

`coder` — code inlined in a \LaTeX document^{*}

Jérôme LAURENS[†]

Released 2022/02/07

Abstract

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

This \LaTeX package requires Lua \TeX and may use syntax coloring based on `pygments`.

1 Package dependencies

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

2 Similar technologies

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

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

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

3 Known bugs and limitations

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

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

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

4 Namespace and conventions

\LaTeX 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.

5 Presentation

`coder` is a triptych of three complementary components

1. `coder.sty`, on the \LaTeX side,
2. `coder-util.lua`, to manage some data and call `coder-tool.py`,
3. `coder-tool.py`, to color code with the help of `pygments`.

`coder.sty` mainly declares the `\CDRCode` command and the `CDRBlock` environment. The former allows to insert code chunks as running text whereas the latter allows to insert code snippets as blocks. Moreover, block code chunks can be exported to files, once declared with `\CDRExport` command. The `\CDRSet` command is used to set various parameters, including display engines declared with either `\CDRCodeEngineNew` or `\CDRBlockEngineNew`.

5.1 Code flow

The normal code flow is

1. from `coder.sty`, \LaTeX parses a code snippet as `\CDRCode` argument of `CDRBlock` environment body, somehow stores it, and calls `CDR:highlight_source`,
2. `coder-util.lua` reads the content of some command, and stores it in a `json` file, together with informations to process this code snippet properly,
3. `coder-tool.py` is asked by `coder-util.lua` to read the `json` file and eventually uses `pygments` to translate the code snippet into dedicated \LaTeX coloring commands. These are stored in a `*.pyg.tex` file named after the md5 digest of the original code chunk, a `*.pyg.sty` \LaTeX style file is recorded as well. On return, `coder-tool.py` gives to `coder-util.lua` some information, to allow the input of both the `*.pyg.sty` and the `*.pyg.tex` file, which are finally executed and the code is displayed with colors. `coder-tool.py` is also partially responsible of code line numbering in conjunction with `coder.sty`.

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

5.2 File exports

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

5.3 Display engine

The display management is partly delegated to other packages. `coder.sty` provides default engines for running code and code blocks, and new engines can be declared with `\CDRCodeEngineNew` and `\CDRBlockEngineNew`.

5.4 L^AT_EX user interface

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

5.5 Properties and inheritance

Properties cover various informations, from the language of the code, to the color and font. They are uniquely identified by a path component, the *tag*, which is used for inheritance. All tags starting with two leading underscore characters are reserved by the package. Other tags are at the user disposal.

Each processed code chunk has a list of associate tags. Most tag inherits from default ones.

6 Options

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

6.1 fancyvrb

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

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

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

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

6.2 pygments options

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

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

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

6.3 LaTeX

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

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

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

- **Single** It may contain tag related information and number as well. When the block consists of only one line.
- **First** When the block consists of more than one line. If the tag information is required or new, display only the tag. Display the number if required, otherwise.
- **Second** If the first line did not, display the line number, but only when required.
- **Black** for numbered lines,
- **White** for unnumbered lines,

File I

coder-util.lua implementation

1 Usage

This lua library is loaded by `coder.sty` with the instruction `CDR=require(coder-util)`. In the sequel, the syntax to call class methods and instance methods are presented with either a `CDR.` or a `CDR:` prefix. This is what is used in the library for convenience. Of course either a `self.` or a `self:` prefix would be possible.

2 Declarations

```

1 %<*lua>
2 local lfs    = _ENV.lfs
3 local tex    = _ENV.tex
4 local token  = _ENV.token
5 local md5    = _ENV.md5
6 local kpse   = _ENV.kpse
7 local rep    = string.rep
8 local lpeg   = require("lpeg")
9 local P, Cg, Cp, V = lpeg.P, lpeg.Cg, lpeg.Cp, lpeg.V
10 local json  = require('lualibs-util-jsn')
```

3 General purpose material

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

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


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

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

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

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

set_python_path CDR:set_python_path(<path var>)

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


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

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

Execute <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 $\langle jobname \rangle$.pygd/.

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

json_p The path of the JSON file used to communicate with coder-tool.py, in general $\langle jobname \rangle$.pygd/ $\langle jobname \rangle$

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

```

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

```

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

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

```

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

```

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

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

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

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

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

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

load_exec_output

CDR:load_exec_output(*lua code chunk*)

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

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

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

?LUA:*?Lua instructions* these *?Lua instructions* are executed asynchronously once the control comes back to T_EX through a call to `\directlua`, which means that they will wait until any previous asynchronous *?TeX instructions* or *?Lua instructions* completes.

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

4 Properties

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

5 Hiligting

5.1 Common

highlight_set CDR:highlight_set(...)

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

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

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

Highlight the currently entered block if <src> is `true`, build the style definitions if <sty> is `true`. Build a configuration table with all data necessary for the processing, save it as a JSON file and launch `coder-tool.py` with the proper arguments. Set the `\l_CDR_pyg_sty_tl` and `\l_CDR_pyg_tex_tl` macros on return, depending on <src> and <sty>.

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

```

```

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

```

```

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

```

5.2 Code

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.

```

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

```

5.3 Code

highlight_code_setup CDR:highlight_code_setup()

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

```

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

```

```

254     fv_opts = {
255         __cls__ = 'FV0pts',
256     }
257 }
258 self.highlight_json_written = false
259 end
260

```

5.4 Block

highlight_block_setup CDR:highlight_block_setup(*<tags_clist var>*)

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

```

261 local function highlight_block_setup(self, tags_clist_var)
262     local tags_clist = assert(token.get_macro(assert(tags_clist_var)))
263     local t = {}
264     for tag in string.gmatch(tags_clist, '([~,]+)') do
265         t[#t+1]=tag
266     end
267     self['.tags_clist'] = tags_clist
268     self['.block_tags'] = t
269     self['.lines'] = {}
270     self['.arguments'] = {
271         __cls__ = 'Arguments',
272         cache = false,
273         debug = false,
274         source = nil,
275         pygopts = {
276             __cls__ = 'PygOpts',
277             lang = 'tex',
278             style = 'default',
279         },
280         texopts = {
281             __cls__ = 'TeXOpts',
282             tags = tags_clist,
283             is_inline = false,
284             pyg_sty_p = '',
285         },
286         fv_opts = {
287             __cls__ = 'FV0pts',
288         }
289     }
290     self.highlight_json_written = false
291 end
292

```

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

Store the content of the given named variable.

```

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

```

```

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

```

highlight_advance CDR:highlight_advance(*<count>*)
<count> is the number of line highlighted.

```

305 local function highlight_advance(self, count)
306 end

```

6 Exportation

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

export_file CDR:export_file(*<file name var>*)
This is called at export time. *<file name var>* is the name of an str variable containing the file name.

```

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

```

export_file_info CDR:export_file_info(*<key>*, *<value name var>*)
This is called at export time. *<value name var>* is the name of an str variable containing the value.

```

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

```

export_complete CDR:export_complete()
This is called at export time.


```

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

```

7 Caching

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

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

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

```

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

```

```

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

```

`_DESCRIPTION` Short text description of the module.

