_line_L_R:n}
1777 \CDR_line_box:nnn {
1778     \CDR_info_T_L:n { }
1779 } { #1 } {
1780     \CDR_if_no_number:TF {
1781         \CDR_info_N_R:n { \CDR@NumberMain }
1782     } {
1783         \CDR_info_N_R:n { \CDR_number_alt: }
1784     }
1785 }
1786 }
1787
1788 \cs_set_eq:NN \CDR_line_S_R:n \CDR_line_R_R:n
1789 \cs_set_eq:NN \CDR_line_O_R:n \CDR_line_L_R:n
1790
1791
1792 \cs_new:Npn \CDR_line_box_N:n #1 {

```

```

1793 \CDR@Debug {STEP:CDR_line_box_N:n}
1794 \CDR_line_box:nnn { } { #1 } {}
1795 }
1796
1797 \cs_new:Npn \CDR_line_box_L:n #1 {
1798 \CDR@Debug {STEP:CDR_line_box_L:n}
1799 \CDR_line_box:nnn {
1800 \CDR_info_N_L:n { \CDR_number_alt: }
1801 } { #1 } {}
1802 }
1803
1804 \cs_new:Npn \CDR_line_box_R:n #1 {
1805 \CDR@Debug {STEP:CDR_line_box_R:n}
1806 \CDR_line_box:nnn { } { #1 } {
1807 \CDR_info_N_R:n { \CDR_number_alt: }
1808 }
1809 }

```

\CDR_line_box:nnn	\CDR_line_box:nnn {<left info>} {<line content>} {<right info>}
\CDR_line_box_L:nn	\CDR_line_box_L:nn {<left info>} {<line content>}
\CDR_line_box_R:nn	\CDR_line_box_R:nn {<right info>} {<line content>}
\CDR_line_box:nn	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 signarture).

```

1810 \cs_new:Npn \CDR_line_box:nnn #1 #2 #3 {
1811 \CDR@Debug {\string\CDR_line_box:nnn/\tl_to_str:n{#1}/.../\tl_to_str:n{#3}/}
1812 \directlua {
1813 tex.set_synctex_tag( CDR.synctex_tag )
1814 }
1815 \lua_now:e {
1816 tex.set_synctex_line(CDR.synctex_line+( \CDR_int_use:c { __ } ) )
1817 }
1818 \hbox to \hsize {
1819 \kern \leftmargin
1820 #1
1821 \hbox to \linewidth {
1822 \FV@LeftListFrame
1823 #2
1824 \hss
1825 \FV@RightListFrame
1826 }
1827 #3
1828 }
1829 }
1830 \cs_new:Npn \CDR_line_box_L:nn #1 #2 {
1831 \CDR_line_box:nnn { #1 } { #2 } {}
1832 }
1833 \cs_new:Npn \CDR_line_box_R:nn #1 #2 {
1834 \CDR@Debug {STEP:CDR_line_box_R:nn}
1835 \CDR_line_box:nnn { } { #2 } { #1 }
1836 }
1837 \cs_new:Npn \CDR_line_box_N:nn #1 #2 {

```



```

1838 \CDR@Debug {STEP:CDR_line_box_N:nn}
1839 \CDR_line_box:nnn { } { #2 } {}
1840 }

```

Lines

```

1841 \cs_new:Npn \CDR@Line {
1842 \CDR@Debug {\string\CDR@Line}
1843 \peek_meaning_ignore_spaces:NTF [%]
1844 { \CDR_line:nnn } {
1845 \PackageError { code } { Missing~['%]
1846 ~at~first~\string\CDR@Line~call }
1847 }
1848 }

```

\CDR_line:nnn \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.

