

# `coder` — code inlined in a $\text{\LaTeX}$ document<sup>\*</sup>

Jérôme LAURENS<sup>†</sup>

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

---

<sup>\*</sup>This file describes version 2022/02/07, last revised 2022/02/07.

<sup>†</sup>E-mail: [jerome.laurens@u-bourgogne.fr](mailto:jerome.laurens@u-bourgogne.fr)

<sup>1</sup>The `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  $\text{\LaTeX}$  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`<sup>2</sup>.

### 4.1 Code flow

The normal code flow is

1. from `coder.sty`,  $\text{\LaTeX}$  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  $\text{\LaTeX}$  coloring commands. These are stored in a `*.pyg.tex` file named after the md5 digest of the original code chunk, a `*.pyg.sty`  $\text{\LaTeX}$  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  $\text{\LaTeX}$  logic as possible. It receives instructions from `coder.sty` as command line arguments,  $\text{\LaTeX}$  options, `pygments` options and `fancyvrb` options.

### 4.2 File exportation

1. The `\CDRExport` command declares a file path, a list of tags and other usefull informations like a coding language. These data are saved as export records by `coder-util.lua`.
2. When some `tags={...}` have been given to the `CDRBlock` environment, the `coder-util.lua` records the corresponding code chunk and its associate tags for later save.
3. Once the typesetting process is complete, `coder-util.lua`'s `CDR_export_...` methods are called to save all the files externally. For each export record, `coder-util.lua` collects all the chunks with the same tag and save them at the proper location.

---

<sup>2</sup>Work 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<sup>A</sup>T<sub>E</sub>X 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<sup>A</sup>T<sub>E</sub>X 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  $\text{\LaTeX}$  **\fboxsep** macro is added between the left vertical line and the text. Initially **none**: no frame.
- **label**={ [**top string**]  $\langle$ *string* $\rangle$  } label(s) to print on top, bottom or both, frame lines. If the label(s) contains special characters, comma or equal sign, it must be placed inside a group. If an optional  $\langle$ *top string* $\rangle$  is given between square brackets, it will be used for the top line and  $\langle$ *string* $\rangle$  for the bottom line. Otherwise,  $\langle$ *string* $\rangle$  is used for both the top or bottom lines. Label(s) are printed only if the **frame** parameter is one of **topline**, **bottomline**, **lines** or **single**. Initially empty: no label.
- **labelposition**=**none**|**topline**|**bottomline**|**all** position where to print the label(s) when defined. When options happen to be contradictory, like **frame**=**topline** and **labelposition**=**bottomline**, nothing is displayed. Initially **none** when no labels are defined, **topline** for one label and **all** otherwise.
- **numbers**=**none**|**left**|**right** numbering of the verbatim lines. If requested, this numbering is done outside the verbatim environment. Initially **none**: no numbering.
- **numbersep**= $\langle$ *dimension* $\rangle$  gap between numbers and verbatim lines. Initially 12pt.

- **firstnumber=auto|last| $\langle integer \rangle$**  number of the first line. **last** means that the numbering is continued from the previous verbatim environment. If an integer is given, its value will be used to start the numbering. Initially **auto**: numbering starts from 1.
- **stepnumber= $\langle integer \rangle$**  interval at which line numbers are printed. Initially 1: all lines are numbered.
- **numberblanklines[=true|false]** to number or not the white lines (really empty or containing blank characters only). Initially **true**: all lines are numbered.
- **firstline= $\langle integer \rangle$**  first line to print. Initially empty: all lines from the first are printed.
- **lastline= $\langle integer \rangle$**  last line to print. Initially empty: all lines until the last one are printed.
- **baselinestretch=auto| $\langle dimension \rangle$**  value to give to the usual `\baselinestretch` L<sup>A</sup>T<sub>E</sub>X 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<sub>E</sub>X `\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<sup>A</sup>T<sub>E</sub>X 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 T<sub>E</sub>X.

**!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 T<sub>E</sub>X 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 `<code var name>`. Build a configuration table with all data necessary for the processing, save it as a JSON file and launch `coder-tool.py` with the proper arguments.

```
237 local function highlight_code_setup(self)
238   self['.arguments'] = {
239     __cls__ = 'Arguments',
240     source = '',
241     cache = true,
242     debug = false,
243     pygopts = {
244       __cls__ = 'PygOpts',
245       lang = 'tex',
246       style = 'default',
247     },
248     texopts = {
249       __cls__ = 'TeXOpts',
250       tags = '',
251       is_inline = true,
252       pyg_sty_p = '',
253     },
254     fv_opts = {
255       __cls__ = 'FV0pts',
256     }
257   }
258   self.highlight_json_written = false
259 end
260
```

## 5.4 Block

---

`highlight_block_setup` CDR:highlight\_block\_setup(`<tags clist var>`)

---

Records the contents of the `<tags clist var>` L<sup>A</sup>T<sub>E</sub>X 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<sup>A</sup>T<sub>E</sub>X 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.

---

**export\_file**     CDR:export\_file(*<file name var>*)

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

```
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
```

---

**export\_file\_info**     CDR:export\_file\_info(*<key>*, *<value name var>*)

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

```
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
```

---

**export\_complete**     CDR:export\_complete()

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<sup>A</sup>T<sub>E</sub>X run and possibly between different L<sup>A</sup>T<sub>E</sub>X 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>&lt;style name.pyg.sty&gt;</i>, <i>&lt;digest.pyg.tex&gt;</i>)</code>
<code>cache_clean_unused</code>	<code>CDR:cache_clean_unused()</code>

---

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

```

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

```

```

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

```

**\_DESCRIPTION** Short text description of the module.

```

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

```

## 8 Return the module

```

392 return {
    Known fields are

393     _DESCRIPTION      = _DESCRIPTION,

    _VERSION to store ⟨version string⟩,

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

    date to store ⟨date string⟩,

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

    Various paths ,

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

    is_truthy

399     is_truthy         = is_truthy,

    escape

400     escape            = escape,

    make_directory

```

```

401 make_directory      = make_directory,

load_exec

402 load_exec           = load_exec,

403 load_exec_output    = load_exec_output,

record_line

404 record_line         = record_line,

highlight common

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

highlight code

408 highlight_code_setup = highlight_code_setup,

highlight_block_setup

409 highlight_block_setup = highlight_block_setup,
410 highlight_block_teardown = highlight_block_teardown,

cache

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

Internals

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

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

419 already              = false,

Other

420 dir_p                = dir_p,
421 json_p               = json_p,

Exportation

```

```

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

```

## File II

# coder-tool.py implementation

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

```

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

```

## 1 Usage

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

## 2 Header and global declarations

```

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

```

## 3 Options classes

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

```

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

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

```

### 3.1 TeXOpts class

```

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

```

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

```

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

```

### 3.2 PygOptsclass

`pygments` `LaTeXFormatter` options. Some of them may be deliberately unused. In particular, line numbering is governed by `fancyvrb` options. The description of these options is in a forthcoming section.

```

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

```

```

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

```

### 3.3 FVclass

```

68 class FVOpts(BaseOpts):
69     gobble = 0
70     tabsize = 4
71     linenosep = '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<sub>E</sub>X 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 {&lt;tag name&gt;} {&lt;relative key path&gt;}</code>
<code>\CDR_tag_get_path:c</code>	<code>*</code>	<code>\CDR_tag_get_path:c {&lt;relative key path&gt;}</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 <sub>E</sub> 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 {&lt;tag name&gt;} {&lt;relative key path&gt;} {&lt;true code&gt;} {&lt;false code&gt;} \CDR_tag_if_truthy:cTF {&lt;relative key path&gt;} {&lt;true code&gt;} {&lt;false code&gt;} </pre> <p>Execute <i>&lt;true code&gt;</i> when the property for <i>&lt;tag name&gt;</i> and <i>&lt;relative key path&gt;</i> is a truthy value, <i>&lt;false code&gt;</i> otherwise. A truthy value is a text which is not “false” in a case insensitive comparison. In the second version, the <i>&lt;tag name&gt;</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 {&lt;tag name&gt;} {&lt;relative key path&gt;} {&lt;value&gt;} {&lt;true code&gt;} {&lt;false code&gt;} \CDR_tag_if_eq:cnTF {&lt;relative key path&gt;} {&lt;value&gt;} {&lt;true code&gt;} {&lt;false code&gt;} </pre> <p>Execute <i>&lt;true code&gt;</i> when the property for <i>&lt;tag name&gt;</i> and <i>&lt;relative key path&gt;</i> is equal to <i>&lt;value&gt;</i>, <i>&lt;false code&gt;</i> otherwise. The comparison is based on <code>\str_compare:....</code>. In the second version, the <i>&lt;tag name&gt;</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`  $\{\langle token\ list\rangle\}$   $\{\langle true\ code\rangle\}$   $\{\langle false\ code\rangle\}$   
`\CDR_if_truthy:nTF` ★ Execute  $\langle true\ code\rangle$  when  $\langle token\ list\rangle$  is a truthy value,  $\langle false\ code\rangle$  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`  $\{\langle choice\rangle\}$   
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  $\text{\TeX}$  groups only. For running text code chunks, this module inherits from

1. `\c_CDR_tag_get/⟨tag name⟩` for the provided  $\langle tag\ name\rangle$ ,
2. `\c_CDR_tag_get/default.code`
3. `\c_CDR_tag_get/default`
4. `\c_CDR_tag_get/__pygments`
5. `\c_CDR_tag_get/__fancyvrb`

6. \c\_CDR\_tag\_get/\_\_\_fancyvrb.all when no using pygments

For text block code chunks, this module inherits from

1. \c\_CDR\_tag\_get/⟨name<sub>1</sub>⟩, ..., \c\_CDR\_tag\_get/⟨name<sub>n</sub>⟩ 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 and 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 and 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 }

```

---

<code>\CDR_get_inherit:cn</code> <code>\CDR_get_inherit:cf</code>	<code>\CDR_get_inherit:cn {&lt;child name&gt;} {&lt;parent names comma list&gt;}</code> Set the parents of <code>&lt;child name&gt;</code> to the given list.
--	--

---

```

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

```

## 7 Cache management

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

```

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 f expanded when used as n argument of l3keys 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:nTF { #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 }
```

---

```
\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*.

```
395 \cs_new_protected:Npn \CDR_local_set:n {
396   \CDR_tag_keys_set:nn { __local }
397 }
398 \cs_generate_variant:Nn \CDR_local_set:n { V }
```

### 9.1.1 Handling unknown tags

While using *\keys\_set:nn* and variants, each time a full key path matching the pattern *\c\_CDR\_tag/<tag name>/<relative key path>* is not recognized, we assume that the client implicitly wants a tag with the given *<tag name>* to be defined. For that purpose, we collect unknown keys with *\keys\_set\_known:nnnN* then process them to find each *<tag name>* and define the new tag accordingly. A similar situation occurs for display engine options where the full key path reads *\c\_CDR\_tag/<tag name>/<engine name>* engine options where *<engine name>* is not known in advance.

---

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

---

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

```
399 \cs_new_protected_nopar:Npn \CDR_tag_keys_inherit__:nnn #1 #2 #3 {
400   \keys_define:nn { #1 } { #2 .inherit:n = { #1 / #3 } }
401 }
402 \cs_new_protected_nopar:Npn \CDR_tag_keys_inherit_:nnn #1 #2 #3 {
403   \exp_args:Nnx
404   \use:n { \CDR_tag_keys_inherit__:nnn { #1 } { #2 } } {
405     \clist_use:nn { #3 } { ,#1/ }
406   }
407 }
408 \cs_new_protected_nopar:Npn \CDR_tag_keys_inherit:nn {
409   \exp_args:Nf
410   \CDR_tag_keys_inherit_:nnn { \CDR_tag_module:n { } }
411 }
```

---

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

---

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

```
412 \cs_new_protected_nopar:Npn \CDR_local_inherit:n {
413   \CDR_tag_keys_inherit:nn { __local }
414 }
```

---

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

---

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

```

415 \cs_new_protected_nopar:Npn \CDR_tag_keys_set_known__:nnN #1 #2 {
416   \keys_set_known:nnnN { #1 } { #2 } { #1 }
417 }
418 \cs_new_protected_nopar:Npn \CDR_tag_keys_set_known:nnN #1 {
419   \exp_args:Nf
420   \CDR_tag_keys_set_known__:nnN { \CDR_tag_module:n { #1 } }
421 }
422 \cs_generate_variant:Nn \CDR_tag_keys_set_known:nnN { nV }
423 \cs_new_protected_nopar:Npn \CDR_tag_keys_set_known:nN #1 #2 {
424   \CDR_tag_keys_set_known:nVN { #1 } #2 #2
425 }

```

---

```

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

```

---

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

```

426 \cs_new_protected_nopar:Npn \CDR_local_set_known:nN {
427   \CDR_tag_keys_set_known:nnN { __local }
428 }
429 \cs_generate_variant:Nn \CDR_local_set_known:nN { V }
430 \cs_new_protected_nopar:Npn \CDR_local_set_known:N #1 {
431   \CDR_local_set_known:VN #1 #1
432 }

```

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

```

433 \tl_set:Nn \l_CDR_tl { /([~/*])(?:/(.))*?$ } \use_none:n { $ }
434 \exp_args:NNf
435 \tl_put_left:Nn \l_CDR_tl { \CDR_tag_module:n { } }
436 \tl_put_left:Nn \l_CDR_tl { ^ }
437 \exp_args:NNV
438 \regex_const:Nn \c_CDR_provide_regex \l_CDR_tl

```

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

---

```

\@CDR@TEST
\CDR_tag_provide_from_kv:n

```

---

```

\CDR_tag_provide:n {<deep comma list>}
\CDR_tag_provide_from_kv:n {<key-value list>}

```

`<deep comma list>` has format `tag/<tag name comma list>`. Parse the `<key-value list>` for full key path matching `tag/<tag name>/<relative key path>`, then ensure that `\c_CDR_tag/<tag name>` is a known full key path. For that purpose, we use `\keyval_parse:nnn` with two `\CDR_tag_provide:` helper.

Notice that a tag name should contain no `/`. Implementation detail: uses `\l_CDR_tl`.