```

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

```

8 Return the module

```

372 return {
      Known fields are

373   _DESCRIPTION      = _DESCRIPTION,

      _VERSION to store <version string>,

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

      date to store <date string>,

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

      Various paths ,

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

      is_truthy

```

```

379  is_truthy          = is_truthy,

    escape

380  escape              = escape,

    make_directory

381  make_directory     = make_directory,

    load_exec

382  load_exec          = load_exec,

383  load_exec_output   = load_exec_output,

    record_line

384  record_line        = record_line,

    highlight common

385  highlight_set       = highlight_set,
386  highlight_set_var   = highlight_set_var,
387  highlight_source    = highlight_source,
388  highlight_advance    = highlight_advance,
389  highlight_complete   = highlight_complete,

    highlight code

390  highlight_code_setup = highlight_code_setup,

    highlight_block_setup

391  highlight_block_setup = highlight_block_setup,

    cache_clean_all

392  cache_clean_all     = cache_clean_all,

    cache_record

393  cache_record        = cache_record,

    cache_clean_unused

394  cache_clean_unused = cache_clean_unused,

Internals

```

```

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

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

400     already              = false,

    Other

401     json_p               = json_p,

402 }

403 %</lua>

```

File II

coder-tool.py implementation

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

```

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

```

1 Usage

Run: coder-tool.py -h.

2 Header and global declarations

```

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

```

3 Options classes

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

```
20 class BaseOpts(object):
21     @staticmethod
22     def ensure_bool(x):
23         if x == True or x == False: return x
24         x = x[0:1]
25         return x == 'T' or x == 't'
26
27     def __init__(self, d={}):
28         for k, v in d.items():
29             if type(v) == str:
30                 if v.lower() == 'true':
31                     setattr(self, k, True)
32                 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     line_template = r'''\CDR@Line{<placeholder:type>}{<placeholder:number>}{<placeholder:line>}'
46     def __init__(self, *args, **kwargs):
47         super().__init__(*args, **kwargs)
48         self.inline_p = self.ensure_bool(self.is_inline)
49         self.pyg_sty_p = Path(self.pyg_sty_p or '')
```

3.2 PygOptsclass

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

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

```

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

```

3.3 FVclass

```

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

```

3.4 Argumentsclass

```

100 class Arguments(BaseOpts):
101     cache = False

```

```

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

```

4 Controller main class

```

110 class Controller:

```

4.1 Static methods

object_hook Helper for json parsing.

```

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

```

lua_command `self.lua_command(<asynchronous lua command>)`
lua_command_now `self.lua_command_now(<synchronous lua command>)`
lua_debug Wraps the given command between markers. It will be in the output of the `coder-tool.py`, further captured by `coder-util.lua` and either forwarded to \TeX or executed synchronously.

```

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

```

lua_text_escape `self.lua_text_escape(<text>)`
Wraps the given command between `[=...=[` and `]...=]` with as many equal signs as necessary to ensure a correct lua syntax.

```

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

```

4.2 Computed properties

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

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

```

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

```

self.parser The correctly set up argparse instance.

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

```

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

```



```

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

```

4.3 Methods

4.3.1 __init__

__init__ Constructor. Reads the command line arguments.

```

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

```

```

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

```

4.3.2 create_style

self.create_style self.create_style()

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

```

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

```

```

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

```

4.3.3 pygmentize

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

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

```

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

```

```

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

```

4.3.4 create_pygmented

```
self.create_pygmented
```

```
self.create_pygmented()
```

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

```

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

```

4.4 Main entry

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

File III

coder.sty implementation

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

1 Installation test

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

2 Messages

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

3 Constants

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

5 Variables

5.1 Internal scratch variables

These local variables are used in a very limited scope.

`\l_CDR_bool` Local scratch variable.

```

24 \bool_new:N \l_CDR_bool

```

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

`\l_CDR_tl` Local scratch variable.

```

25 \tl_new:N \l_CDR_tl

```

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

`\l_CDR_str` Local scratch variable.

```

26 \str_new:N \l_CDR_str

```

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

`\l_CDR_seq` Local scratch variable.

```

27 \seq_new:N \l_CDR_seq

```

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

`\l_CDR_prop` Local scratch variable.

```

28 \prop_new:N \l_CDR_prop

```

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

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

```

29 \clist_new:N \l_CDR_clist

```

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

5.2 Files

`\l_CDR_ior` Input file identifier

30 `\ior_new:N \l_CDR_ior`

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

`\l_CDR_iow` Output file identifier

31 `\iow_new:N \l_CDR_iow`

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

5.3 Global variables

Line number counter for the source code chunks.

`\g_CDR_source_int` Chunk number counter.

32 `\int_new:N \g_CDR_source_int`

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

`\g_CDR_source_prop` Global source property list.

33 `\prop_new:N \g_CDR_source_prop`

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

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

34 `\tl_new:N \g_CDR_chunks_tl`

35 `\tl_new:N \l_CDR_chunks_tl`

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

`\g_CDR_vars` Tree storage for global variables.

36 `\prop_new:N \g_CDR_vars`

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

`\g_CDR_hook_tl` Hook general purpose.

37 `\tl_new:N \g_CDR_hook_tl`

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

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

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

5.4 Local variables

`\l_CDR_keyval_tl` keyval storage.

38 `\tl_new:N \l_CDR_keyval_tl`

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

`\l_CDR_options_tl` options storage.

39 `\tl_new:N \l_CDR_options_tl`

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

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

40 `\tl_new:N \l_CDR_recorded_tl`

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

`\g_CDR_int` Global integer to store linenos locally in time.

41 `\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.

42 `\tl_new:N \l_CDR_line_tl`

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

`\l_CDR_linenos_tl` Token list for linenos display.

43 `\tl_new:N \l_CDR_linenos_tl`

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

`\l_CDR_name_tl` Token list for chunk name display.

44 `\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.

45 `\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>.

```

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

\g_CDR@int.default Generic and named line number counter.

\g_CDR@int.<tag name> 50 \CDR_int_new:cn { default } { 1 }

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

\CDR_int_if_exist:p:c ★ \CDR_int_if_exist: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.

```

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

\CDR_int_compare:p:cNn ★ \CDR_int_compare:cNnTF {<tag name>} <operator> {<intexpr₂>} {<true code>} {<false code>}

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

```

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

\CDR_int_set:cn \CDR_int_set:cn {<tag name>} {<value>}

\CDR_int_gset:cn Set the integer named after <tag name> to the <value>. \CDR_int_gset:cn makes a global change.

```

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

<code>\CDR_int_add:cn</code> <code>\CDR_int_gadd:cn</code>	<code>\CDR_int_add:cn {<tag name>} {<value>}</code> Add the <code><value></code> to the integer named after <code><tag name></code> . <code>\CDR_int_gadd:cn</code> makes a global change.
---	---

```

71 \cs_new:Npn \CDR_int_add:cn #1 #2 {
72   \int_add:cn { g_CDR@int.#1 } { #2 }
73 }
74 \cs_new:Npn \CDR_int_gadd:cn #1 #2 {
75   \int_gadd:cn { g_CDR@int.#1 } { #2 }
76 }

```

<code>\CDR_int_sub:cn</code> <code>\CDR_int_gsub:cn</code>	<code>\CDR_int_sub:cn {<tag name>} {<value>}</code> Subtract the <code><value></code> from the integer named after <code><tag name></code> . <code>\CDR_int_gsub:cn</code> makes a global change.
---	--

```

77 \cs_new:Npn \CDR_int_sub:cn #1 #2 {
78   \int_sub:cn { g_CDR@int.#1 } { #2 }
79 }
80 \cs_new:Npn \CDR_int_gsub:cn #1 #2 {
81   \int_gsub:cn { g_CDR@int.#1 } { #2 }
82 }

```

<code>\CDR_int_use:c</code> ★	<code>\CDR_int_use:n {<tag name>}</code> Use the integer named after <code><tag name></code> .
-------------------------------	---