```

1849 \keys_define:nn { CDR@Line } {
1850 last .code:n = \CDR_int_set:cn { __last } { #1 },
1851 }
1852 \cs_new:Npn \CDR_line:nnn [ #1 ] #2 {
1853 \CDR@Debug {\string\CDR_line:nnn}
1854 \keys_set:nn { CDR@Line } { #1 }
1855 \CDR_int_set:cn { __ } { 0 }
1856 \CDR_if_number_on:TF {
1857 \CDR_tag_if_eq:cnTF { firstnumber } { last } {
1858 \clist_map_inline:Nn \g_CDR_tags_clist {
1859 \clist_map_break:n {
1860 \CDR_int_set:cc { __start } { ##1 }
1861 \CDR@Debug {START: ##1=\CDR_int_use:c { ##1 } }
1862 }
1863 }
1864 } {
1865 \CDR_tag_if_eq:cnTF { firstnumber } { auto } {
1866 \CDR_int_set:cn { __start } { 1 }
1867 } {
1868 \CDR_int_set:cn { __start } { \CDR_tag_get:c { firstnumber } }
1869 }
1870 }

```

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

```

1871 \CDR_set_conditional:Nn \CDR_if_number_single: {
1872 \CDR_int_compare_p:cNn { __last } = 1
1873 }
1874 \CDR@Debug{***** TEST: \CDR_if_number_single:TF { SINGLE } { MULTI } }
1875 \CDR_int_add:cn { __last } { \CDR_int:c { __start } - 1 }
1876 \CDR_int_set:cn { __step } { \CDR_tag_get:c { stepnumber } }
1877 \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.

```

1878 \CDR_int_compare:cNnTF { __step } < 2 {
1879 \CDR_int_set:cn { __step } { 1 }
1880 \CDR_set_conditional_alt:Nn \CDR_if_visible_at_index:n {
1881 ! \CDR_int_compare_p:cNn { __last } < { ##1 + \CDR_int:c { __start } - 1 }
1882 }
1883 \CDR_set_conditional_alt:Nn \CDR_if_number_visible:n {
1884 ! \CDR_int_compare_p:cNn { __last } < { ##1 }
1885 }
1886 } {
1887 \CDR_set_conditional_alt:Nn \CDR_if_visible_at_index:n {
1888 ! \CDR_int_compare_p:cNn { __last } < { ##1 + \CDR_int:c { __start } - 1 }
1889 }
1890 \CDR_set_conditional_alt:Nn \CDR_if_number_visible:n {
1891 \int_compare_p:nNn {
1892 ( ##1 + \CDR_int:c { __start } - 1 )
1893 / \CDR_int:c { __step } * \CDR_int:c { __step }
1894 - \CDR_int:c { __start } + 1
1895 } = { ##1 }
1896 && \CDR_if_visible_at_index_p:n { ##1 }
1897 }
1898 }
1899 \CDR@Debug {CDR_line:nnn:1}

1900 \CDR_set_conditional:Nn \CDR_if_no_number: {
1901 \CDR_int_compare_p:cNn { __start } > {
1902 \CDR_int:c { __last } / \CDR_int:c { __step } * \CDR_int:c { __step }
1903 }
1904 }
1905 \cs_set:Npn \CDR@Line ##1 {
1906 \CDR@Debug {\string\CDR@Line(A), \the\inputlineno}
1907 \CDR_int_set:cn { __ } { ##1 + \CDR_int:c { __start } - #2 }
1908 \CDR@@Line
1909 }
1910 \CDR_int_set:cn { __ } { \CDR_int:c { __start } + 1 - #2 }
1911 } {
1912 \CDR@Debug {NUMBER-OFF}
1913 \cs_set:Npn \CDR@Line ##1 {
1914 \CDR@Debug {\string\CDR@Line(B), \the\inputlineno}
1915 \CDR@@Line
1916 }
1917 }
1918 \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 always display the first one but it is not satisfying when lines are numbered stepwise, moreover when the tags should be displayed.

```

1919 \tl_clear:N \l_CDR_tl
1920 \CDR_tags_if_already:TF {
1921   \tl_put_right:Nn \l_CDR_tl { _N }
1922 } {
1923   \exp_args:Nx
1924   \str_case:nnF { \CDR_tag_get:c { show-tags } } {
1925     { left } { \tl_put_right:Nn \l_CDR_tl { _L } }
1926     { right } { \tl_put_right:Nn \l_CDR_tl { _R } }
1927     { none } { \tl_put_right:Nn \l_CDR_tl { _N } }
1928     { numbers } { \tl_put_right:Nn \l_CDR_tl { _S } }
1929     { mirror } { \tl_put_right:Nn \l_CDR_tl { _O } }
1930   } { \PackageError
1931     { coder }
1932     { Unknown-show-tags-options~:~ \CDR_tag_get:c { show-tags } }
1933   }
1934 }

```

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