```

439 \regex_const:Nn \c_CDR_engine_regex { ^([~/*]+\sengine\soptions$ ) \use_none:n { $ }
440 \cs_new_protected_nopar:Npn \CDR_tag_provide:n #1 {
441   \CDR@Debug { \string\CDR_tag_provide:n: #1 }
442   \exp_args:NNf
443   \regex_extract_once:NnNTF \c_CDR_provide_regex {

```

```

444     \CDR_tag_module:n { .. } / #1
445 } \l_CDR_seq {
446     \tl_set:Nx \l_CDR_tl { \seq_item:Nn \l_CDR_seq 3 }
447     \exp_args:Nx
448     \clist_map_inline:nn {
449         \seq_item:Nn \l_CDR_seq 2
450     } {
451         \CDR_tag_keys_if_exist:nnF { } { ##1 } {
452             \CDR_tag_keys_inherit:nn { ##1 } {
453                 __pygments, __pygments.block,
454                 default.block, default.code, default, __tags, __engine,
455                 __fancyvrb, __fancyvrb.block, __fancyvrb.frame,
456                 __fancyvrb.number, __fancyvrb.all,
457             }
458             \CDR_tag_keys_define:nn { } {
459                 ##1 .code:n = \CDR_tag_keys_set:nn { ##1 } { #####1 },
460                 ##1 .value_required:n = true,
461             }
462 \CDR@Debug{\string\CDR_tag_provide:n \CDR_tag_module:n {##1} = ...}
463     }
464     \exp_args:NnV
465     \CDR_tag_keys_if_exist:nnF { ##1 } \l_CDR_tl {
466         \exp_args:NNV
467         \regex_match:NnT \c_CDR_engine_regex
468             \l_CDR_tl {
469             \exp_args:Nnf
470             \CDR_tag_keys_define:nn { ##1 } {
471                 \use:n { \l_CDR_tl } .code:n = \CDR_tag_set:n { #####1 },
472             }
473             \exp_args:Nnf
474             \CDR_tag_keys_define:nn { ##1 } {
475                 \use:n { \l_CDR_tl } .value_required:n = true,
476             }
477 \CDR@Debug{\string\CDR_tag_provide:n: \CDR_tag_module:n { ##1 } / \l_CDR_tl = ...}
478     }
479 }
480 }
481 } {
482     \regex_match:NnT \c_CDR_engine_regex { #1 } {
483         \CDR_tag_keys_define:nn { default } {
484             #1 .code:n = \CDR_tag_set:n { ##1 },
485             #1 .value_required:n = true,
486         }
487 \CDR@Debug{\string\CDR_tag_provide:n:C:\CDR_tag_module:n { default } / #1 = ...}
488     }
489 }
490 }
491 \cs_new:Npn \CDR_tag_provide:nn #1 #2 {
492     \CDR_tag_provide:n { #1 }
493 }
494 \cs_new:Npn \CDR_tag_provide_from_kv:n {
495     \keyval_parse:nnn {
496         \CDR_tag_provide:n
497     } {

```



```

498   \CDR_tag_provide:nn
499 }
500 }
501 \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

---

```

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

```

---

```

502 \sys_get_shell:nnN {which~pygmentize} {} \l_CDR_tl
503 \prg_new_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } { }
504 \tl_if_in:NnTF \l_CDR_tl { pygmentize } {
505   \prg_set_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } {
506     \prg_return_true:
507   }
508 } {
509   \prg_set_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } {
510     \prg_return_false:
511   }
512 }

```

### 9.2.2 \_\_pygments l3keys module

```

513 \CDR_tag_keys_define:nn { __pygments } {

```

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

```

514   lang .code:n = \CDR_tag_set:,
515   lang .value_required:n = true,

```

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

```

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

```

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

```

518   style .code:n = \CDR_tag_set:,
519   style .value_required:n = true,

```

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

```
520 commandprefix .code:n = \CDR_tag_set:,
521 commandprefix .value_required:n = true,
```

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

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

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

```
524 escapeinside .code:n = \CDR_tag_set:,
525 escapeinside .value_required:n = true,
```

🔴 **\_\_initialize** Initializer.

```
526 __initialize .meta:n = {
527   lang = tex,
528   pygments = \CDR_has_pygments:TF { true } { false },
529   style = default,
530   commandprefix = PY,
531   mathescape = false,
532   escapeinside = ,
533 },
534 __initialize .value_forbidden:n = true,

535 }
536 \AtBeginDocument{
537   \CDR_tag_keys_set:nn { __pygments } { __initialize }
538 }
```

### 9.2.3 `__pygments.block l3keys` module

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

🔴 **texcomments**[*=true|false*] If set to `true`, enables  $\LaTeX$  comment lines. That is,  $\LaTeX$  markup in comment tokens is not escaped so that  $\LaTeX$  can render it. Initially `false`.

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

🔴 **\_\_initialize** Initializer.

```
542 __initialize .meta:n = {
543   texcomments = false,
544 },
545 __initialize .value_forbidden:n = true,
```

```

546 }
547 \AtBeginDocument{
548   \CDR_tag_keys_set:n { __pygments.block } { __initialize }
549 }

```

## 9.3 Specific to coder

### 9.3.1 default l3keys module

```

550 \CDR_tag_keys_define:n { default } {

```

Keys are:

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

```

551   format .code:n = \CDR_tag_set:,
552   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.

```

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

```

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

```

555   debug .code:n = \CDR_tag_boolean_set:x { #1 },
556   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.

```

557   post~processor .code:n = \CDR_tag_set:,
558   post~processor .value_required:n = true,

```

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

```

559   parskip .code:n = \CDR_tag_set:,
560   parskip .value_required:n = true,

```

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

```

561   default~engine~options .code:n = \CDR_tag_set:,
562   default~engine~options .value_required:n = true,

```

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

```

563   default~options .code:n = \CDR_tag_set:,
564   default~options .value_required:n = true,

```

- **<engine name> engine options=<engine options>** to specify the options for the named engine,
- **<engine name> options=<coder options>** to specify the coder options that should apply when the named engine is selected.
- **\_\_initialize** to initialize storage properly. We cannot use **.initial:n** actions because the **\l\_keys\_path\_str** is not set up properly.

```

565 __initialize .meta:n = {
566     format = ,
567     cache = true,
568     debug = false,
569     post-processor = ,
570     parskip = \the\parskip,
571     default~engine~options = ,
572     default~options = ,
573 },
574 __initialize .value_forbidden:n = true,

575 }
576 \AtBeginDocument{
577   \CDR_tag_keys_set:nn { default } { __initialize }
578 }

```

### 9.3.2 default.code l3keys module

Void for the moment.

```

579 \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.

```

580 __initialize .meta:n = {
581 },
582 __initialize .value_forbidden:n = true,

583 }
584 \AtBeginDocument{
585   \CDR_tag_keys_set:nn { default.code } { __initialize }
586 }

```

### 9.3.3 \_\_tags l3keys module

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

```

587 \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.

```
588 tags .code:n = {
589     \clist_set:Nn \l_CDR_clist { #1 }
590     \clist_remove_duplicates:N \l_CDR_clist
591     \exp_args:NV
592     \CDR_tag_set:n \l_CDR_clist
593 },
594 tags .value_required:n = true,
```

🔴 **\_\_initialize** Initialization.

```
595 __initialize .meta:n = {
596     tags = ,
597 },
598 __initialize .value_forbidden:n = true,

599 }
600 \AtBeginDocument{
601     \CDR_tag_keys_set:nn { __tags } { __initialize }
602 }
```

### 9.3.4 **\_\_engine** l3keys module

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

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

Known keys include:

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

```
604 engine .code:n = \CDR_tag_set:,
605 engine .value_required:n = true,
```

🔴 **\_\_initialize** Initialization.

```
606 __initialize .meta:n = {
607     engine = default,
608 },
609 __initialize .value_forbidden:n = true,

610 }
611 \AtBeginDocument{
612     \CDR_tag_keys_set:nn { __engine } { __initialize }
613 }
```

### 9.3.5 **default.block** l3keys module

```
614 \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.

```
615 tags~format .code:n = \CDR_tag_set:,
616 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).

```
617 numbers~format .code:n = \CDR_tag_set:,
618 numbers~format .value_required:n = true,
```

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

```
619 show~tags .choices:nn =
620   { none, left, right, numbers, mirror }
621   { \CDR_tag_choices_set: },
622 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.

```
623 only~top .code:n = \CDR_tag_boolean_set:x { #1 },
624 only~top .default:n = true,
```

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

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

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

```
627 blockskip .code:n = \CDR_tag_set:,
628 blockskip .value_required:n = true,
```

● **\_\_initialize** Initialization.

```
629 __initialize .meta:n = {
630   show~tags = numbers,
631   only~top = true,
632   use~margin = true,
633   numbers~format = {
634     \sffamily
635     \scriptsize
636     \color{gray}
637   },
638   tags~format = {
639     \bfseries
640   },
641   blockskip = \topsep,
642 },
643 __initialize .value_forbidden:n = true,
```

```

644 }
645 \AtBeginDocument{
646   \CDR_tag_keys_set:nn { default.block } { __initialize }
647 }

```

## 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

```

648 \CDR_tag_keys_define:nn { __fancyvrb } {


```

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

```

649   formatcom .code:n = \CDR_tag_set:,
650   formatcom .value_required:n = true,


```

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

```

651   fontfamily .code:n = \CDR_tag_set:,
652   fontfamily .value_required:n = true,


```

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

```

653   fontsize .code:n = \CDR_tag_set:,
654   fontsize .value_required:n = true,


```

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

```

655   fontshape .code:n = \CDR_tag_set:,
656   fontshape .value_required:n = true,


```

 **fontseries**=*<series name>* L<sup>A</sup>T<sub>E</sub>X font series to use. Initially `auto`: the same as the current font.

```

657   fontseries .code:n = \CDR_tag_set:,
658   fontseries .value_required:n = true,

```

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

```

659   showspaces .code:n = \CDR_tag_boolean_set:x { #1 },
660   showspaces .default:n = true,

```

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

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

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

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

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

```
665 tabsize .code:n = \CDR_tag_set:,
666 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.

```
667 defineactive .code:n = \CDR_tag_set:,
668 defineactive .value_required:n = true,
```

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

```
669 relabel .code:n = \CDR_tag_set:,
670 relabel .value_required:n = true,
```

✅ **\_\_initialize** Initialization.

```
671 __initialize .meta:n = {
672   formatcom = ,
673   fontfamily = tt,
674   fontsize = auto,
675   fontseries = auto,
676   fontshape = auto,
677   showspaces = false,
678   showtabs = false,
679   obeytabs = false,
680   tabsize = 2,
681   defineactive = ,
682   relabel = ,
683 },
684 __initialize .value_forbidden:n = true,

685 }
686 \AtBeginDocument{
687   \CDR_tag_keys_set:nn { __fancyvrb } { __initialize }
688 }
```



### 9.4.2 `__fancyvrb.frame` l3keys module

Block specific options, frame related.

```
689 \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<sup>A</sup>T<sub>E</sub>X `\fboxsep` macro is added between the left vertical line and the text. Initially `none`: no frame.