```

83 \cs_new:Npn \CDR_int_use:c #1 {
84   \int_use:c { g_CDR@int.#1 }
85 }

```

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>\CDR_tag_get_path:c</code> ★	<code>\CDR_tag_get_path:cc {<tag name>} {<relative key path>}</code> <code>\CDR_tag_get_path:c {<relative key path>}</code> Internal: return a unique key based on the arguments. Used to store and retrieve values. In the second version, the <code><tag name></code> is not provided and set to <code>__local</code> .
---	---

```

86 \cs_new:Npn \CDR_tag_get_path:cc #1 #2 {
87   \c_CDR_tag_get @ #1 / #2
88 }
89 \cs_new:Npn \CDR_tag_get_path:c {

```

```

90 \CDR_tag_get_path:cc { __local }
91 }

```

6.2 Set

<pre> \CDR_tag_set:ccn \CDR_tag_set:ccV </pre>	<pre> \CDR_tag_set:ccn {<tag name>} {<relative key path>} {<value>} </pre> <p>Store <i><value></i>, which is further retrieved with the instruction <code>\CDR_tag_get:cc {<tag name>} {<relative key path>}</code>. Only <i><tag name></i> and <i><relative key path></i> containing no @ character are supported. All the affectations are made at the current T_EX group level. <i>Nota Bene:</i> <code>\cs_generate_variant:Nn</code> is buggy when there is a 'c' argument.</p>
--	--

```

92 \cs_new_protected:Npn \CDR_tag_set:ccn #1 #2 #3 {
93   \cs_set:cpn { \CDR_tag_get_path:cc { #1 } { #2 } } { \exp_not:n { #3 } }
94 }
95 \cs_new_protected:Npn \CDR_tag_set:ccV #1 #2 #3 {
96   \exp_args:NnnV
97   \CDR_tag_set:ccn { #1 } { #2 } #3
98 }

```

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

```

99 \tl_set:Nn \l_CDR_tl { /([~/]*)/(.*)$ } \use_none:n { $ }
100 \tl_put_left:NV \l_CDR_tl \c_CDR_tag
101 \tl_put_left:Nn \l_CDR_tl { ^ }
102 \exp_args:NNV
103 \regex_const:Nn \c_CDR_tag_regex \l_CDR_tl

```

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

<pre> \CDR_tag_set:n </pre>	<pre> \CDR_tag_set:n {<value>} </pre>
-----------------------------	---

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

```

104 \cs_new_protected:Npn \CDR_tag_set:n {
105   \exp_args:NnV
106   \regex_extract_once:NnNTF \c_CDR_tag_regex
107   \l_keys_path_str \l_CDR_seq {
108     \CDR_tag_set:ccn
109     { \seq_item:Nn \l_CDR_seq 2 }
110     { \seq_item:Nn \l_CDR_seq 3 }
111   } {
112     \PackageWarning
113     { coder }
114     { Unexpected-key~path~'\l_keys_path_str' }
115     \use_none:n
116   }
117 }

```

`\CDR_tag_set:` `\CDR_tag_set:`

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

```
118 \cs_new_protected:Npn \CDR_tag_set: {
119   \exp_args:NV
120   \CDR_tag_set:n \l_keys_value_tl
121 }
```

`\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.

```
122 \cs_new:Npn \CDR_tag_set:cn #1 {
123   \exp_args:NnV
124   \regex_extract_once:NnNTF \c_CDR_tag_regex
125     \l_keys_path_str \l_CDR_seq {
126     \CDR_tag_set:cn
127     { \seq_item:Nn \l_CDR_seq 2 }
128     { #1 }
129   } {
130     \PackageWarning
131     { coder }
132     { Unexpected~key~path~'\l_keys_path_str' }
133     \use_none:n
134   }
135 }
```

`\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.

```
136 \regex_const:Nn \c_CDR_root_regex { ^(.*)/.*$ } \use_none:n { $ }
137 \cs_new:Npn \CDR_tag_choices: {
138   \exp_args:NVV
139   \str_if_eq:nnT \l_keys_key_tl \l_keys_choice_tl {
140     \exp_args:NnV
141     \regex_extract_once:NnNT \c_CDR_root_regex
142       \l_keys_path_str \l_CDR_seq {
143       \str_set:Nx \l_keys_path_str {
144         \seq_item:Nn \l_CDR_seq 2
145       }
146     }
147   }
148 }
```

`\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.

```
149 \cs_new_protected:Npn \CDR_tag_choices_set: {  
150   \CDR_tag_choices:  
151   \exp_args:NV  
152   \CDR_tag_set:n \l_keys_choice_tl  
153 }
```

`\CDR_if_tag_truthy_p:cc` `\CDR_if_tag_truthy:ccTF {<tag name>} {<relative key path>} {<true code>} {<false code>}`
`\CDR_if_tag_truthy:ccTF` `code}`
`\CDR_if_tag_truthy_p:c` `\CDR_if_tag_truthy:cTF {<relative key path>} {<true code>} {<false code>}`
`\CDR_if_tag_truthy:cTF` `Execute <true code> when the property for <tag name> and <relative key path> is a`

truthy value, `<false code>` otherwise. A truthy value is a text which is not “false” in a case insensitive comparison. In the second version, the `<tag name>` is not provided and set to `__local`.

```
154 \prg_new_conditional:Nnn \CDR_if_tag_truthy:cc { p, T, F, TF } {  
155   \exp_args:Ne  
156   \str_compare:nNnTF {  
157     \exp_args:Ne \str_lowercase:n { \CDR_tag_get:cc { #1 } { #2 } }  
158   } = { true } {  
159     \prg_return_true:  
160   } {  
161     \prg_return_false:  
162   }  
163 }  
164 \prg_new_conditional:Nnn \CDR_if_tag_truthy:c { p, T, F, TF } {  
165   \exp_args:Ne  
166   \str_compare:nNnTF {  
167     \exp_args:Ne \str_lowercase:n { \CDR_tag_get:c { #1 } }  
168   } = { true } {  
169     \prg_return_true:  
170   } {  
171     \prg_return_false:  
172   }  
173 }
```

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

`\CDR_if_truthy:nTF` `Execute <true code> when <token list> is a truthy value, <false code> otherwise. A`
truthy value is a text which leading character, if any, is none of “fFnN”.

```
174 \prg_new_conditional:Nnn \CDR_if_truthy:n { p, T, F, TF } {  
175   \exp_args:Ne  
176   \str_compare:nNnTF { \exp_args:Ne \str_lowercase:n { #1 } } = { true } {  
177     \prg_return_true:  
178   } {  
179     \prg_return_false:  
180   }  
181 }
```

`\CDR_tag_boolean_set:n` `\CDR_tag_boolean_set:n {<choice>}`
 Calls `\CDR_tag_set:n` with `true` if the argument is truthy, `false` otherwise.