```

1935 \exp_args:Nx
1936 \str_case:nnF { \CDR_tag_get:c { numbers } } {
1937   { left } {
1938     \tl_put_right:Nn \l_CDR_tl { _L }
1939     \cs_set:Npn \CDR@@Line { \CDR_line_box_L:n }
1940   }
1941   { right } {
1942     \tl_put_right:Nn \l_CDR_tl { _R }
1943     \cs_set:Npn \CDR@@Line { \CDR_line_box_R:n }
1944   }
1945   { none } {
1946     \tl_put_right:Nn \l_CDR_tl { _N }
1947     \cs_set:Npn \CDR@@Line { \CDR_line_box_N:n }
1948   }
1949 } { \PackageError
1950   { coder }
1951   { Unknown-numbers-options~:~ \CDR_tag_get:c { numbers } }
1952 }
1953 \CDR@Debug {BRANCH:CDR_line \l_CDR_tl :n}
1954 \use:c { CDR_line \l_CDR_tl :n }
1955 }

```

15.2.5 fancyvrb only

pygments is not used, fall back to fancyvrb features.

```

CDRBlock@FV \CDRBlock@Fv

```

```

1956 \cs_new_protected:Npn \CDRBlock@FV {
1957 \CDR@Debug {DEBUG.Block.FV}
1958 % \tl_clear:N \FV@KeyValues

```

```

1959 % \cs_set:Npn \CDR:nn ##1 ##2 {
1960 %\CDR@Debug{Debug.CDRBlock.FV.KEYS:\tl_to_str:n{##1}->\tl_to_str:n{##2}}
1961 % \fvset{ ##1 = { ##2 }, }
1962 % }
1963 % \clist_map_inline:Nn \c_CDRBlock@FV_clist {
1964 % \exp_args:Nnf
1965 % \CDR:nn { ##1 } { \CDR_tag_get:c { ##1 } }
1966 % }
1967 \FV@UseKeyValues
1968 \FV@UseVerbatim {
1969 \CDR_tag_get:c { format }
1970 \CDR_if_no_export:T {
1971 \CDR_tag_get:c { no-export-format }
1972 }
1973 \tl_set:Nx \l_CDR_tl { [ last=%]
1974 \seq_count:N \l_CDR_vrb_seq %[
1975 ] }
1976 \seq_map_indexed_inline:Nn \l_CDR_vrb_seq {
1977 \exp_last_unbraced:NV \CDR@Line \l_CDR_tl { ##1 } { ##2 }
1978 \tl_clear:N \l_CDR_tl
1979 }
1980 \tl_clear:N \FV@ProcessLine
1981 }
1982 \CDR_if_number_on:T {
1983 \CDR_int_compare:cNnTF { __ } > 0 {
1984 \CDR_int_set:cn { __ } {
1985 \value{FancyVerbLine} - \CDR_int_use:c { __ } + 1
1986 }
1987 \clist_map_inline:Nn \g_CDR_tags_clist {
1988 \CDR_int_gadd:cc { ##1 } { __ }
1989 }
1990 } {
1991 \CDR_int_set:cn { __ } { \value{FancyVerbLine} + 1 }
1992 \clist_map_inline:Nn \g_CDR_tags_clist {
1993 \CDR_int_gset:cc { ##1 } { __ }
1994 \CDR@Debug { DEBUG.CDRBlock.FV.Last: ##1/\CDR_int_use:c { ##1 } }
1995 }
1996 }
1997 }
1998 }

```

\c_CDRBlock@FV_clist

(End definition for \c_CDRBlock@FV_clist. This variable is documented on page ??.)