```
690   frame .choices:nn =
691     { none, leftline, topline, bottomline, lines, single }
692     { \CDR_tag_choices_set: },
```

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

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

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

```
695   framesep .code:n = \CDR_tag_set:,
696   framesep .value_required:n = true,
```

- **rulecolor**=`<color command>` color of the frame rule, expressed in the standard L<sup>A</sup>T<sub>E</sub>X way. Initially black.

```
697   rulecolor .code:n = \CDR_tag_set:,
698   rulecolor .value_required:n = true,
```

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

```
699   fillcolor .code:n = \CDR_tag_set:,
700   fillcolor .value_required:n = true,
```

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

```
701   label .code:n = \CDR_tag_set:,
702   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.

```

703   labelposition .choices:nn =
704     { none, topline, bottomline, all }
705     { \CDR_tag_choices_set: },

```

✓ **\_\_initialize** Initialization.

```

706   __initialize .meta:n = {
707     frame = none,
708     framerule = 0.4pt,
709     framesep = \fboxsep,
710     rulecolor = black,
711     fillcolor = ,
712     label = ,
713     labelposition = none,% auto?
714   },
715   __initialize .value_forbidden:n = true,

716 }
717 \AtBeginDocument{
718   \CDR_tag_keys_set:nn { __fancyvrb.frame } { __initialize }
719 }

```

### 9.4.3 **\_\_fancyvrb.block l3keys** module

Block specific options, except numbering.

```

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

```

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

```

722   commentchar .code:n = \CDR_tag_set:,
723   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.

```

724   gobble .choices:nn = {
725     0,1,2,3,4,5,6,7,8,9
726   } {
727     \CDR_tag_choices_set:
728   },

```

● **baselinestretch**=*auto|<dimension>* value to give to the usual `\baselinestretch` L<sup>A</sup>T<sub>E</sub>X parameter. Initially `auto`: its current value just before the verbatim command.

```

729   baselinestretch .code:n = \CDR_tag_set:,
730   baselinestretch .value_required:n = true,

```

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

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

```
731 xleftmargin .code:n = \CDR_tag_set:,
732 xleftmargin .value_required:n = true,
```

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

```
733 xrightmargin .code:n = \CDR_tag_set:,
734 xrightmargin .value_required:n = true,
```

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

```
735 resetmargins .code:n = \CDR_tag_boolean_set:x { #1 },
736 resetmargins .default:n = true,
```

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

```
737 hfuzz .code:n = \CDR_tag_set:,
738 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`.

```
739 samepage .code:n = \CDR_tag_boolean_set:x { #1 },
740 samepage .default:n = true,
```

✅ **\_\_initialize** Initialization.

```
741 __initialize .meta:n = {
742   commentchar = ,
743   gobble = 0,
744   baselinestretch = auto,
745   resetmargins = true,
746   xleftmargin = 0pt,
747   xrightmargin = 0pt,
748   hfuzz = 2pt,
749   samepage = false,
750 },
751 __initialize .value_forbidden:n = true,

752 }
753 \AtBeginDocument{
754   \CDR_tag_keys_set:nn { __fancyvrb.block } { __initialize }
755 }
```

#### 9.4.4 `__fancyvrb.number` l3keys module

Block line numbering.

```
756 \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.

```
757   numbers .choices:nn =  
758     { none, left, right }  
759     { \CDR_tag_choices_set: },
```

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

```
760   numbersep .code:n = \CDR_tag_set:,  
761   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.

```
762   firstnumber .code:n = {  
763     \regex_match:NnTF \c_CDR_integer_regex { #1 } {  
764       \CDR_tag_set:  
765     } {  
766       \str_case:nnF { #1 } {  
767         { auto } { \CDR_tag_set: }  
768         { last } { \CDR_tag_set: }  
769       } {  
770         \PackageWarning  
771           { CDR }  
772           { Value~'#1'~not~in~auto,~last. }  
773       }  
774     }  
775   },  
776   firstnumber .value_required:n = true,
```

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

```
777   stepnumber .code:n = \CDR_tag_set:,  
778   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.

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

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

```

781 firstline .code:n = \CDR_tag_set:,
782 firstline .value_required:n = true,

```

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

```

783 lastline .code:n = \CDR_tag_set:,
784 lastline .value_required:n = true,

```

✅ **\_\_initialize** Initialization.

```

785 __initialize .meta:n = {
786   numbers = left,
787   numbersep = 1ex,
788   firstnumber = auto,
789   stepnumber = 1,
790   numberblanklines = true,
791   firstline = ,
792   lastline = ,
793 },
794 __initialize .value_forbidden:n = true,
795 }
796 \AtBeginDocument{
797   \CDR_tag_keys_set:nn { __fancyvrb.number } { __initialize }
798 }

```

#### 9.4.5 **\_\_fancyvrb.all** l3keys module

Options available when `pygments` is not used.

```

799 \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.

```

800 commandchars .code:n = \CDR_tag_set:,
801 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.

```

802 codes .code:n = \CDR_tag_set:,
803 codes .value_required:n = true,

```

✅ **\_\_initialize** Initialization.

```

804 __initialize .meta:n = {
805   commandchars = ,
806   codes = ,
807 },
808 __initialize .value_forbidden:n = true,

```

```

809 }
810 \AtBeginDocument{
811   \CDR_tag_keys_set:nn { __fancyvrb.all } { __initialize }
812 }

```

## 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

```

813 \keys_define:nn { CDR@Set } {


```

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

```

814   only~description .choices:nn = { false, true, {} } {
815     \int_compare:nNnTF \l_keys_choice_int = 1 {
816       \prg_set_conditional:Nnn \CDR_if_only_description: { p, T, F, TF } { \prg_return_true: }
817     } {
818       \prg_set_conditional:Nnn \CDR_if_only_description: { p, T, F, TF } { \prg_return_false: }
819     }
820   },
821   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`.

```

822   python-path .code:n = {
823     \str_set:Nn \l_CDR_str { #1 }
824     \lua_now:n { CDR:set_python_path('l_CDR_str') }
825   },
826 }

```

### 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.

```

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

828 \NewDocumentCommand \CDRSet { m } {
829 \CDR@Debug{ \string\CDRSet}
830 \CDR_set_preflight:n { #1 }
831 \keys_set_known:nnnN { CDR@Set } { #1 } { CDR@Set } \l_CDR_kv_clist
832 \clist_map_inline:nn {
833   __pygments, __pygments.block,
834   __tags, __engine, default.block, default.code, default,
835   __fancyvrb, __fancyvrb.frame, __fancyvrb.block, __fancyvrb.number, __fancyvrb.all
836 } {
837   \CDR_tag_keys_set_known:nN { ##1 } \l_CDR_kv_clist
838 \CDR@Debug{ Debug.CDRSet.1:##1/\l_CDR_kv_clist/ }
839 }
840 \CDR_tag_keys_set_known:nN { .. } \l_CDR_kv_clist
841 \CDR@Debug{ Debug.CDRSet.2:\CDR_tag_module:n { .. }//\l_CDR_kv_clist/ }
842 \CDR_tag_provide_from_kv:V \l_CDR_kv_clist
843 \CDR@Debug{ Debug.CDRSet.2a:\CDR_tag_module:n { .. }//\l_CDR_kv_clist/ }
844 \CDR_tag_keys_set_known:nN { .. } \l_CDR_kv_clist
845 \CDR@Debug{ Debug.CDRSet.3:\CDR_tag_module:n { .. }//\l_CDR_kv_clist/ }
846 \CDR_tag_keys_set:nV { default } \l_CDR_kv_clist
847 \CDR@Debug{ Debug.CDRSet.4:\CDR_tag_module:n { default } /\l_CDR_kv_clist/ }
848 \keys_define:nn { CDR@Set@tags } {
849   tags .code:n = {
850     \clist_set:Nn \g_CDR_tags_clist { ##1 }
851     \clist_remove_duplicates:N \g_CDR_tags_clist
852   },
853 }
854 \keys_set_known:nn { CDR@Set@tags } { #1 }
855 \ignorespaces
856 }
```

## 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.

```

857 \cs_new:Npn \CDR_export_get_path:cc #1 #2 {
858   CDR @ export @ get @ #1 / #2
859 }
```

---

<code>\CDR_export_set:ccn</code> <code>\CDR_export_set:Vcn</code> <code>\CDR_export_set:VcV</code>	<code>\CDR_export_set:ccn {&lt;file name&gt;} {&lt;relative key path&gt;} {&lt;value&gt;}</code> Store <i>&lt;value&gt;</i> , which is further retrieved with the instruction <code>\CDR_get_get:cc {&lt;file name&gt;} {&lt;relative key path&gt;}</code> . All the affectations are made at the current T <sub>E</sub> X group level.
--	--

---

```

860 \cs_new_protected:Npn \CDR_export_set:ccn #1 #2 #3 {
861   \cs_set:cpn { \CDR_export_get_path:cc { #1 } { #2 } } { \exp_not:n { #3 } }
862 }
863 \cs_new_protected:Npn \CDR_export_set:Vcn #1 {
864   \exp_args:NV
865   \CDR_export_set:ccn { #1 }
866 }
867 \cs_new_protected:Npn \CDR_export_set:VcV #1 #2 #3 {
868   \exp_args:NnV
869   \use:n {
870     \exp_args:NV \CDR_export_set:ccn #1 { #2 }
871   } #3
872 }

```

---

<code>\CDR_export_if_exist:ccTF</code> ★	<code>\CDR_export_if_exist:ccTF {&lt;file name&gt;} &lt;relative key path&gt; {&lt;true code&gt;} {&lt;false code&gt;}</code>
--	---

---

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

```

873 \prg_new_conditional:Nnn \CDR_export_if_exist:cc { p, T, F, TF } {
874   \cs_if_exist:ccTF { \CDR_export_get_path:cc { #1 } { #2 } } {
875     \prg_return_true:
876   } {
877     \prg_return_false:
878   }
879 }

```

---

<code>\CDR_export_get:cc</code> ★	<code>\CDR_export_get:cc {&lt;file name&gt;} {&lt;relative key path&gt;}</code>
-----------------------------------	---

---

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

```

880 \cs_new:Npn \CDR_export_get:cc #1 #2 {
881   \CDR_export_if_exist:ccT { #1 } { #2 } {
882     \use:c { \CDR_export_get_path:cc { #1 } { #2 } }
883   }
884 }

```

---

<code>\CDR_export_get:ccNTF</code>	<code>\CDR_export_get:ccNTF {&lt;file name&gt;} {&lt;relative key path&gt;} &lt;tl var&gt; {&lt;true code&gt;} {&lt;false code&gt;}</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.

```

885 \prg_new_protected_conditional:Nnn \CDR_export_get:ccN { T, F, TF } {
886   \CDR_export_if_exist:ccTF { #1 } { #2 } {
887     \tl_set:Nx #3 { \CDR_export_get:cc { #1 } { #2 } }

```



```

888     \prg_return_true:
889   } {
890     \prg_return_false:
891   }
892 }

```

## 11.2 Storage

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

```

893 \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.

```

894 \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.

```

895 \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.

```

896 \keys_define:nn { CDR@Export } {


```

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

```

897   file .tl_set:N = \l_CDR_file_tl,
898   file .value_required:n = true,


```

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

```

899   tags .code:n = {
900     \clist_set:Nn \l_CDR_clist { #1 }
901     \clist_remove_duplicates:N \l_CDR_clist
902     \prop_put:NVV \l_CDR_export_prop \l_keys_key_str \l_CDR_clist
903   },
904   tags .value_required:n = true,


```

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

```

905   lang .code:n = {
906     \prop_put:NVN \l_CDR_export_prop \l_keys_key_str { #1 }
907   },
908   lang .value_required:n = true,

```

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

```

909 preamble .code:n = {
910   \prop_put:NVn \l_CDR_export_prop \l_keys_key_str { #1 }
911 },
912 preamble .value_required:n = true,

```

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

```

913 postamble .code:n = {
914   \prop_put:NVn \l_CDR_export_prop \l_keys_key_str { #1 }
915 },
916 postamble .value_required:n = true,

```

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

```

917 raw .choices:nn = { false, true, {} } {
918   \prop_put:NVx \l_CDR_export_prop \l_keys_key_str {
919     \int_compare:nNnTF
920       \l_keys_choice_int = 1 { false } { true }
921   }
922 },

```

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

```

923 once .choices:nn = { false, true, {} } {
924   \prop_put:NVx \l_CDR_export_prop \l_keys_key_str {
925     \int_compare:nNnTF
926       \l_keys_choice_int = 1 { false } { true }
927   }
928 },

```

✅ **\_\_initialize** Meta key to properly initialize all the variables.