```

182 \cs_new_protected:Npn \CDR_tag_boolean_set:n #1 {
183   \CDR_if_truthy:nTF { #1 } {
184     \CDR_tag_set:n { true }
185   } {
186     \CDR_tag_set:n { false }
187   }
188 }
189 \cs_generate_variant:Nn \CDR_tag_boolean_set:n { x }

```

6.3 Retrieving tag properties

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

1. `\c_CDR_tag_get/<tag name>` for the provided `<tag name>`,
2. `\c_CDR_tag_get/default.code`
3. `\c_CDR_tag_get/default`
4. `\c_CDR_tag_get/__pygments`
5. `\c_CDR_tag_get/__fancyvrb`
6. `\c_CDR_tag_get/__fancyvrb.all` when no using `pygments`

For text block code chunks, this module inherits from

1. `\c_CDR_tag_get/<name1>`, ..., `\c_CDR_tag_get/<namen>` for each tag name of the ordered tags list
2. `\c_CDR_tag_get/default.block`
3. `\c_CDR_tag_get/default`
4. `\c_CDR_tag_get/__pygments`
5. `\c_CDR_tag_get/__pygments.block`
6. `\c_CDR_tag_get/__fancyvrb`
7. `\c_CDR_tag_get/__fancyvrb.block`
8. `\c_CDR_tag_get/__fancyvrb.all` when no using `pygments`

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

If the `<relative key path>` is known within `<tag name>`, the `<true code>` is executed, otherwise, the `<false code>` is executed. No inheritance.

```

190 \prg_new_conditional:Nnn \CDR_tag_if_exist_here:cc { T, F, TF } {
191   \cs_if_exist:cTF { \CDR_tag_get_path:cc { #1 } { #2 } } {
192     \prg_return_true:
193   } {
194     \prg_return_false:
195   }
196 }

```

```

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

```

If the *<relative key path>* is known within *<tag name>*, the *<true code>* is executed, otherwise, the *<false code>* is executed if none of the parents has the *<relative key path>* on its own. In the second version, the *<tag name>* is not provided and set to `__local`.

```

197 \prg_new_conditional:Nnn \CDR_tag_if_exist:cc { T, F, TF } {
198   \cs_if_exist:cTF { \CDR_tag_get_path:cc { #1 } { #2 } } {
199     \prg_return_true:
200   } {
201     \seq_if_exist:cTF { \CDR_tag_parent_seq:c { #1 } } {
202       \seq_map_tokens:cn
203         { \CDR_tag_parent_seq:c { #1 } }
204         { \CDR_tag_if_exist_f:cn { #2 } }
205     } {
206       \prg_return_false:
207     }
208   }
209 }
210 \prg_new_conditional:Nnn \CDR_tag_if_exist:c { T, F, TF } {
211   \cs_if_exist:cTF { \CDR_tag_get_path:c { #1 } } {
212     \prg_return_true:
213   } {
214     \seq_if_exist:cTF { \CDR_tag_parent_seq:c { __local } } {
215       \seq_map_tokens:cn
216         { \CDR_tag_parent_seq:c { __local } }
217         { \CDR_tag_if_exist_f:cn { #1 } }
218     } {
219       \prg_return_false:
220     }
221   }
222 }
223 \cs_new:Npn \CDR_tag_if_exist_f:cn #1 #2 {
224   \quark_if_no_value:nTF { #2 } {
225     \seq_map_break:n {
226       \prg_return_false:
227     }
228   } {
229     \CDR_tag_if_exist:ccT { #2 } { #1 } {
230       \seq_map_break:n {
231         \prg_return_true:
232       }
233     }

```

```

234 }
235 }

```

```

\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`.

```

236 \cs_new:Npn \CDR_tag_get:cc #1 #2 {
237   \CDR_tag_if_exist_here:ccTF { #1 } { #2 } {
238     \use:c { \CDR_tag_get_path:cc { #1 } { #2 } }
239   } {
240     \seq_if_exist:cT { \CDR_tag_parent_seq:c { #1 } } {
241       \seq_map_tokens:cn
242         { \CDR_tag_parent_seq:c { #1 } }
243         { \CDR_tag_get_f:cn { #2 } }
244     }
245   }
246 }
247 \cs_new:Npn \CDR_tag_get_f:cn #1 #2 {
248   \quark_if_no_value:nF { #2 } {
249     \CDR_tag_if_exist_here:ccT { #2 } { #1 } {
250       \seq_map_break:n {
251         \use:c { \CDR_tag_get_path:cc { #2 } { #1 } }
252       }
253     }
254   }
255 }
256 \cs_new:Npn \CDR_tag_get:c {
257   \CDR_tag_get:cc { __local }
258 }

```

```

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

```

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

```

259 \cs_new_protected:Npn \CDR_tag_get:ccN #1 #2 #3 {
260   \tl_set:Nf #3 { \CDR_tag_get:cc { #1 } { #2 } }
261 }
262 \cs_new_protected:Npn \CDR_tag_get:cN {
263   \CDR_tag_get:ccN { __local }
264 }

```

```

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

```

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


```

265 \prg_new_protected_conditional:Nnn \CDR_tag_get:ccN { T, F, TF } {
266   \CDR_tag_if_exist:ccTF { #1 } { #2 } {
267     \CDR_tag_get:ccN { #1 } { #2 } #3
268     \prg_return_true:
269   } {
270     \prg_return_false:
271   }
272 }
273 \prg_new_protected_conditional:Nnn \CDR_tag_get:cN { T, F, TF } {
274   \CDR_tag_if_exist:cTF { #1 } {
275     \CDR_tag_get:cN { #1 } #2
276     \prg_return_true:
277   } {
278     \prg_return_false:
279   }
280 }

```

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.

```

281 \cs_new:Npn \CDR_tag_parent_seq:c #1 {
282   g_CDR:parent.tag @ #1 _seq
283 }

```

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

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

```

284 \cs_new:Npn \CDR_tag_inherit:cn #1 #2 {
285   \seq_set_from_clist:cn { \CDR_tag_parent_seq:c { #1 } } { #2 }
286   \seq_remove_duplicates:c \l_CDR_tl
287   \seq_remove_all:cn \l_CDR_tl {}
288   \seq_put_right:cn \l_CDR_tl { \q_no_value }
289 }
290 \cs_new:Npn \CDR_tag_inherit:cf {
291   \exp_args:Nnf \CDR_tag_inherit:cn
292 }
293 \cs_new:Npn \CDR_tag_inherit:cV {
294   \exp_args:NnV \CDR_tag_inherit:cn
295 }

```

7 Cache management

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

```

296 \AddToHook { begindocument/before } {
297   \IfFileExists {./\jobname.aux} {} {
298     \lua_now:n {CDR:cache_clean_all()}
299   }
300 }

```

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

```

301 \AddToHook { enddocument/end } {
302   \lua_now:n {CDR:cache_clean_unused()}
303 }

```

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>.

```

304 \cs_new:Npn \CDR_clist_map_inline:Nnn #1 #2 {
305   \clist_if_empty:NTF #1 {
306     #2
307     \use_none:n
308   } {
309     \clist_map_inline:Nn #1
310   }
311 }

```

\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.

```

312 \prg_new_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
313   \PackageError
314     { coder }
315     { Conditional~not~available }
316 }

```

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

```

317 \cs_new:Npn \CDR_process_record: {}

```

9 l3keys modules for code chunks

All these modules are initialized at the beginning of the document using the `__initialize` meta key.