```

1999 \clist_const:Nn \c_CDRBlock@FV_clist {
2000 % __fancyvrb:
2001 formatcom,% = ,
2002 fontfamily,% = tt,
2003 fontsize,% = auto,
2004 fontseries,% = auto,
2005 fontshape,% = auto,
2006 showspaces,% = false,
2007 showtabs,% = false,
2008 obeytabs,% = false,

```

```

2009 tabsize,% = 2,
2010 defineactive,% = ,
2011 rellabel,% = ,
2012 % __fancyvrb.frame:
2013 frame,% = none,
2014 framerule,% = 0.4pt,
2015 framesep,% = \fboxsep,
2016 rulecolor,% = black,
2017 fillcolor,% = ,
2018 label,% = ,
2019 labelposition,% = none,% auto?
2020 % __fancyvrb.block:
2021 % commentchar,% = ,
2022 gobble,% = 0,
2023 baselinestretch,% = auto,
2024 resetmargins,% = true,
2025 xleftmargin,% = 0pt,
2026 xrightmargin,% = 0pt,
2027 hfuzz,% = 2pt,
2028 samepage,% = false,
2029 % __fancyvrb.number
2030 numbers,% = none,
2031 numbersep,% = 1ex,
2032 firstnumber,% = auto,
2033 stepnumber,% = 1,
2034 numberblanklines,% = true,
2035 firstline,% = ,
2036 lastline,% = ,
2037 }

```

15.2.6 Utilities

This is put aside for better clarity.

```

\CDR_set_conditional:Nn \CDR_set_conditional:Nn <core name> {<condition>}

```

Wrapper over \prg_set_conditional:Nnn.

```

2038 \cs_new:Npn \CDR_set_conditional:Nn #1 #2 {
2039   \bool_if:nTF { #2 } {
2040     \prg_set_conditional:Nnn #1 { p, T, F, TF } { \prg_return_true: }
2041   } {
2042     \prg_set_conditional:Nnn #1 { p, T, F, TF } { \prg_return_false: }
2043   }
2044 }

```

```

\CDR_set_conditional_alt:Nn \CDR_set_conditional_alt:Nnn <core name> {<condition>}

```

Wrapper over \prg_set_conditional:Nnn.

```

2045 \cs_new:Npn \CDR_set_conditional_alt:Nn #1 #2 {
2046   \prg_set_conditional:Nnn #1 { p, T, F, TF } {
2047     \bool_if:nTF { #2 } { \prg_return_true: } { \prg_return_false: }
2048   }
2049 }

```

<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.

```

2050 \prg_set_conditional:Nnn \CDR_if_middle_column: { p, T, F, TF } { \prg_return_false: }
2051 \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.

<code>\CDR_tags_if_visible_p:n *</code>	<code>\CDR_tags_if_visible:nTF {<left right>} {<true code>} {<false code>}</code>
<code>\CDR_tags_if_visible:nTF *</code>	

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

```

2052 \prg_set_conditional:Nnn \CDR_tags_if_visible:n { p, T, F, TF } {
2053   \bool_if:nTF {
2054     ( \CDR_tag_if_eq_p:cn { show-tags } { ##1 } ||
2055       \CDR_tag_if_eq_p:cn { show-tags } { numbers } &&
2056       \CDR_tag_if_eq_p:cn { numbers } { ##1 }
2057     ) && ! \CDR_tags_if_already_p:
2058   } {
2059     \prg_return_true:
2060   } {
2061     \prg_return_false:
2062   }
2063 }

```

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

```

2064 \cs_new_protected_nopar:Npn \CDRBlock_setup_tags: {
2065   \CDR_tag_keys_inherit:nn { __local } { __tags }
2066   \CDR_tag_keys_set_known:nVN { __local } \FV@KeyValues \FV@KeyValues
2067   \CDR_tag_if_exist_here:ccT { __local } { tags } {
2068     \CDR_tag_get:cN { tags } \l_CDR_clist
2069     \clist_if_empty:NF \l_CDR_clist {
2070       \clist_gset_eq:NN \g_CDR_tags_clist \l_CDR_clist
2071     }
2072   }
2073   \clist_if_empty:NT \g_CDR_tags_clist {
2074     \PackageWarning
2075       { coder }
2076       { No~(default)~tags~provided. }
2077   }
2078   \CDR@Debug {CDRBlock_setup_tags:\space\g_CDR_tags_clist}