```

929 __initialize .meta:n = {
930   __initialize_prop = #1,
931   file =,
932   tags =,
933   lang = tex,
934   preamble =,
935   postamble =,
936   raw = false,
937   once = true,
938 },
939 __initialize .default:n = \l_CDR_export_prop,

```

✅ **\_\_initialize\_prop** Goody: properly initialize the local property storage.

```

940 __initialize_prop .code:n = \prop_clear:N #1,
941 __initialize_prop .value_required:n = true,
942 }

```

## 11.4 Implementation

```

943 \NewDocumentCommand \CDRExport { m } {
944   \keys_set:nn { CDR@Export } { __initialize }
945   \keys_set:nn { CDR@Export } { #1 }
946   \tl_if_empty:NTF \l_CDR_file_tl {
947     \PackageWarning
948       { coder }
949     { Missing~export~key~‘file’ }
950   } {
951     \CDR_export_set:VcV \l_CDR_file_tl { file } \l_CDR_file_tl
952     \prop_map_inline:Nn \l_CDR_export_prop {
953       \CDR_export_set:Vcn \l_CDR_file_tl { ##1 } { ##2 }
954     }

```

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.

```

955   \prop_get:NnNTF \l_CDR_export_prop { tags } \l_CDR_clist {
956     \tl_if_empty:NTF \l_CDR_clist {
957       \PackageWarning
958         { coder }
959       { Missing~export~key~‘tags’ }
960     } {
961       \clist_set_eq:NN \g_CDR_tags_clist \l_CDR_clist
962       \clist_remove_duplicates:N \g_CDR_tags_clist
963       \clist_put_left:NV \g_CDR_all_tags_clist \l_CDR_clist
964       \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`.

```

965       \exp_args:NV
966       \CDR_export_get:ccNT \l_CDR_file_tl { lang } \l_CDR_tl {
967         \clist_map_inline:Nn \g_CDR_tags_clist {
968           \CDR_tag_set:ccV { ##1 } { lang } \l_CDR_tl
969         }
970       }
971     }
972     \seq_put_left:NV \g_CDR_export_seq \l_CDR_file_tl
973   } {
974     \PackageWarning
975       { coder }
976     { Missing~export~key~‘tags’ }
977   }
978 }
979 \ignorespaces
980 }

```

Files are created at the end of the typesetting process.

```

981 \AddToHook { enddocument / end } {
982   \seq_map_inline:Nn \g_CDR_export_seq {
983     \str_set:Nx \l_CDR_str { #1 }
984     \lua_now:n { CDR:export_file('l_CDR_str') }
985     \clist_map_inline:nn {

```

```

986     tags, raw, once, preamble, postamble
987   } {
988     \CDR_export_get:ccNT { #1 } { ##1 } \l_CDR_tl {
989       \exp_args:NNx
990       \str_set:Nn \l_CDR_str { \l_CDR_tl }
991       \lua_now:n {
992         CDR:export_file_info('##1','l_CDR_str')
993       }
994     }
995   }
996   \lua_now:n { CDR:export_complete() }
997 }
998 }

```

## 12 Style

pygments, through `coder-tool.py`, creates style commands, but the storage is managed on the L<sup>A</sup>T<sub>E</sub>X side by `coder.sty`. This is a L<sup>A</sup>T<sub>E</sub>X style API.

---

<code>\CDR@StyleDefine</code>	<code>\CDR@StyleDefine {&lt;pygments style name&gt;} {&lt;definitions&gt;}</code>
-------------------------------	---

---

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

```

999 \cs_set:Npn \CDR@StyleDefine #1 {
1000   \tl_gset:cn { g_CDR@Style/#1 }
1001 }

```

---

<code>\CDR@StyleUse</code>	<code>\CDR@StyleUse {&lt;pygments style name&gt;}</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.

```

1002 \cs_set:Npn \CDR@StyleUse #1 {
1003   \tl_use:c { g_CDR@Style/#1 }
1004 }
1005 \cs_set:Npn \CDR@StyleUseTag {
1006   \CDR@StyleUse { \CDR_tag_get:c { style } }
1007 }

```

---

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

---

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

```

1008 \prg_new_conditional:Nnn \CDR@StyleIfExist:c { TF } {
1009   \tl_if_exist:cTF { g_CDR@Style/#1 } {
1010     \prg_return_true:
1011   } {
1012     \prg_return_false:
1013   }
1014 }
1015 \cs_set_eq:NN \CDR@StyleIfExist \CDR@StyleIfExist:cTF

```

## 13 Creating display engines

### 13.1 Utilities

---

<code>\CDRCode_engine:c</code>	<code>*</code>	<code>\CDRCode_engine:c {&lt;engine name&gt;}</code>
<code>\CDRCode_engine:V</code>	<code>*</code>	<code>\CDRBlock_engine:c {&lt;engine name&gt;}</code>
<code>\CDRBlock_engine:c</code>	<code>*</code>	<code>\CDRCode_engine:c</code> builds a command sequence name based on <code>&lt;engine name&gt;</code> . <code>\CDRBlock_engine:c</code>
<code>\CDRBlock_engine:V</code>	<code>*</code>	builds an environment name based on <code>&lt;engine name&gt;</code> .

---

```
1016 \cs_new:Npn \CDRCode_engine:c #1 {
1017   CDR@colored/code/#1:nn
1018 }
1019 \cs_new:Npn \CDRBlock_engine:c #1 {
1020   CDR@colored/block/#1
1021 }
1022 \cs_new:Npn \CDRCode_engine:V {
1023   \exp_args:NV \CDRCode_engine:c
1024 }
1025 \cs_new:Npn \CDRBlock_engine:V {
1026   \exp_args:NV \CDRBlock_engine:c
1027 }
```

---

<code>\CDRCode_options:c</code>	<code>*</code>	<code>\CDRCode_options:c {&lt;engine name&gt;}</code>
<code>\CDRCode_options:V</code>	<code>*</code>	<code>\CDRBlock_options:c {&lt;engine name&gt;}</code>
<code>\CDRBlock_options:c</code>	<code>*</code>	<code>\CDRCode_options:c</code> builds a command sequence name based on <code>&lt;engine name&gt;</code> used
<code>\CDRBlock_options:V</code>	<code>*</code>	to store the comma list of key value options. <code>\CDRBlock_options:c</code> builds a command

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

```
1028 \cs_new:Npn \CDRCode_options:c #1 {
1029   CDR@colored/code~options/#1:nn
1030 }
1031 \cs_new:Npn \CDRBlock_options:c #1 {
1032   CDR@colored/block~options/#1
1033 }
1034 \cs_new:Npn \CDRCode_options:V {
1035   \exp_args:NV \CDRCode_options:c
1036 }
1037 \cs_new:Npn \CDRBlock_options:V {
1038   \exp_args:NV \CDRBlock_options:c
1039 }
```

---

<code>\CDRCode_options_use:c</code>	<code>*</code>	<code>\CDRCode_options_use:c {&lt;engine name&gt;}</code>
<code>\CDRCode_options_use:V</code>	<code>*</code>	<code>\CDRBlock_options_use:c {&lt;engine name&gt;}</code>
<code>\CDRBlock_options_use:c</code>	<code>*</code>	<code>\CDRCode_options_use:c</code> builds a command sequence name based on <code>&lt;engine name&gt;</code>
<code>\CDRBlock_options_use:V</code>	<code>*</code>	and use it. <code>\CDRBlock_options:c</code> builds a command sequence name based on <code>&lt;engine</code>

`name>` and use it.

```
1040 \cs_new:Npn \CDRCode_options_use:c #1 {
1041   \CDRCode_if_options:cT { #1 } {
1042     \use:c { \CDRCode_options:c { #1 } }
```

```

1043 }
1044 }
1045 \cs_new:Npn \CDRBlock_options_use:c #1 {
1046   \CDRBlock_if_options:cT { #1 } {
1047     \use:c { \CDRBlock_options:c { #1 } }
1048   }
1049 }
1050 \cs_new:Npn \CDRCode_options_use:V {
1051   \exp_args:NV \CDRCode_options_use:c
1052 }
1053 \cs_new:Npn \CDRBlock_options_use:V {
1054   \exp_args:NV \CDRBlock_options_use:c
1055 }

```

`\l_CDR_engine_tl` Storage for an engine name.

```

1056 \tl_new:N \l_CDR_engine_tl

```

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

---

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

---

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

## 13.2 Implementation

---

<code>\CDRCodeEngineNew</code>	<code>\CDRCodeEngineNew {&lt;engine name&gt;}{&lt;engine body&gt;}</code>
<code>\CDRCodeEngineRenew</code>	<code>\CDRCodeEngineRenew{&lt;engine name&gt;}{&lt;engine body&gt;}</code>

---

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

```

1057 \NewDocumentCommand \CDRCodeEngineNew { mO{}m } {
1058   \exp_args:Nx
1059   \tl_if_empty:nTF { #1 } {
1060     \PackageWarning
1061       { coder }
1062       { The~engine~cannot~be~void. }
1063   } {
1064     \cs_new:cpn { \CDRCode_engine:c {#1} } ##1 ##2 {
1065       \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
1066       #3
1067     }
1068     \ignorespaces
1069   }
1070 }

1071 \NewDocumentCommand \CDRCodeEngineRenew { mO{}m } {
1072   \exp_args:Nx
1073   \tl_if_empty:nTF { #1 } {

```

```

1074     \PackageWarning
1075     { coder }
1076     { The~engine~cannot~be~void. }
1077     \use_none:n
1078   } {
1079     \cs_if_exist:cTF { \CDRCode_engine:c { #1 } } {
1080       \cs_set:cpn { \CDRCode_engine:c { #1 } } ##1 ##2 {
1081         \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
1082         #3
1083       }
1084     } {
1085       \PackageWarning
1086       { coder }
1087       { No~code~engine~#1.}
1088     }
1089     \ignorespaces
1090   }
1091 }

```

---

**\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.

```

1092 \cs_new_protected:Npn \CDR@CodeEngineApply {
1093   \CDRCode_if_engine:cF { \CDR_tag_get:c { engine } } {
1094     \PackageError
1095     { coder }
1096     { \CDR_tag_get:c { engine }~code~engine~unknown,~replaced~by~‘default’ }
1097     {See~\CDRCodeEngineNew~in~the~coder~manual}
1098     \CDR_tag_set:cn { engine } { default }
1099   }
1100   \CDR_tag_get:c { format }
1101   \exp_args:Nnx
1102   \use:c { \CDRCode_engine:c { \CDR_tag_get:c { engine } } } {
1103     \CDR_tag_get:c { \CDR_tag_get:c { engine }~engine~options },
1104     \CDR_tag_get:c { engine~options }
1105   }
1106 }

```

---

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

---

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

---

Create a L<sup>A</sup>T<sub>E</sub>X 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.

```

1107 \NewDocumentCommand \CDRBlockEngineNew { m0{}m } {
1108   \cs_set:cpn { \CDRBlock_options:c { #1 } } { \exp_not:n { #2 } }
1109   \NewDocumentEnvironment { \CDRBlock_engine:c { #1 } } { m } {
1110     \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
1111     #3
1112   }
1113 }

1114 \NewDocumentCommand \CDRBlockEngineRenew { mm } {
1115   \tl_if_empty:nTF { #1 } {
1116     \PackageWarning
1117       { coder }
1118       { The~engine~cannot~be~void. }
1119     \use_none:n
1120   } {
1121     \RenewDocumentEnvironment { \CDRBlock_engine:c { #1 } } { m } {
1122       \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
1123       #2
1124     }
1125   }
1126 }

```

---

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

---

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

```

1127 \cs_new:Npn \CDRBlock_engine_begin: {
1128   \CDRBlock_if_engine:cF { \CDR_tag_get:c { engine } } {
1129     \PackageError
1130       { coder }
1131       { \CDR_tag_get:c { engine }~block~engine~unknown,~replaced~by~‘default’ }
1132       { See~\CDRBlockEngineNew~in~the~coder~manual }
1133     \CDR_tag_set:cn { engine } { default }
1134   }
1135   \exp_args:Nnx
1136   \use:c { \CDRBlock_engine:c \CDR_tag_get:c { engine } } {
1137     \CDR_tag_get:c { \CDR_tag_get:c { engine }~engine~options },
1138     \CDR_tag_get:c { engine~options },
1139   }
1140 }
1141 \cs_new:Npn \CDRBlock_engine_end: {
1142   \use:c { end \CDRBlock_engine:c \CDR_tag_get:c { engine } }
1143 }
1144 %   \begin{MacroCode}
1145 %
1146 % \subsection{Conditionals}
1147 %
1148 % \begin{function}[EXP,TF]{\CDRCode_if_engine:c}
1149 % \begin{syntax}
1150 % \cs{CDRCode_if_engine:cTF} \Arg{engine name} \Arg{true code} \Arg{false code}
1151 % \end{syntax}
1152 % If there exists a code engine with the given \metatt{engine name},