9.1 Utilities

`\CDR_tag_keys_define:nn` `\CDR_tag_keys_define:nn {< module base >} {< keyval list >}`

The `<module>` is uniquely based on `<module base>` before forwarding to `\keys_define:nn`.

```

318 \cs_generate_variant:Nn \keys_define:nn { Vn, xn }
319 \cs_new:Npn \CDR_tag_keys_define:nn #1 {
320   \keys_define:xn { \c_CDR_tag / \exp_not:n { #1 } }
321 }
322 \cs_generate_variant:Nn \CDR_tag_keys_define:nn { nx }
```

`\CDR_tag_keys_set:nn` `\CDR_tag_keys_set:nn {<module base>} {<keyval list>}`

The `<module>` is uniquely based on `<module base>` before forwarding to `\keys_set:nn`.

```

323 \cs_new:Npn \CDR_tag_keys_set:nn #1 {
324   \exp_args:Nx
325   \keys_set:nn { \c_CDR_tag / \exp_not:n { #1 } }
326 }
327 \cs_generate_variant:Nn \CDR_tag_keys_set:nn { nV }
```

9.1.1 Handling unknown tags

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

`\CDR_keys_set_known:nnN` `\CDR_keys_set_known:nnN {<module>} {<key[=value] items>} <tl var>`

Wrappers over `\keys_set_known:nnnN` where the `<root>` is also the `<module>`.

```

328 \cs_new:Npn \CDR_keys_set_known:nnN #1 #2 {
329   \keys_set_known:nnnN { #1 } { #2 } { #1 }
330 }
331 \cs_generate_variant:Nn \CDR_keys_set_known:nnN { x, VV }
```

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

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

```

332 \cs_new:Npn \CDR_keys_inherit__:nnn #1 #2 #3 {
333   \keys_define:nn { #1 } { #2 .inherit:n = { #3 } }
334 }
335 \cs_new:Npn \CDR_keys_inherit:nnn #1 #2 #3 {
336   \tl_if_empty:nTF { #1 } {
337     \CDR_keys_inherit__:nnn { } { #2 } { #3 }
338   } {
```

```

339 \clist_set:Nn \l_CDR_clist { #3 }
340 \exp_args:Nnnx
341 \CDR_keys_inherit__:nnn { #1 } { #2 } {
342   #1 / \clist_use:Nn \l_CDR_clist { ,#1/ }
343 }
344 }
345 }
346 \cs_generate_variant:Nn \CDR_keys_inherit:nnn { VnV, Vnn }

```

`\CDR_tag_keys_set_known:nnN` `\CDR_tag_keys_set_known:nnN {<tag name>} {<key[=value] items>} <tl var>`

Wrappers over `\keys_set_known:nnnN` where the module is given by `\c_CDR_tag/<tag name>`. *Implementation detail* the remaining arguments are absorbed by the last macro.

```

347 \cs_generate_variant:Nn \keys_set_known:nnnN { VVV, nVx }
348 \cs_new:Npn \CDR_tag_keys_set_known:nnN #1 {
349   \CDR_keys_set_known:xnN { \c_CDR_tag / \exp_not:n { #1 } }
350 }
351 \cs_generate_variant:Nn \CDR_tag_keys_set_known:nnN { nV }

```

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

```

352 \tl_set:Nn \l_CDR_tl { /([~/*])(?:/(.))*? $ } \use_none:n { $ }
353 \tl_put_left:NV \l_CDR_tl \c_CDR_tag
354 \tl_put_left:Nn \l_CDR_tl { ^ }
355 \exp_args:NNV
356 \regex_const:Nn \c_CDR_provide_regex \l_CDR_tl

```

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

`\CDR_tag_provide_from_clist:n` `\CDR_tag_provide_from_clist:n {<deep comma list>}`
`\CDR_tag_provide_from_keyval:n` `\CDR_tag_provide_from_keyval:n {<key-value list>}`

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

Notice that a tag name should contain no `'/'`.

```

357 \regex_const:Nn \c_CDR_engine_regex { ^[~/]*\sengine\soptions$ } \use_none:n { $ }
358 \cs_new:Npn \CDR_tag_provide_from_clist:n #1 {
359   \exp_args:NNx
360   \regex_extract_once:NnNTF \c_CDR_provide_regex {
361     \c_CDR_Tags / #1
362   } \l_CDR_seq {
363     \tl_set:Nx \l_CDR_tl { \seq_item:Nn \l_CDR_seq 3 }
364     \exp_args:Nx
365     \clist_map_inline:nn {
366       \seq_item:Nn \l_CDR_seq 2
367     } {
368       \exp_args:NV
369       \keys_if_exist:nnF \c_CDR_tag { ##1 } {
370         \CDR_keys_inherit:Vnn \c_CDR_tag { ##1 } {
371           __pygments, __pygments.block,

```

```

372         default.block, default.code, default,
373         __fancyvrb, __fancyvrb.block, __fancyvrb.all
374     }
375     \keys_define:Nn \c_CDR_tag {
376         ##1 .code:n = \CDR_tag_keys_set:nn { ##1 } { #####1 },
377         ##1 .value_required:n = true,
378     }
379 }
380 \exp_args:NxV
381 \keys_if_exist:nnF { \c_CDR_tag / ##1 } \l_CDR_tl {
382     \exp_args:NNV
383     \regex_match:NnT \c_CDR_engine_regex
384     \l_CDR_tl {
385         \CDR_tag_keys_define:nx { ##1 } {
386             \l_CDR_tl .code:n = \exp_not:n { \CDR_tag_set:n { #####1 } },
387             \l_CDR_tl .value_required:n = true,
388         }
389     }
390 }
391 }
392 } {
393     \regex_match:NnT \c_CDR_engine_regex { #1 } {
394         \CDR_tag_keys_define:nn { default } {
395             #1 .code:n = \CDR_tag_set:n { ##1 },
396             #1 .value_required:n = true,
397         }
398     }
399 }
400 }
401 \cs_new:Npn \CDR_tag_provide_from_clist:nn #1 #2 {
402     \CDR_tag_provide_from_clist:n { #1 }
403 }
404 \cs_new:Npn \CDR_tag_provide_from_keyval:n {
405     \keyval_parse:nnn {
406         \CDR_tag_provide_from_clist:n
407     } {
408         \CDR_tag_provide_from_clist:nn
409     }
410 }
411 \cs_generate_variant:Nn \CDR_tag_provide_from_keyval:n { V }

```

9.2 pygments

These are `pygments`'s `LatexFormatter` options, that are not covered by `__fancyvrb`. They are made available at the end user level, but may not be relevant when `pygments` is not used.

9.2.1 Utilities

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

```

412 \sys_get_shell:nnN {which-pygmentize} {} \l_CDR_tl
413 \prg_new_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } { }
414 \tl_if_in:NnTF \l_CDR_tl { pygmentize } {
415   \prg_set_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } {
416     \prg_return_true:
417   }
418 } {
419   \prg_set_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } {
420     \prg_return_false:
421   }
422 }

```

9.2.2 `__pygments` l3keys module

```

423 \CDR_tag_keys_define:nn { __pygments } {

```

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

```

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

```

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

```

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

```

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

```

427   style .code:n = \CDR_tag_set:,
428   style .value_required:n = true,

```

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

```

429   commandprefix .code:n = \CDR_tag_set:,
430   commandprefix .value_required:n = true,

```

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

```

431   mathescape .code:n = \CDR_tag_boolean_set:x { #1 },
432   mathescape .default:n = true,

```

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

```
433 escapeinside .code:n = \CDR_tag_set:,
434 escapeinside .value_required:n = true,
```

- **__initialize** Initializer.

```
435 __initialize .meta:n = {
436   lang = tex,
437   pygments = \CDR_has_pygments:TF { true } { false },
438   style=default,
439   commandprefix=PY,
440   mathescape=false,
441   escapeinside=,
442 },
443 __initialize .value_forbidden:n = true,

444 }
445 \AtBeginDocument{
446   \CDR_tag_keys_set:nn { __pygments } { __initialize }
447 }
```

9.2.3 \c_CDR_tag / __pygments.block l3keys module

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

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

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

- **__initialize** Initializer.

```
451 __initialize .meta:n = {
452   texcomments=false,
453 },
454 __initialize .value_forbidden:n = true,

455 }
456 \AtBeginDocument{
457   \CDR_tag_keys_set:nn { __pygments.block } { __initialize }
458 }
```

9.3 Specific to coder

9.3.1 default l3keys module

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

Keys are:

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

```
460 format .code:n = \CDR_tag_set:,
461 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.

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

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

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

- **post processor**=*(command)* the command for pygments post processor. This is a string where every occurrence of “%%file%%” is replaced by the full path of the *.pyg.tex file to be post processed and then executed as terminal instruction. Initially empty.