```

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

```

2079 \CDR_tag_inherit:cf { __local } {
2080   \g_CDR_tags_clist,
2081   __block, __tags, default.block, __pygments.block,
2082   __fancyvrb.block __fancyvrb.frame, __fancyvrb.number,
2083   __pygments, default, __fancyvrb,
2084 }

```

For each $\langle tag\ name \rangle$, create an l3int variable and initialize it to 1.

```

2085 \clist_map_inline:Nn \g_CDR_tags_clist {
2086   \CDR_int_if_exist:cF { ##1 } {
2087     \CDR_int_new:cn { ##1 } { 1 }
2088   }
2089 }
2090 }

```

16 Management

$\backslash g_CDR_in_impl_bool$ Whether we are currently in the implementation section.

```

2091 \bool_new:N \g_CDR_in_impl_bool

```

(End definition for $\backslash g_CDR_in_impl_bool$. 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 \langle true code \rangle when code should be printed, \langle false code \rangle otherwise.

```

```

2092 \prg_new_conditional:Nnn \CDR_if_show_code: { p, T, F, TF } {
2093   \bool_if:nTF {
2094     \g_CDR_in_impl_bool && !\g_CDR_with_impl_bool
2095   } {
2096     \prg_return_false:
2097   } {
2098     \prg_return_true:
2099   }
2100 }

```

$\backslash g_CDR_with_impl_bool$

```

2101 \bool_new:N \g_CDR_with_impl_bool

```

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

```

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

```

Store the content of $\langle file\ name \rangle$ into the variable $\langle variable \rangle$. This is currently unstable.

```

2102 \DeclareDocumentCommand \CDRPreamble { m m } {
2103   \msg_info:nnn
2104     { coder }
2105     { :n }
2106     { Reading-preamble-from-file-"#2". }
2107   \tl_set:Nn \l_CDR_tl { #2 }
2108   \exp_args:NNx
2109   \tl_set:Nx #1 { \lua_now:n {CDR.print_file_content('l_CDR_tl')} }
2110 }

```

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

```

2111 \newcounter{CDR@impl@page}
2112 \DeclareDocumentCommand \CDRImplementation {} {
2113   \bool_if:NF \g_CDR_with_impl_bool {
2114     \clearpage
2115     \bool_gset_true:N \g_CDR_in_impl_bool
2116     \let\CDR@old@part\part
2117     \DeclareDocumentCommand\part{som}{}
2118     \let\CDR@old@section\section
2119     \DeclareDocumentCommand\section{som}{}
2120     \let\CDR@old@subsection\subsection
2121     \DeclareDocumentCommand\subsection{som}{}
2122     \let\CDR@old@subsubsection\subsubsection
2123     \DeclareDocumentCommand\subsubsection{som}{}
2124     \let\CDR@old@paragraph\paragraph
2125     \DeclareDocumentCommand\paragraph{som}{}
2126     \let\CDR@old@subparagraph\subparagraph
2127     \DeclareDocumentCommand\subparagraph{som}{}
2128     \cs_if_exist:NT \refsection{ \refsection }
2129     \setcounter{ CDR@impl@page }{ \value{page} }
2130   }
2131 }
2132 \DeclareDocumentCommand\CDRFinale {} {
2133   \bool_if:NF \g_CDR_with_impl_bool {
2134     \clearpage
2135     \bool_gset_false:N \g_CDR_in_impl_bool
2136     \let\part\CDR@old@part
2137     \let\section\CDR@old@section
2138     \let\subsection\CDR@old@subsection
2139     \let\subsubsection\CDR@old@subsubsection
2140     \let\paragraph\CDR@old@paragraph
2141     \let\subparagraph\CDR@old@subparagraph
2142     \setcounter { page } { \value{ CDR@impl@page } }
2143   }
2144 }
2145 %\cs_set_eq:NN \CDR_line_number: \prg_do_nothing:

```

19 Finale

```

2146 %\AddToHook { cmd/FancyVerbFormatLine/before } {
2147 %   \CDR_line_number:
2148 %}

```


2149
2150 \ExplSyntaxOff
2151
2152 %</sty>