```



```

1153 % execute \metatt{true code}.
1154 % Otherwise, execute \metatt{false code}.
1155 % \end{function}
1156 % \begin{MacroCode}[OK]
1157 \prg_new_conditional:Nnn \CDRCode_if_engine:c { p, T, F, TF } {
1158   \cs_if_exist:CTF { \CDRCode_engine:c { #1 } } {
1159     \prg_return_true:
1160   } {
1161     \prg_return_false:
1162   }
1163 }
1164 \prg_new_conditional:Nnn \CDRCode_if_engine:V { p, T, F, TF } {
1165   \cs_if_exist:CTF { \CDRCode_engine:V #1 } {
1166     \prg_return_true:
1167   } {
1168     \prg_return_false:
1169   }
1170 }

```

---

\CDRBlock\_if\_engine:cTF ★ \CDRBlock\_if\_engine:c {<engine name>} {<true code>} {<false code>}

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

```

1171 \prg_new_conditional:Nnn \CDRBlock_if_engine:c { p, T, F, TF } {
1172   \cs_if_exist:CTF { \CDRBlock_engine:c { #1 } } {
1173     \prg_return_true:
1174   } {
1175     \prg_return_false:
1176   }
1177 }
1178 \prg_new_conditional:Nnn \CDRBlock_if_engine:V { p, T, F, TF } {
1179   \cs_if_exist:CTF { \CDRBlock_engine:V #1 } {
1180     \prg_return_true:
1181   } {
1182     \prg_return_false:
1183   }
1184 }

```

---

\CDRCode\_if\_options:cTF ★ \CDRCode\_if\_options:cTF {<engine name>} {<true code>} {<false code>}

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

```

1185 \prg_new_conditional:Nnn \CDRCode_if_options:c { p, T, F, TF } {
1186   \cs_if_exist:CTF { \CDRCode_options:c { #1 } } {
1187     \prg_return_true:
1188   } {
1189     \prg_return_false:
1190   }
1191 }
1192 \prg_new_conditional:Nnn \CDRCode_if_options:V { p, T, F, TF } {
1193   \cs_if_exist:CTF { \CDRCode_options:V #1 } {

```

```

1194     \prg_return_true:
1195   } {
1196     \prg_return_false:
1197   }
1198 }

```

---

```

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

```

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

```

1199 \prg_new_conditional:Nnn \CDRBlock_if_options:c { p, T, F, TF } {
1200   \cs_if_exist:cTF { \CDRBlock_options:c { #1 } } {
1201     \prg_return_true:
1202   } {
1203     \prg_return_false:
1204   }
1205 }
1206 \prg_new_conditional:Nnn \CDRBlock_if_options:V { p, T, F, TF } {
1207   \cs_if_exist:cTF { \CDRBlock_options:V #1 } {
1208     \prg_return_true:
1209   } {
1210     \prg_return_false:
1211   }
1212 }

```

### 13.3 Default code engine

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

```

1213 \CDRCodeEngineNew { default } { #2 }

```

### 13.4 efbox code engine

```

1214 \AtBeginDocument {
1215   \@ifpackageloaded{efbox} {
1216     \CDRCodeEngineNew {efbox} {
1217       \efbox[#1]{#2}
1218     }
1219   } {}
1220 }

```

### 13.5 Block mode default engine

```

1221 \CDRBlockEngineNew {default} {
1222 } {
1223 }

```

### 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.

```
1224 \cs_new:Npn \CDR@DefineSp {
1225   \CDR_tag_if_truthy:cTF { showspaces } {
1226     \cs_set:Npn \CDR@Sp {{\FancyVerbSpace}}
1227   } {
1228     \cs_set_eq:NN \CDR@Sp \space
1229   }
1230 }
```

---

**\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.


```
1231 \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.


```
1232 \CDR_tag_keys_define:nn { __code } {
```

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

```
1233   tag .tl_set:N = \l_CDR_tag_tl,
1234   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.

```
1235   engine~options .code:n = \CDR_tag_set:,
1236   engine~options .value_required:n = true,
```

 **\_\_initialize** initialize

```
1237   __initialize .meta:n = {
1238     tag = default,
1239     engine~options = ,
1240   },
1241   __initialize .value_forbidden:n = true,
1242 }
```

## 14.4 Implementation

---

`\CDRCodeformat:` `\CDRCodeformat:`

---

Private utility to setup the formatting.

```

1243 \cs_new:Npn \CDR_brace_if_contains_comma:n #1 {
1244   \tl_if_in:nnTF { #1 } { , } { { #1 } } { #1 }
1245 }
1246 \cs_generate_variant:Nn \CDR_brace_if_contains_comma:n { V }
1247 \cs_new:Npn \CDRCodeformat: {
1248   \frenchspacing
1249   \CDR_tag_get:cN { baselinestretch } \l_CDR_tl
1250   \str_if_eq:VnF \l_CDR_tl { auto } {
1251     \exp_args:NNV
1252     \def \baselinestretch \l_CDR_tl
1253   }
1254   \CDR_tag_get:cN { fontfamily } \l_CDR_tl
1255   \str_if_eq:VnT \l_CDR_tl { tt } { \tl_set:Nn \l_CDR_tl { lmtt } }
1256   \exp_args:NV
1257   \fontfamily \l_CDR_tl
1258   \clist_map_inline:nn { series, shape } {
1259     \CDR_tag_get:cN { font##1 } \l_CDR_tl
1260     \str_if_eq:VnF \l_CDR_tl { auto } {
1261       \exp_args:NnV
1262       \use:c { font##1 } \l_CDR_tl
1263     }
1264   }
1265   \CDR_tag_get:cN { fontsize } \l_CDR_tl
1266   \str_if_eq:VnF \l_CDR_tl { auto } {
1267     \tl_use:N \l_CDR_tl
1268   }
1269   \selectfont
1270 % \@noligs ?? this is in fancyvrb but does not work here as is
1271 }

1272 \NewDocumentCommand \CDRCode { O{} } {
1273   \group_begin:
1274   \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
1275     \prg_return_false:
1276   }
1277   \clist_set:Nn \FV@KeyValues { #1 }
1278   \CDR_local_inherit:n {
1279     __code, __engine, default.code, __pygments, default,
1280   }
1281   \CDR_local_set_known:N \FV@KeyValues
1282   \CDR_tag_provide_from_kv:V \FV@KeyValues
1283   \CDR_local_set_known:N \FV@KeyValues
1284   \CDR_local_inherit:n {
1285     __fancyvrb,
1286   }
1287   \CDR_local_set:V \FV@KeyValues
1288   \CDR_get_inherit:cf { __local } {
1289     \tl_if_empty:NF \l_CDR_tag_tl { \l_CDR_tag_tl, }

```

```

1290   __code, __engine, default.code, __pygments, default, __fancyvrb,
1291 }
1292 \CDRCode:n
1293 }

```

---

**\CDRCode:n**    \CDRCode:n *<delimiter>*

---

Main utility used by \CDRCode.

```

1294 \cs_set:Npn \CDRCode:n #1 {
1295   \CDR_tag_if_truthy:cTF {pygments} {
1296     \cs_set:Npn \CDR@StyleUseTag {
1297       \CDR@StyleUse { \CDR_tag_get:c { style } }
1298       \cs_set_eq:NN \CDR@StyleUseTag \prg_do_nothing:
1299     }
1300     \DefineShortVerb { #1 }
1301     \SaveVerb [
1302       aftersave = {
1303         \exp_args:Nx \UndefineShortVerb { #1 }
1304         \lua_now:n { CDR:highlight_code_setup() }
1305         \CDR_tag_get:cN {lang} \l_CDR_tl
1306         \lua_now:n { CDR:highlight_set_var('lang') }
1307         \CDR_tag_get:cN {cache} \l_CDR_tl
1308         \lua_now:n { CDR:highlight_set_var('cache') }
1309         \CDR_tag_get:cN {debug} \l_CDR_tl
1310         \lua_now:n { CDR:highlight_set_var('debug') }
1311         \CDR_tag_get:cN {style} \l_CDR_tl
1312         \lua_now:n { CDR:highlight_set_var('style') }
1313         \lua_now:n { CDR:highlight_set_var('source', 'FV@SV@CDR@Source') }
1314         \FV@UseKeyValues
1315         \frenchspacing
1316         % \FV@SetupFont Break
1317         \FV@DefineWhiteSpace
1318         \FancyVerbDefineActive
1319         \FancyVerbFormatCom
1320         \CDRCodeformat:
1321         \CDR@DefineSp
1322         \CDR_tag_get:c { format }
1323         \CDR@CodeEngineApply {
1324           \CDR@StyleIfExist { \CDR_tag_get:c { style } } {
1325             \CDR@StyleUseTag
1326             \lua_now:n { CDR:highlight_source(false, true) }
1327           } {
1328             \lua_now:n { CDR:highlight_source(true, true) }
1329             \input { \l_CDR_pyg_sty_tl }
1330             \CDR@StyleUseTag
1331           }
1332           \makeatletter
1333           \lua_now:n {
1334             CDR.synctex_tag = tex.get_synctex_tag();
1335             CDR.synctex_line = tex.inputlineno;
1336             tex.set_synctex_mode(1)
1337           }
1338           \input { \l_CDR_pyg_tex_tl }\ignorespaces

```

```

1339         \lua_now:n {
1340             tex.set_synctex_mode(0)
1341         }
1342         \makeatother
1343     }
1344     \group_end:
1345
1346 }
1347 ] { CDR@Source } #1
1348 } {
1349     \DefineShortVerb { #1 }
1350     \SaveVerb [
1351         aftersave = {
1352             \UndefineShortVerb { #1 }
1353             \cs_set_eq:NN \CDR@FormattingPrep \FV@FormattingPrep
1354             \cs_set:Npn \FV@FormattingPrep {
1355                 \CDR@FormattingPrep
1356                 \CDR_tag_get:c { format }
1357             }
1358             \CDR@CodeEngineApply { \mbox {
1359                 \FV@UseKeyValues
1360                 \FV@FormattingPrep
1361                 \FV@SV@CDR@Code
1362             } }
1363             \group_end:
1364         }
1365     ] { CDR@Code } #1
1366 }
1367 }

```

## 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.


```
1368 \CDR_tag_keys_define:nn { __block } {
```

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

```
1369 no-export .code:n = \CDR_tag_boolean_set:x { #1 },
1370 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.

```
1371 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.

```

1372 dry~numbers .code:n = \CDR_tag_set:,
1373 dry~numbers .default:n = true,

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

1374 test .code:n = \CDR_tag_boolean_set:x { #1 },
1375 test .default:n = true,

🔴 engine options=(engine options) options forwarded to the engine. They are ap-
    pended to the options given with key (engine name) engine options. Mainly
    a convenient user interface shortcut.

1376 engine~options .code:n = \CDR_tag_set:,
1377 engine~options .value_required:n = true,

🔴 __initialize initialize

1378 __initialize .meta:n = {
1379     no~export = false,
1380     no~export~format = ,
1381     dry~numbers = false,
1382     test = false,
1383     engine~options = ,
1384 },
1385 __initialize .value_forbidden:n = true,
1386 }

```

## 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

```

1387 \CDR_int_new:cn { __start } { 0 }
1388 \CDR_int_new:cn { __step } { 0 }
1389 \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.