```
464 post~processor .code:n = \CDR_tag_set:,
465 post~processor .value_required:n = true,
```

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

```
466 parskip .code:n = \CDR_tag_set:,
467 parskip .value_required:n = true,
```

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

```
468 engine .code:n = \CDR_tag_set:,
469 engine .value_required:n = true,
```

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

```
470 default~engine~options .code:n = \CDR_tag_set:,
471 default~engine~options .value_required:n = true,
```

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

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

```
472 __initialize .meta:n = {
473   format = ,
474   cache = true,
475   debug = false,
476   post~processor = ,
```



```

477     parskip = \the\parskip,
478     engine = default,
479     default~engine~options = ,
480 },
481 __initialize .value_forbidden:n = true,

482 }
483 \AtBeginDocument{
484   \CDR_tag_keys_set:nn { default } { __initialize }
485 }

```

9.3.2 default.code l3keys module

Void for the moment.

```

486 \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.

```

487   __initialize .meta:n = {
488   },
489   __initialize .value_forbidden:n = true,

490 }
491 \AtBeginDocument{
492   \CDR_tag_keys_set:nn { default.code } { __initialize }
493 }

```

9.3.3 default.block l3keys module

```

494 \CDR_tag_keys_define:nn { default.block } {

```

Known keys include:

- **show tags[=true|false]** to enable/disable the display of the code chunks tags. Initially `true`. Set it to `false` when there happens to be only one tag.

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

```

495   tags .code:n = {
496     \clist_set:Nn \l_CDR_clist { #1 }
497     \clist_remove_duplicates:N \l_CDR_clist
498     \exp_args:NV
499     \CDR_tag_set:n \l_CDR_clist
500   },
501   tags .value_required:n = true,

```

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

```

502 tags~format .code:n = \CDR_tag_set:,
503 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).

```

504 numbers~format .code:n = \CDR_tag_set:,
505 numbers~format .value_required:n = true,

```

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

```

506 show~tags .code:n = \CDR_tag_boolean_set:x { #1 },

```

● **only top**=[true|false] to avoid chunk tags repetitions, if on the same page, two consecutive code chunks have the same tag names, the second names are not displayed.

```

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

```

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

```

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

```

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

```

509 blockskip .code:n = \CDR_tag_set:,
510 blockskip .value_required:n = true,

```

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

```

511 __initialize .meta:n = {
512   tags = ,
513   show~tags = true,
514   only~top = true,
515   use~margin = true,
516   numbers~format = {
517     \sffamily
518     \scriptsize
519     \color{gray}
520   },
521   tags~format = {
522     \bfseries
523   },
524   blockskip = \topsep,
525 },
526 __initialize .value_forbidden:n = true,
527 }
528 \AtBeginDocument{
529   \CDR_tag_keys_set:nn { default.block } { __initialize }
530 }

```

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

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

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

```
532   formatcom .code:n = \CDR_tag_set:,
533   formatcom .value_required:n = true,
```

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

```
534   fontfamily .code:n = \CDR_tag_set:,
535   fontfamily .value_required:n = true,
```

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

```
536   fontsize .code:n = \CDR_tag_set:,
537   fontsize .value_required:n = true,
```

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

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

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

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

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

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

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

```
543   showtabs .code:n = \CDR_tag_boolean_set:x { #1 },
```

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

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

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

```
545 tabsize .code:n = \CDR_tag_set:,
546 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.

```
547 defineactive .code:n = \CDR_tag_set:,
548 defineactive .value_required:n = true,
```

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

```
549 rellabel .code:n = \CDR_tag_set:,
550 rellabel .value_required:n = true,
```

✅ **__initialize** Initialization.

```
551 __initialize .meta:n = {
552   formatcom = ,
553   fontfamily = tt,
554   fontsize = auto,
555   fontseries = auto,
556   fontshape = auto,
557   showspaces = false,
558   showtabs = false,
559   obeytabs = false,
560   tabsize = 2,
561   defineactive = ,
562   rellabel = ,
563 },
564 __initialize .value_forbidden:n = true,

565 }
566 \AtBeginDocument{
567   \CDR_tag_keys_set:nn { __fancyvrb } { __initialize }
568 }
```

9.4.2 __fancyvrb.block l3keys module

Block specific options, except numbering.

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

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

```

571 frame .choices:nn =
572   { none, leftline, topline, bottomline, lines, single }
573   { \CDR_tag_choices_set: },

```

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

```

574 framerule .code:n = \CDR_tag_set:,
575 framerule .value_required:n = true,

```

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

```

576 framesep .code:n = \CDR_tag_set:,
577 framesep .value_required:n = true,

```

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

```

578 rulecolor .code:n = \CDR_tag_set:,
579 rulecolor .value_required:n = true,

```

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

```

580 fillcolor .code:n = \CDR_tag_set:,
581 fillcolor .value_required:n = true,

```

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

```

582 label .code:n = \CDR_tag_set:,
583 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.

```

584 labelposition .choices:nn =
585   { none, topline, bottomline, all }
586   { \CDR_tag_choices_set: },

```

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

```

587 baselinestretch .code:n = \CDR_tag_set:,
588 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.

```
589 xleftmargin .code:n = \CDR_tag_set:,
590 xleftmargin .value_required:n = true,
```

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

```
591 xrightmargin .code:n = \CDR_tag_set:,
592 xrightmargin .value_required:n = true,
```

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

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

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

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

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

✅ **__initialize** Initialization.

```
597 __initialize .meta:n = {
598   frame = none,
599   label = ,
600   labelposition = none,% auto?
601   baselinestretch = auto,
602   resetmargins = true,
603   xleftmargin = 0pt,
604   xrightmargin = 0pt,
605   hfuzz = 2pt,
606   samepage = false,
607 },
608 __initialize .value_forbidden:n = true,

609 }
610 \AtBeginDocument{
611   \CDR_tag_keys_set:nn { __fancyvrb.block } { __initialize }
612 }
```

9.4.3 `__fancyvrb.number l3keys` module

Block line numbering.

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

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

```
614 commentchar .code:n = \CDR_tag_set:,
615 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.