```

1390 \clist_map_inline:nn { i, ii, iii, iv } {
1391     \cs_set_eq:cc { CDR@ListProcessLine@ #1 } { FV@ListProcessLine@ #1 }
1392 }

```

---

```

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

```

---

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

```

1393 \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 ??.)*

```
1394 \seq_new:N \l_CDR_vrb_seq
```

---

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

```
1395 \cs_new:Npn \FVB@CDRBlock {
1396   \@bsphack
1397   \exp_args:NV \CDRBlock_preflight:n \FV@KeyValues
1398   \begingroup
1399   \lua_now:n {
1400     CDR.synctex_tag = tex.get_synctex_tag();
1401     CDR.synctex_line = tex.inputlineno;
1402     tex.set_synctex_mode(1)
1403   }
1404   \seq_clear:N \l_CDR_vrb_seq
1405   \cs_set_protected_nopar:Npn \FV@ProcessLine ##1 {
1406     \seq_put_right:Nn \l_CDR_vrb_seq { ##1 }
1407   }
1408   \FV@Scan
1409 }
```

---

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

```
1410 \cs_new:Npn \FVE@CDRBlock {%
1411   \CDRBlock_setup:
1412   \CDR_if_no_export:F {
1413     \seq_map_inline:Nn \l_CDR_vrb_seq {
1414       \tl_set:Nn \l_CDR_tl { ##1 }
1415       \lua_now:n { CDR:record_line('l_CDR_tl') }
1416     }
1417   }
1418   \CDRBlock_engine_begin:
1419   \CDR_if_pygments:TF {
1420     \CDRBlock@Pyg
1421   } {
1422     \CDRBlock@FV
1423   }
1424   \lua_now:n {
1425     tex.set_synctex_mode(0);
1426     CDR.synctex_line = 0;
1427   }
1428   \CDRBlock_engine_end:
1429   \CDRBlock_teardown:
1430   \endgroup
1431   \@esphack
```



```

1432 }
1433 \DefineVerbatimEnvironment{CDRBlock}{CDRBlock}{-}
1434 % \begin{MacroCode}
1435 \cs_new_protected_nopar:Npn \CDRBlock_setup: {
1436   \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
1437     \prg_return_true:
1438   }
1439   \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.

```

1440 \CDRBlock_setup_tags_and_engine:
1441 \CDR_local_inherit:n {
1442   __block, __pygments.block, default.block,
1443   __pygments, default
1444 }
1445 \CDR_local_set_known:N \FV@KeyValues
1446 \CDR_tag_provide_from_kv:V \FV@KeyValues
1447 \CDR_local_set_known:N \FV@KeyValues
1448 \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`.

```

1449 \CDR_local_inherit:n {
1450   \CDR_tag_if_eq:cnF { engine } { default } {
1451     __fancyvrb.frame,
1452   },
1453   __fancyvrb.number,
1454 }
1455 \CDR_local_set_known:N \FV@KeyValues

```

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

```

1456 \CDR_local_inherit:n {
1457   __fancyvrb.block,
1458   __fancyvrb,
1459 }
1460 \CDR_local_set_known:VN \FV@KeyValues \l_CDR_kv_clist
1461 \lua_now:n {
1462   CDR:highlight_block_setup('g_CDR_tags_clist')
1463 }
1464 \CDR_set_conditional:Nn \CDR_if_pygments:
1465   { \CDR_tag_if_truthy_p:c { pygments } }
1466 \CDR_set_conditional:Nn \CDR_if_no_export:
1467   { \CDR_tag_if_truthy_p:c { no-export } }
1468 \CDR_set_conditional:Nn \CDR_if_dry_numbers:
1469   { \CDR_tag_if_truthy_p:c { dry-numbers } }
1470 \CDR_set_conditional:Nn \CDR_if_number_on:
1471   { ! \CDR_tag_if_eq_p:cn { numbers } { none } }

```

```

1472 \CDR_set_conditional:Nn \CDR_tags_if_already: {
1473   \CDR_tag_if_truthy_p:c { only-top } &&
1474   \CDR_clist_if_eq_p:NN \g_CDR_tags_clist \g_CDR_last_tags_clist
1475 }
1476 \CDR_if_number_on:T {
1477   \clist_map_inline:Nn \g_CDR_tags_clist {
1478     \CDR_int_if_exist:cF { ##1 } {
1479       \CDR_int_new:cn { ##1 } { 1 }
1480     }
1481   }
1482 }
1483 }

1484 \cs_new_protected_nopar:Npn \CDRBlock_teardown: {
1485   \CDR_if_dry_numbers:F {
1486     \tl_set:Nx \l_CDR_tl { \seq_count:N \l_CDR_vrb_seq }
1487     \clist_map_inline:Nn \g_CDR_tags_clist {
1488       \CDR_int_gadd:cn { ##1 } { \l_CDR_tl }
1489     }
1490   }
1491   \lua_now:n {
1492     CDR:highlight_block_teardown()
1493   }
1494 }

```

#### 15.2.4 pygments only

Parts of CDRBlock environment specific to pygments.

---

<u>\CDRBlock@Pyg</u>	\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.

```

1495 \cs_set_protected:Npn \CDRBlock@Pyg {
1496 \CDR@Debug {\string\CDRBlock@Pyg/\the\inputlineno}
1497 \CDR_tag_get:cN {lang} \l_CDR_tl
1498 \lua_now:n { CDR:highlight_set_var('lang') }
1499 \CDR_tag_get:cN {cache} \l_CDR_tl
1500 \lua_now:n { CDR:highlight_set_var('cache') }
1501 \CDR_tag_get:cN {debug} \l_CDR_tl
1502 \lua_now:n { CDR:highlight_set_var('debug') }
1503 \CDR_tag_get:cN {texcomments} \l_CDR_tl
1504 \lua_now:n { CDR:highlight_set_var('texcomments') }
1505 \CDR_tag_get:cN {escapeinside} \l_CDR_tl
1506 \lua_now:n { CDR:highlight_set_var('escapeinside') }
1507 \CDR_tag_get:cN {mathescape} \l_CDR_tl
1508 \lua_now:n { CDR:highlight_set_var('mathescape') }
1509 \CDR_tag_get:cN {style} \l_CDR_tl
1510 \lua_now:n { CDR:highlight_set_var('style') }
1511 \CDR@StyleIfExist { \l_CDR_tl } { } {
1512   \lua_now:n { CDR:highlight_source(true, false) }
1513   \input { \l_CDR_pyg_sty_tl }
1514 }
1515 \CDR@StyleUseTag

```

```

1516 % \CDR_tag_get:c { format }
1517 % \cs_set:Npn \CDR:nn ##1 ##2 {
1518 %\CDR@Debug{Debug.CDRBlock.FV.KEYS:\tl_to_str:n{##1}->\tl_to_str:n{##2}}
1519 % \fvset{ ##1 = ##2, }
1520 % }
1521 % \clist_map_inline:Nn \c_CDRBlock@Pyg_clist {
1522 % \exp_args:Nnx
1523 % \CDR:nn { ##1 } { \CDR_tag_get:c { ##1 } }
1524 % }
1525 \CDR@DefineSp
1526 \lua_now:n { CDR:highlight_source(false, true) }
1527 \fvset{ commandchars=\\{\} }
1528 \FV@UseVerbatim {
1529 \CDR_tag_get:c { format }
1530 \CDR_if_no_export:T {
1531 \CDR_tag_get:c { no~export~format }
1532 }
1533 \makeatletter
1534 \input{ \l_CDR_pyg_tex_tl }
1535 \makeatother
1536 \def \FV@ProcessLine {}
1537 }
1538 \CDR_if_number_on:T {
1539 \CDR_int_add:cn { __last } { 1 }
1540 \clist_map_inline:Nn \g_CDR_tags_clist {
1541 \CDR_int_gset:cc { ##1 } { __last }
1542 \CDR@Debug {DEBUG.CDRBlock.LAST: ##1 -> \CDR_int_use:c { ##1 } }
1543 }
1544 }
1545 }

```

\c\_CDRBlock@Pyg\_clist

*(End definition for \c\_CDRBlock@Pyg\_clist. This variable is documented on page ??.)*

```

1546 \clist_const:Nn \c_CDRBlock@Pyg_clist {
1547 % __fancyvrb:
1548 formatcom,% = ,
1549 fontfamily,% = tt,
1550 fontsize,% = auto,
1551 fontseries,% = auto,
1552 fontshape,% = auto,
1553 showspaces,% = false,
1554 showtabs,% = false,
1555 obeytabs,% = false,
1556 tabsize,% = 2,
1557 defineactive,% = ,
1558 reflabel,% = ,
1559 % __fancyvrb.frame:
1560 frame,% = none,
1561 framerule,% = 0.4pt,
1562 framesep,% = \fboxsep,
1563 rulecolor,% = black,
1564 fillcolor,% = ,
1565 label,% = ,

```

```

1566 labelposition,% = none,% auto?
1567 % __fancyvrb.block:
1568 % commentchar,% = ,
1569 % gobble,% = 0,
1570 baselinestretch,% = auto,
1571 resetmargins,% = true,
1572 xleftmargin,% = 0pt,
1573 xrightmargin,% = 0pt,
1574 hfuzz,% = 2pt,
1575 samepage,% = false,
1576 % __fancyvrb.number
1577 % numbers,% = left,
1578 % numbersep,% = 1ex,
1579 % firstnumber,% = auto,
1580 % stepnumber,% = 1,
1581 % numberblanklines,% = true,
1582 % firstline,% = ,
1583 % lastline,% = ,
1584 }

```

### Info

```

1585 \cs_new:Npn \CDR@NumberFormat {
1586   \CDR_tag_get:c { numbers~format }
1587 }
1588 \cs_new:Npn \CDR@NumberSep {
1589   \hspace{ \CDR_tag_get:c { numbersep } }
1590 }
1591 \cs_new:Npn \CDR@TagsFormat {
1592   \CDR_tag_get:c { tags~format }
1593 }

```

---

<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>	

---

```

1594 \cs_new:Npn \CDR_info_N_L:n #1 {
1595   \hbox_overlap_left:n {
1596     \cs_set:Npn \baselinestretch { 1 }
1597     { \CDR@NumberFormat
1598       #1
1599     }
1600     \CDR@NumberSep
1601   }
1602 }
1603 \cs_new:Npn \CDR_info_T_L:n #1 {
1604   \hbox_overlap_left:n {
1605     \cs_set:Npn \baselinestretch { 1 }
1606     \CDR@NumberFormat
1607     \smash{
1608       \parbox[b]{\marginparwidth}{

```

```

1609     \raggedleft
1610     { \CDR@TagsFormat \g_CDR_tags_clist :}
1611     }
1612     #1
1613   }
1614   \CDR@NumberSep
1615 }
1616 }
1617 \cs_new:Npn \CDR_info_N_R:n #1 {
1618   \hbox_overlap_right:n {
1619     \CDR@NumberSep
1620     \cs_set:Npn \baselinestretch { 1 }
1621     \CDR@NumberFormat
1622     #1
1623   }
1624 }
1625 \cs_new:Npn \CDR_info_T_R:n #1 {
1626   \hbox_overlap_right:n {
1627     \cs_set:Npn \baselinestretch { 1 }
1628     \CDR@NumberSep
1629     \CDR@NumberFormat
1630     \smash {
1631       \parbox[b]{\marginparwidth}{
1632         \raggedright
1633         #1:
1634         {\CDR@TagsFormat \space \g_CDR_tags_clist}
1635       }
1636     }
1637   }
1638 }

```

---

`\CDR_number_alt:n` First line.

```

1639 \cs_set:Npn \CDR_number_alt:n #1 {
1640   \use:c { CDRNumber
1641     \CDR_if_number_visible:nTF { #1 } { Main } { Other }
1642   } { #1 }
1643 }
1644 \cs_set:Npn \CDR_number_alt: {
1645   \CDR@Debug{ALT: \CDR_int_use:c { __ } }
1646   \CDR_number_alt:n { \CDR_int_use:c { __ } }
1647 }

```

---

<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.

```

1648 \cs_new:Npn \CDRNumberMain {
1649 }

```

```

1650 \cs_new:Npn \CDRNumberOther {
1651         \use_none:n
1652 }

```

\CDR@NumberMain	\CDR@NumberMain
\CDR@NumberOther	\CDR@NumberOther

Respectively apply \CDR@NumberMain or \CDR@NumberOther on \CDR\_int\_use:c { \_\_ }

```

1653 \cs_new:Npn \CDR@NumberMain {
1654         \CDRNumberMain { \CDR_int_use:c { __ } }
1655 }
1656 \cs_new:Npn \CDR@NumberOther {
1657         \CDRNumberOther { \CDR_int_use:c { __ } }
1658 }

```

**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.

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

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.

```

1659 \cs_new:Npn \CDR_line_N_N:n {
1660 \CDR@Debug {Debug.CDR_line_N_N:n}
1661 \CDR_line_box_N:n
1662 }
1663
1664 \cs_new:Npn \CDR_line_L_N:n #1 {
1665 \CDR@Debug {Debug.CDR_line_L_N:n}
1666 \CDR_line_box:nnn { \CDR_info_T_L:n { } } { #1 } { }
1667 }
1668
1669 \cs_new:Npn \CDR_line_R_N:n #1 {
1670 \CDR@Debug {Debug.CDR_line_R_N:n}
1671 \CDR_line_box:nnn { } { #1 } { \CDR_info_T_R:n { } }
1672 }
1673
1674 \cs_new:Npn \CDR_line_S_N:n {
1675 \CDR@Debug {Debug.CDR_line_S_N:n}
1676 \CDR_line_box_N:n
1677 }
1678
1679 \cs_new:Npn \CDR_line_O_N:n {
1680 \CDR@Debug {STEP:CDR_line_O_N:n}
1681 \CDR_line_box_N:n
1682 }
1683
1684 \cs_new:Npn \CDR_line_N_L:n #1 {
1685 \CDR@Debug {STEP:CDR_line_N_L:n}
1686 \CDR_if_no_number:TF {

```

```

1687 \CDR_line_box:nnn {
1688     \CDR_info_N_L:n { \CDR@NumberMain }
1689 } { #1 } {}
1690 } {
1691     \CDR_line_box_L:n { #1 }
1692 }
1693 }
1694
1695 \cs_new:Npn \CDR_line_L_L:n #1 {
1696 \CDR@Debug {STEP:CDR_line_L_L:n}
1697 \CDR_if_number_single:TF {
1698     \CDR_line_box:nnn {
1699         \CDR_info_T_L:n { \space \CDR@NumberMain }
1700     } { #1 } {}
1701 } {
1702     \CDR_if_no_number:TF {
1703         \cs_set:Npn \CDR@@Line {
1704             \cs_set:Npn \CDR@@Line {
1705                 \CDR_line_box_L:nn { \CDR_info_N_L:n { \CDR@NumberOther } }
1706             }
1707             \CDR_line_box_L:nn { \CDR_info_N_L:n { \CDR@NumberMain } }
1708         }
1709     } {
1710         \cs_set:Npn \CDR@@Line {
1711             \CDR_line_box_L:nn { \CDR_info_N_L:n { \CDR_number_alt: } }
1712         }
1713     }
1714     \CDR_line_box:nnn { \CDR_info_T_L:n { } } { #1 } { }
1715 }
1716 }
1717
1718 \cs_new:Npn \CDR_line_R_R:n #1 {
1719 \CDR@Debug {STEP:CDR_line_R_R:n}
1720 \CDR_if_number_single:TF {
1721     \CDR_line_box:nnn { } { #1 } {
1722         \CDR_info_T_R:n { \CDR@NumberMain }
1723     }
1724 } {
1725     \CDR_if_no_number:TF {
1726         \cs_set:Npn \CDR@@Line {
1727             \cs_set:Npn \CDR@@Line {
1728                 \CDR_line_box_R:nn { \CDR_info_N_R:n { \CDR@NumberOther } }
1729             }
1730             \CDR_line_box_R:nn { \CDR_info_N_R:n { \CDR@NumberMain } }
1731         }
1732     } {
1733         \cs_set:Npn \CDR@@Line {
1734             \CDR_line_box_R:nn { \CDR_info_N_R:n { \CDR_number_alt: } }
1735         }
1736     }
1737     \CDR_line_box:nnn { } { #1 } { \CDR_info_T_R:n { } }
1738 }
1739 }
1740

```

```

1741 \cs_new:Npn \CDR_line_R_L:n #1 {
1742 \CDR@Debug {STEP:CDR_line_R_L:n}
1743 \CDR_line_box:nnn {
1744 \CDR_if_no_number:TF {
1745 \CDR_info_N_L:n { \CDR@NumberMain }
1746 } {
1747 \CDR_info_N_L:n { \CDR_number_alt: }
1748 }
1749 } { #1 } {
1750 \CDR_info_T_R:n { }
1751 }
1752 }
1753
1754 \cs_set_eq:NN \CDR_line_S_L:n \CDR_line_L_L:n
1755 \cs_set_eq:NN \CDR_line_O_L:n \CDR_line_R_L:n
1756
1757 \cs_new:Npn \CDR_line_N_R:n {
1758 \typeout {STEP:CDR_line_N_R:n}
1759 \CDR_line_box_R:n
1760 }
1761
1762 \cs_new:Npn \CDR_line_L_R:n #1 {
1763 \CDR@Debug {STEP:CDR_line_L_R:n}
1764 \CDR_line_box:nnn {
1765 \CDR_info_T_L:n { }
1766 } { #1 } {
1767 \CDR_if_no_number:TF {
1768 \CDR_info_N_R:n { \CDR@NumberMain }
1769 } {
1770 \CDR_info_N_R:n { \CDR_number_alt: }
1771 }
1772 }
1773 }
1774
1775 \cs_set_eq:NN \CDR_line_S_R:n \CDR_line_R_R:n
1776 \cs_set_eq:NN \CDR_line_O_R:n \CDR_line_L_R:n
1777
1778
1779 \cs_new:Npn \CDR_line_box_N:n #1 {
1780 \CDR@Debug {STEP:CDR_line_box_N:n}
1781 \CDR_line_box:nnn { } { #1 } {}
1782 }
1783
1784 \cs_new:Npn \CDR_line_box_L:n #1 {
1785 \CDR@Debug {STEP:CDR_line_box_L:n}
1786 \CDR_line_box:nnn {
1787 \CDR_info_N_L:n { \CDR_number_alt: }
1788 } { #1 } {}
1789 }
1790
1791 \cs_new:Npn \CDR_line_box_R:n #1 {
1792 \CDR@Debug {STEP:CDR_line_box_R:n}
1793 \CDR_line_box:nnn { } { #1 } {
1794 \CDR_info_N_R:n { \CDR_number_alt: }

```



```

1795 }
1796 }

```

---

<code>\CDR_line_box:nnn</code>	<code>\CDR_line_box:nnn {&lt;left info&gt;} {&lt;line content&gt;} {&lt;right info&gt;}</code>
<code>\CDR_line_box_L:nn</code>	<code>\CDR_line_box_L:nn {&lt;left info&gt;} {&lt;line content&gt;}</code>
<code>\CDR_line_box_R:nn</code>	<code>\CDR_line_box_R:nn {&lt;right info&gt;} {&lt;line content&gt;}</code>
<code>\CDR_line_box:nn</code>	Returns an hbox with the given material. The first LR command is the reference, from which are derived the L, R and N commands. At run time the <code>\CDR_line_box:nn</code> is defined to call one of the above commands (with the same signarture).

---

```

1797 \cs_new:Npn \CDR_line_box:nnn #1 #2 #3 {
1798 \CDR@Debug {\string\CDR_line_box:nnn/\tl_to_str:n{#1}/.../\tl_to_str:n{#3}/}
1799 \directlua {
1800   tex.set_synctex_tag( CDR.synctex_tag )
1801 }
1802 \lua_now:e {
1803   tex.set_synctex_line(CDR.synctex_line+( \CDR_int_use:c { __ } ) )
1804 }
1805 \hbox to \hsize {
1806   \kern \leftmargin
1807   #1
1808   \hbox to \linewidth {
1809     \FV@LeftListFrame
1810     #2
1811     \hss
1812     \FV@RightListFrame
1813   }
1814   #3
1815 }
1816 }
1817 \cs_new:Npn \CDR_line_box_L:nn #1 #2 {
1818   \CDR_line_box:nnn { #1 } { #2 } {}
1819 }
1820 \cs_new:Npn \CDR_line_box_R:nn #1 #2 {
1821 \CDR@Debug {STEP:CDR_line_box_R:nn}
1822   \CDR_line_box:nnn { } {#2} { #1 }
1823 }
1824 \cs_new:Npn \CDR_line_box_N:nn #1 #2 {
1825 \CDR@Debug {STEP:CDR_line_box_N:nn}
1826   \CDR_line_box:nnn { } { #2 } {}
1827 }

```

## Lines

```

1828 \cs_new:Npn \CDR@Line {
1829 \CDR@Debug {\string\CDR@Line}
1830   \peek_meaning_ignore_spaces:NTF [%]
1831   { \CDR_line:nnn } {
1832     \PackageError { code } { Missing~ '['~%]
1833     ~at~first~\string\CDR@Line~call }
1834   }
1835 }

```

---

<code>\CDR_line:nnn</code>	<code>\CDR_line:nnn {&lt;CDR@Line kv list&gt;} {&lt;line number&gt;} {&lt;line content&gt;}</code>
----------------------------	--

---

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.

```

1836 \keys_define:nn { CDR@Line } {
1837   last .code:n = \CDR_int_set:cn { __last } { #1 },
1838 }
1839 \cs_new:Npn \CDR_line:nnn [ #1 ] #2 {
1840   \CDR@Debug {\string\CDR_line:nnn}
1841   \keys_set:nn { CDR@Line } { #1 }
1842   \CDR_int_set:cn { __ } { 0 }
1843   \CDR_if_number_on:TF {
1844     \CDR_tag_if_eq:cnTF { firstnumber } { last } {
1845       \clist_map_inline:Nn \g_CDR_tags_clist {
1846         \clist_map_break:n {
1847           \CDR_int_set:cc { __start } { ##1 }
1848         }
1849       }
1850     }
1851   } {
1852     \CDR_tag_if_eq:cnTF { firstnumber } { auto } {
1853       \CDR_int_set:cn { __start } { 1 }
1854     } {
1855       \CDR_int_set:cn { __start } { \CDR_tag_get:c { firstnumber } }
1856     }
1857   }

```

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

```

1858   \CDR_set_conditional:Nn \CDR_if_number_single: {
1859     \CDR_int_compare_p:cNn { __last } = 1
1860   }
1861   \CDR@Debug{***** TEST: \CDR_if_number_single:TF { SINGLE } { MULTI } }
1862   \CDR_int_add:cn { __last } { \CDR_int:c { __start } - 1 }
1863   \CDR_int_set:cn { __step } { \CDR_tag_get:c { stepnumber } }
1864   \CDR@Debug {CDR_line:nnn:START/STEP/LAST=\CDR_int_use:c { __start }/\CDR_int_use:c { __step } /\

```

---

<code>\CDR_if_visible_at_index_p:n *</code>	<code>\CDR_if_visible_at_index:nTF {&lt;relative line number&gt;} {&lt;true code&gt;}</code>
<code>\CDR_if_visible_at_index:nTF *</code>	<code>{&lt;false code&gt;}</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.

```

1865   \CDR_int_compare:cNnTF { __step } < 2 {
1866     \CDR_int_set:cn { __step } { 1 }
1867     \CDR_set_conditional_alt:Nn \CDR_if_visible_at_index:n {
1868       ! \CDR_int_compare_p:cNn { __last } < { ##1 + \CDR_int:c { __start } - 1 }

```

```

1869     }
1870     \CDR_set_conditional_alt:Nn \CDR_if_number_visible:n {
1871       ! \CDR_int_compare_p:cNn { __last } < { ##1 }
1872     }
1873   } {
1874     \CDR_set_conditional_alt:Nn \CDR_if_visible_at_index:n {
1875       ! \CDR_int_compare_p:cNn { __last } < { ##1 + \CDR_int:c { __start } - 1 }
1876     }
1877     \CDR_set_conditional_alt:Nn \CDR_if_number_visible:n {
1878       \int_compare_p:nNn {
1879         ( ##1 + \CDR_int:c { __start } - 1 )
1880         / \CDR_int:c { __step } * \CDR_int:c { __step }
1881         - \CDR_int:c { __start } + 1
1882       } = { ##1 }
1883       && \CDR_if_visible_at_index_p:n { ##1 }
1884     }
1885   }
1886   \CDR@Debug {CDR_line:nnn:1}

1887   \CDR_set_conditional:Nn \CDR_if_no_number: {
1888     \CDR_int_compare_p:cNn { __start } > {
1889       \CDR_int:c { __last } / \CDR_int:c { __step } * \CDR_int:c { __step }
1890     }
1891   }
1892   \cs_set:Npn \CDR@Line ##1 {
1893     \CDR@Debug {\string\CDR@Line(A), \the\inputlineno}
1894     \CDR_int_set:cn { __ } { ##1 + \CDR_int:c { __start } - #2 }
1895     \CDR@@Line
1896   }
1897   \CDR_int_set:cn { __ } { \CDR_int:c { __start } + 1 - #2 }
1898 } {
1899 \CDR@Debug {NUMBER-OFF}
1900   \cs_set:Npn \CDR@Line ##1 {
1901     \CDR@Debug {\string\CDR@Line(B), \the\inputlineno}
1902     \CDR@@Line
1903   }
1904 }
1905 \CDR@Debug {STEP_S, \CDR_int_use:c {__step}, \CDR_int_use:c {__last} }

```

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

```

1906   \tl_clear:N \l_CDR_tl
1907   \CDR_tags_if_already:TF {
1908     \tl_put_right:Nn \l_CDR_tl { _N }
1909   } {
1910     \exp_args:Nx
1911     \str_case:nnF { \CDR_tag_get:c { show-tags } } {
1912       { left } { \tl_put_right:Nn \l_CDR_tl { _L } }
1913       { right } { \tl_put_right:Nn \l_CDR_tl { _R } }
1914       { none } { \tl_put_right:Nn \l_CDR_tl { _N } }
1915       { numbers } { \tl_put_right:Nn \l_CDR_tl { _S } }
1916       { mirror } { \tl_put_right:Nn \l_CDR_tl { _O } }

```

```

1917 } { \PackageError
1918     { coder }
1919     { Unknown~show~tags~options~::~ \CDR_tag_get:c { show~tags } }
1920 }
1921 }

```

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

```

1922 \exp_args:Nx
1923 \str_case:nnF { \CDR_tag_get:c { numbers } } {
1924     { left } {
1925         \tl_put_right:Nn \l_CDR_tl { _L }
1926         \cs_set:Npn \CDR@@Line { \CDR_line_box_L:n }
1927     }
1928     { right } {
1929         \tl_put_right:Nn \l_CDR_tl { _R }
1930         \cs_set:Npn \CDR@@Line { \CDR_line_box_R:n }
1931     }
1932     { none } {
1933         \tl_put_right:Nn \l_CDR_tl { _N }
1934         \cs_set:Npn \CDR@@Line { \CDR_line_box_N:n }
1935     }
1936 } { \PackageError
1937     { coder }
1938     { Unknown~numbers~options~::~ \CDR_tag_get:c { numbers } }
1939 }
1940 \CDR@Debug {BRANCH:CDR_line \l_CDR_tl :n}
1941 \use:c { CDR_line \l_CDR_tl :n }
1942 }

```

### 15.2.5 fancyvrb only

pygments is not used, fall back to fancyvrb features.