```
616 gobble .choices:nn = {
617   0,1,2,3,4,5,6,7,8,9
618 } {
619   \CDR_tag_choices_set:
620 },
```

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

```
621 numbers .choices:nn =
622   { none, left, right }
623   { \CDR_tag_choices_set: },
```

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

```
624 numbersep .code:n = \CDR_tag_set:,
625 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.

```
626 firstnumber .code:n = {
627   \regex_match:NnTF \c_CDR_integer_regex { #1 } {
628     \CDR_tag_set:
629   } {
630     \str_case:nnF { #1 } {
631       { auto } { \CDR_tag_set: }
632       { last } { \CDR_tag_set: }
633     } {
634       \PackageWarning
635         { CDR }
636         { Value-‘#1’~not~in~auto,~last. }
637     }
638   }
639 },
640 firstnumber .value_required:n = true,
```

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

```
641 stepnumber .code:n = \CDR_tag_set:,
642 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.

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

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

```
644 firstline .code:n = \CDR_tag_set:,
645 firstline .value_required:n = true,
```

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

```
646 lastline .code:n = \CDR_tag_set:,
647 lastline .value_required:n = true,
```

✓ **__initialize** Initialization.

```
648 __initialize .meta:n = {
649   commentchar = ,
650   gobble = 0,
651   numbers = left,
652   numbersep = 1ex,
653   firstnumber = auto,
654   stepnumber = 1,
655   numberblanklines = true,
656   firstline = ,
657   lastline = ,
658 },
659 __initialize .value_forbidden:n = true,

660 }
661 \AtBeginDocument{
662   \CDR_tag_keys_set:nn { __fancyvrb.number } { __initialize }
663 }
```

9.4.4 **__fancyvrb.all** **l3keys** module

Options available when **pygments** is not used.

```
664 \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.


```

665 commandchars .code:n = \CDR_tag_set:,
666 commandchars .value_required:n = true,

```

🔴 **codes**= $\langle macro \rangle$ to specify catcode changes. For instance, this allows us to include formatted mathematics in verbatim text. Initially empty. Ignored in **pygments** mode.

```

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

```

✅ **__initialize** Initialization.

```

669 __initialize .meta:n = {
670   commandchars = ,
671   codes = ,
672 },
673 __initialize .value_forbidden:n = true,

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

```

10 \CDRSet

\CDRSet	\CDRSet { $\langle key [=value] list \rangle$ } \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 $\langle key \rangle [= \langle value \rangle]$ items defined by the **CDR@Set l3keys** module.

10.1 CDR@Set l3keys module

```

678 \keys_define:nn { CDR@Set } {

```

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

```

679 only~description .choices:nn = { false, true, {} } {
680   \int_compare:nNnTF \l_keys_choice_int = 1 {
681     \prg_set_conditional:Nnn \CDR_if_only_description: { p, T, F, TF } { \prg_return_true: }
682   } {
683     \prg_set_conditional:Nnn \CDR_if_only_description: { p, T, F, TF } { \prg_return_false: }
684   }
685 },
686 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**.

```

687 python~path .code:n = {
688   \str_set:Nn \l_CDR_str { #1 }
689   \lua_now:n { CDR:set_python_path('l_CDR_str') }
690 },
691 }

```

10.2 Branching

```

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

```

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

10.3 Implementation

```

\CDR_check_unknown:N \CDR_check_unknown:N {<tl variable>}

```

In normal situation, the argument is expected to be empty. When the argument is not empty, send a package warning for each key.

```

692 \exp_args_generate:n { xV, nnV }
693 \cs_new:Npn \CDR_check_unknown:N #1 {
694   \tl_if_empty:NF #1 {
695     \cs_set:Npn \CDR_check_unknown:n ##1 {
696       \PackageWarning
697         { coder }
698         { Unknow~key~'##1' }
699     }
700     \cs_set:Npn \CDR_check_unknown:nn ##1 ##2 {
701       \CDR_check_unknown:n { ##1 }
702     }
703     \exp_args:NnnV
704     \keyval_parse:nnn {
705       \CDR_check_unknown:n
706     } {
707       \CDR_check_unknown:nn
708     } #1
709   }
710 }

711 \NewDocumentCommand \CDRSet { m } {
712   \CDR_keys_set_known:nnN { CDR@Set } { #1 } \l_CDR_keyval_tl
713   \clist_map_inline:nn {
714     __pygments, __pygments.block,
715     default.block, default.code, default,
716     __fancyvrb, __fancyvrb.block, __fancyvrb.all
717   } {
718     \CDR_tag_keys_set_known:nVN { ##1 } \l_CDR_keyval_tl \l_CDR_keyval_tl
719   }
720   \CDR_keys_set_known:VVN \c_CDR_Tags \l_CDR_keyval_tl \l_CDR_keyval_tl

```

```

721 \CDR_tag_provide_from_keyval:V \l_CDR_keyval_tl
722 \CDR_keys_set_known:VVN \c_CDR_Tags \l_CDR_keyval_tl \l_CDR_keyval_tl
723 \CDR_tag_keys_set:nV { default } \l_CDR_keyval_tl
724 }

```

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.

```

725 \cs_new:Npn \CDR_export_get_path:cc #1 #2 {
726   CDR @ export @ get @ #1 / #2
727 }

```

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

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

```

728 \cs_new_protected:Npn \CDR_export_set:ccn #1 #2 #3 {
729   \cs_set:cpn { \CDR_export_get_path:cc { #1 } { #2 } } { \exp_not:n { #3 } }
730 }
731 \cs_new_protected:Npn \CDR_export_set:Vcn #1 {
732   \exp_args:NV
733   \CDR_export_set:ccn { #1 }
734 }
735 \cs_new_protected:Npn \CDR_export_set:VcV #1 #2 #3 {
736   \exp_args:NVnV
737   \CDR_export_set:ccn #1 { #2 } #3
738 }

```

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

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

```

739 \prg_new_conditional:Nnn \CDR_export_if_exist:cc { p, T, F, TF } {
740   \cs_if_exist:cTF { \CDR_export_get_path:cc { #1 } { #2 } } {
741     \prg_return_true:
742   } {
743     \prg_return_false:
744   }
745 }

```

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

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

```
746 \cs_new:Npn \CDR_export_get:cc #1 #2 {
747   \CDR_export_if_exist:ccT { #1 } { #2 } {
748     \use:c { \CDR_export_get_path:cc { #1 } { #2 } }
749   }
750 }
```

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

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

```
751 \prg_new_protected_conditional:Nnn \CDR_export_get:ccN { T, F, TF } {
752   \CDR_export_if_exist:ccTF { #1 } { #2 } {
753     \tl_set:Nx #3 { \CDR_export_get:cc { #1 } { #2 } }
754     \prg_return_true:
755   } {
756     \prg_return_false:
757   }
758 }
```

11.2 Storage

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

```
759 \prop_new:N \g_CDR_export_prop
```

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

`\l_CDR_file_tl` Store the file name used for exportation, used as key in the above property list.

```
760 \tl_new:N \l_CDR_file_tl
```

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

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

```
761 \clist_new:N \g_CDR_tags_clist
762 \clist_new:N \g_CDR_all_tags_clist
```

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

`\l_CDR_export_prop` Used by `CDR@Export l3keys` module to temporarily store properties. *Nota Bene*: nothing
similar with `\g_CDR_export_prop` except the name.

```
763 \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.