---

```

CDRBlock@FV \CDRBlock@Fv

```

---

```

1943 \cs_new_protected:Npn \CDRBlock@FV {
1944 \CDR@Debug {DEBUG.Block.FV}
1945 % \tl_clear:N \FV@KeyValues
1946 % \cs_set:Npn \CDR:nn ##1 ##2 {
1947 %\CDR@Debug{Debug.CDRBlock.FV.KEYS:\tl_to_str:n{##1}->\tl_to_str:n{##2}}
1948 % \fvset{ ##1 = { ##2 }, }
1949 % }
1950 % \clist_map_inline:Nn \c_CDRBlock@FV_clist {
1951 % \exp_args:Nnf
1952 % \CDR:nn { ##1 } { \CDR_tag_get:c { ##1 } }
1953 % }
1954 \FV@UseKeyValues
1955 \FV@UseVerbatim {
1956 \CDR_tag_get:c { format }
1957 \CDR_if_no_export:T {
1958 \CDR_tag_get:c { no~export~format }
1959 }

```

```

1960 \tl_set:Nx \l_CDR_tl { [ last=]
1961 \seq_count:N \l_CDR_vrb_seq %[
1962 ] }
1963 \seq_map_indexed_inline:Nn \l_CDR_vrb_seq {
1964 \exp_last_unbraced:NV \CDR@Line \l_CDR_tl { ##1 } { ##2 }
1965 \tl_clear:N \l_CDR_tl
1966 }
1967 \tl_clear:N \FV@ProcessLine
1968 }
1969 \CDR_if_number_on:T {
1970 \CDR_int_compare:cNnTF { __ } > 0 {
1971 \CDR_int_set:cn { __ } {
1972 \value{FancyVerbLine} - \CDR_int_use:c { __ } + 1
1973 }
1974 \clist_map_inline:Nn \g_CDR_tags_clist {
1975 \CDR_int_gadd:cc { ##1 } { __ }
1976 }
1977 } {
1978 \CDR_int_set:cn { __ } { \value{FancyVerbLine} + 1 }
1979 \clist_map_inline:Nn \g_CDR_tags_clist {
1980 \CDR_int_gset:cc { ##1 } { __ }
1981 \CDR@Debug { DEBUG.CDRBlock.FV.Last: ##1/\CDR_int_use:c { ##1 } }
1982 }
1983 }
1984 }
1985 }

```

\c\_CDRBlock@FV\_clist

(End definition for \c\_CDRBlock@FV\_clist. This variable is documented on page ??.)

```

1986 \clist_const:Nn \c_CDRBlock@FV_clist {
1987 % __fancyvrb:
1988 formatcom,% = ,
1989 fontfamily,% = tt,
1990 fontsize,% = auto,
1991 fontseries,% = auto,
1992 fontshape,% = auto,
1993 showspaces,% = false,
1994 showtabs,% = false,
1995 obeytabs,% = false,
1996 tabsize,% = 2,
1997 defineactive,% = ,
1998 reflabel,% = ,
1999 % __fancyvrb.frame:
2000 frame,% = none,
2001 framerule,% = 0.4pt,
2002 framesep,% = \fboxsep,
2003 rulecolor,% = black,
2004 fillcolor,% = ,
2005 label,% = ,
2006 labelposition,% = none,% auto?
2007 % __fancyvrb.block:
2008 % commentchar,% = ,
2009 gobble,% = 0,

```

```

2010 baselinestretch,% = auto,
2011 resetmargins,% = true,
2012 xleftmargin,% = Opt,
2013 xrightmargin,% = Opt,
2014 hfuzz,% = 2pt,
2015 samepage,% = false,
2016 % __fancyvrb.number
2017 numbers,% = none,
2018 numbersep,% = 1ex,
2019 firstnumber,% = auto,
2020 stepnumber,% = 1,
2021 numberblanklines,% = true,
2022 firstline,% = ,
2023 lastline,% = ,
2024 }

```

### 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.

```

2025 \cs_new:Npn \CDR_set_conditional:Nn #1 #2 {
2026   \bool_if:nTF { #2 } {
2027     \prg_set_conditional:Nnn #1 { p, T, F, TF } { \prg_return_true: }
2028   } {
2029     \prg_set_conditional:Nnn #1 { p, T, F, TF } { \prg_return_false: }
2030   }
2031 }

```

---

```

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

```

---

Wrapper over \prg\_set\_conditional:Nnn.

```

2032 \cs_new:Npn \CDR_set_conditional_alt:Nn #1 #2 {
2033   \prg_set_conditional:Nnn #1 { p, T, F, TF } {
2034     \bool_if:nTF { #2 } { \prg_return_true: } { \prg_return_false: }
2035   }
2036 }

```

---

```

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

```

---

```

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

```

---

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

```

2037 \prg_set_conditional:Nnn \CDR_if_middle_column: { p, T, F, TF } { \prg_return_false: }
2038 \prg_set_conditional:Nnn \CDR_if_right_column: { p, T, F, TF } { \prg_return_false: }

```

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

---

```

\CDR_tags_if_visible_p:n * \CDR_tags_if_visible:nTF {\left|right)} {\true code} {\false code}
\CDR_tags_if_visible:nTF *

```

---

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

```

2039 \prg_set_conditional:Nnn \CDR_tags_if_visible:n { p, T, F, TF } {
2040   \bool_if:nTF {
2041     ( \CDR_tag_if_eq_p:cn { show-tags } { ##1 } ||
2042       \CDR_tag_if_eq_p:cn { show-tags } { numbers } &&
2043       \CDR_tag_if_eq_p:cn { numbers } { ##1 }
2044     ) && ! \CDR_tags_if_already_p:
2045   } {
2046     \prg_return_true:
2047   } {
2048     \prg_return_false:
2049   }
2050 }

```

---

```

\CDRBlock_setup_tags_and_engine:

```

---

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

```

2051 \cs_new_protected_nopar:Npn \CDRBlock_setup_tags_and_engine: {
2052   \CDR_local_inherit:n { __tags }
2053   \CDR_local_set_known:N \FV@KeyValues
2054   \CDR_tag_if_exist_here:ccT { __local } { tags } {
2055     \CDR_tag_get:cN { tags } \l_CDR_clist
2056     \clist_if_empty:NF \l_CDR_clist {
2057       \clist_gset_eq:NN \g_CDR_tags_clist \l_CDR_clist
2058     }
2059   }
2060   \clist_if_empty:NT \g_CDR_tags_clist {
2061     \PackageWarning
2062       { coder }
2063       { No~(default)~tags~provided. }
2064   }
2065   \CDR@Debug {CDRBlock_setup_tags:\space\g_CDR_tags_clist}

```

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

```

2066   \CDR_get_inherit:cf { __local } {
2067     \g_CDR_tags_clist,
2068     __block, __tags, __engine, default.block, __pygments.block,
2069     __fancyvrb.block __fancyvrb.frame, __fancyvrb.number,
2070     __pygments, default, __fancyvrb,
2071   }

```

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

```

2072   \clist_map_inline:Nn \g_CDR_tags_clist {
2073     \CDR_int_if_exist:cF { ##1 } {
2074       \CDR_int_new:cn { ##1 } { 1 }
2075     }
2076   }

```

Now setup the engine options if any.

```

2077 \CDR_local_inherit:n { __engine }
2078 \CDR_local_set_known:N \FV@KeyValues
2079 \CDR_tag_get:cNT { engine } \l_CDR_tl {
2080   \clist_put_left:Nx \FV@KeyValues { \CDRBlock_options_use:V \l_CDR_tl }
2081 }
2082 }

```

## 16 Management

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

```

2083 \bool_new:N \g_CDR_in_impl_bool

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

```

---

<code>\CDR_if_show_code_p: *</code>	<code>\CDR_if_show_code:TF {⟨true code⟩} {⟨false code⟩}</code>
<code>\CDR_if_show_code:TF *</code>	Execute <i>⟨true code⟩</i> when code should be printed, <i>⟨false code⟩</i> otherwise.

---

```

2084 \prg_new_conditional:Nnn \CDR_if_show_code: { p, T, F, TF } {
2085   \bool_if:nTF {
2086     \g_CDR_in_impl_bool && !\g_CDR_with_impl_bool
2087   } {
2088     \prg_return_false:
2089   } {
2090     \prg_return_true:
2091   }
2092 }

```

`\g_CDR_with_impl_bool`

```

2093 \bool_new:N \g_CDR_with_impl_bool

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

```

---

<code>\CDRPreamble</code>	<code>\CDRPreamble {⟨variable⟩} {⟨file name⟩}</code>
	Store the content of <i>⟨file name⟩</i> into the variable <i>⟨variable⟩</i> . This is currently unstable.

---

```

2094 \DeclareDocumentCommand \CDRPreamble { m m } {
2095   \msg_info:nnn
2096     { coder }
2097     { :n }
2098     { Reading-preamble-from-file-"#2". }
2099   \tl_set:Nn \l_CDR_tl { #2 }
2100   \exp_args:NNx
2101   \tl_set:Nx #1 { \lua_now:n {CDR.print_file_content('l_CDR_tl')} }
2102 }

```



## 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

```

2103 \newcounter{CDR@impl@page}
2104 \DeclareDocumentCommand \CDRImplementation {} {
2105   \bool_if:NF \g_CDR_with_impl_bool {
2106     \clearpage
2107     \bool_gset_true:N \g_CDR_in_impl_bool
2108     \let\CDR@old@part\part
2109     \DeclareDocumentCommand\part{som}{}
2110     \let\CDR@old@section\section
2111     \DeclareDocumentCommand\section{som}{}
2112     \let\CDR@old@subsection\subsection
2113     \DeclareDocumentCommand\subsection{som}{}
2114     \let\CDR@old@subsubsection\subsubsection
2115     \DeclareDocumentCommand\subsubsection{som}{}
2116     \let\CDR@old@paragraph\paragraph
2117     \DeclareDocumentCommand\paragraph{som}{}
2118     \let\CDR@old@subparagraph\subparagraph
2119     \DeclareDocumentCommand\subparagraph{som}{}
2120     \cs_if_exist:NT \refsection{ \refsection }
2121     \setcounter{ CDR@impl@page }{ \value{page} }
2122   }
2123 }
2124 \DeclareDocumentCommand\CDRFinale {} {
2125   \bool_if:NF \g_CDR_with_impl_bool {
2126     \clearpage
2127     \bool_gset_false:N \g_CDR_in_impl_bool
2128     \let\part\CDR@old@part
2129     \let\section\CDR@old@section
2130     \let\subsection\CDR@old@subsection
2131     \let\subsubsection\CDR@old@subsubsection
2132     \let\paragraph\CDR@old@paragraph
2133     \let\subparagraph\CDR@old@subparagraph
2134     \setcounter { page } { \value{ CDR@impl@page } }
2135   }
2136 }
2137 %\cs_set_eq:NN \CDR_line_number: \prg_do_nothing:

```

## 19 Finale

```

2138 %\AddToHook { cmd/FancyVerbFormatLine/before } {
2139 %   \CDR_line_number:
2140 %}

```

2141  
2142 \ExplSyntaxOff  
2143  
2144 %</sty>