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

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

```
765   file .tl_set:N = \l_CDR_file_tl,
766   file .value_required:n = true,
```

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

```
767   tags .code:n = {
768     \clist_set:Nn \l_CDR_clist { #1 }
769     \clist_remove_duplicates:N \l_CDR_clist
770     \prop_put:NVV \l_CDR_export_prop \l_keys_key_str \l_CDR_clist
771   },
772   tags .value_required:n = true,
```

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

```
773   lang .code:n = {
774     \prop_put:NVn \l_CDR_export_prop \l_keys_key_str { #1 }
775   },
776   lang .value_required:n = true,
```

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

```
777   preamble .code:n = {
778     \prop_put:NVn \l_CDR_export_prop \l_keys_key_str { #1 }
779   },
780   preamble .value_required:n = true,
```

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

```
781   postamble .code:n = {
782     \prop_put:NVn \l_CDR_export_prop \l_keys_key_str { #1 }
783   },
784   postamble .value_required:n = true,
```

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

```
785   raw .choices:nn = { false, true, {} } {
786     \prop_put:NVx \l_CDR_export_prop \l_keys_key_str {
787       \int_compare:nNnTF
788         \l_keys_choice_int = 1 { false } { true }
789     }
790   },
```

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

```

791 __initialize .meta:n = {
792   __initialize_prop = #1,
793   file=,
794   tags=,
795   lang=tex,
796   preamble=,
797   postamble=,
798   raw=false,
799 },
800 __initialize .default:n = \l_CDR_export_prop,

```

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

```

801 __initialize_prop .code:n = \prop_clear:N #1,
802 __initialize_prop .value_required:n = true,
803 }

```

11.4 Implementation

```

804 \NewDocumentCommand \CDRExport { m } {
805   \keys_set:nn { CDR@Export } { __initialize }
806   \keys_set:nn { CDR@Export } { #1 }
807   \tl_if_empty:NTF \l_CDR_file_tl {
808     \PackageWarning
809       { coder }
810     { Missing-key-‘file’ }
811   } {
812     \CDR_export_set:VcV \l_CDR_file_tl { file } \l_CDR_file_tl
813     \prop_map_inline:Nn \l_CDR_export_prop {
814       \CDR_export_set:Vcn \l_CDR_file_tl { ##1 } { ##2 }
815     }

```

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.

```

816   \prop_get:NnNTF \l_CDR_export_prop { tags } \l_CDR_clist {
817     \tl_if_empty:NTF \l_CDR_clist {
818       \PackageWarning
819         { coder }
820       { Missing-key-‘tags’ }
821     } {
822       \clist_set_eq:NN \g_CDR_tags_clist \l_CDR_clist
823       \clist_put_left:NV \g_CDR_all_tags_clist \l_CDR_clist
824       \clist_remove_duplicates:N \g_CDR_all_tags_clist
825       \CDR_export_set:VcV \l_CDR_file_tl { file } \l_CDR_file_tl

```

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

```

826       \exp_args:NV
827       \CDR_export_get:ccNT \l_CDR_file_tl { lang } \l_CDR_tl {
828         \clist_map_inline:Nn \g_CDR_tags_clist {
829           \CDR_tag_set:ccV { ##1 } { lang } \l_CDR_tl
830         }

```

```

831     }
832   }
833 } {
834   \PackageWarning
835   { coder }
836   { Missing-key~‘tags’ }
837 }
838 }
839 }

```

Files are created at the end of the typesetting process.

```

840 \AddToHook { enddocument / end } {
841   \prop_map_inline:Nn \g_CDR_export_prop {
842     \tl_set:Nn \l_CDR_prop { #2 }
843     \str_set:Nx \l_CDR_str {
844       \prop_item:Nn \l_CDR_prop { file }
845     }
846     \lua_now:n { CDR:export_file('l_CDR_str') }
847     \clist_map_inline:nn {
848       tags, raw, preamble, postamble
849     } {
850       \str_set:Nx \l_CDR_str {
851         \prop_item:Nn \l_CDR_prop { ##1 }
852       }
853       \lua_now:n {
854         CDR:export_file_info('##1','l_CDR_str')
855       }
856     }
857     \lua_now:n { CDR:export_file_complete() }
858   }
859 }

```

12 Style

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

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

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

```

860 \cs_set:Npn \CDR@StyleDefine #1 {
861   \tl_gset:cn { g_CDR@Style/#1 }
862 }

```

<code>\CDR@StyleUse</code>	<code>\CDR@StyleUse {<pygments style name>}</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.

```

863 \cs_set:Npn \CDR@StyleUse #1 {
864   \tl_use:c { g_CDR@Style/#1 }
865 }
866 \cs_set:Npn \CDR@StyleUseTag {
867   \CDR@StyleUse { \CDR_tag_get:c { style } }
868 }

```

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

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

```

869 \prg_new_conditional:Nnn \CDR@StyleIfExist:c { TF } {
870   \tl_if_exist:cTF { g_CDR@Style/#1 } {
871     \prg_return_true:
872   } {
873     \prg_return_false:
874   }
875 }
876 \cs_set_eq:NN \CDR@StyleIfExist \CDR@StyleIfExist:cTF

```

13 Creating display engines

13.1 Utilities

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

```

877 \cs_new:Npn \CDR_code_engine:c #1 {
878   CDR@colored/code/#1:nn
879 }
880 \cs_new:Npn \CDR_block_engine:c #1 {
881   CDR@colored/block/#1
882 }
883 \cs_new:Npn \CDR_code_engine:V {
884   \exp_args:NV \CDR_code_engine:c
885 }
886 \cs_new:Npn \CDR_block_engine:V {
887   \exp_args:NV \CDR_block_engine:c
888 }

```

\l_CDR_engine_tl Storage for an engine name.

```

889 \tl_new:N \l_CDR_engine_tl

```

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

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

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

13.2 Implementation

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

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

```

890 \NewDocumentCommand \CDRCodeEngineNew { mm } {
891   \exp_args:Nx
892   \tl_if_empty:nTF { #1 } {
893     \PackageWarning
894       { coder }
895       { The~engine~cannot~be~void. }
896   } {
897     \cs_new:cpn { \CDR_code_engine:c {#1} } ##1 ##2 {
898       \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
899       #2
900     }
901     \ignorespaces
902   }
903 }

904 \NewDocumentCommand \CDRCodeEngineRenew { mm } {
905   \exp_args:Nx
906   \tl_if_empty:nTF { #1 } {
907     \PackageWarning
908       { coder }
909       { The~engine~cannot~be~void. }
910     \use_none:n
911   } {
912     \cs_if_exist:cTF { \CDR_code_engine:c { #1 } } {
913       \cs_set:cpn { \CDR_code_engine:c { #1 } } ##1 ##2 {
914         \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
915         #2
916       }
917     } {
918       \PackageWarning
919         { coder }
920         { No~code~engine~#1.}
921     }
922     \ignorespaces
923   }
924 }

```

<code>\CDR@CodeEngineApply</code>	<code>\CDR@CodeEngineApply {⟨source⟩}</code>
-----------------------------------	--

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.

```

925 \cs_new:Npn \CDR@CodeEngineApply #1 {
926   \CDR_tag_get:cN { engine } \l_CDR_engine_tl
927   \CDR_if_code_engine:VF \l_CDR_engine_tl {
928     \PackageError
929       { coder }
930       { \l_CDR_engine_tl\space code~engine~unknown,~replaced-by~'default' }
931       {See~\CDRCodeEngineNew~in~the~coder~manual}
932     \tl_set:Nn \l_CDR_engine_tl { default }
933   }
934   \CDR_tag_get:cN { engine~options } \l_CDR_options_tl
935   \tl_if_empty:NTF \l_CDR_options_tl {
936     \CDR_tag_get:cN { \l_CDR_engine_tl\space engine~options } \l_CDR_options_tl
937   } {
938     \tl_put_left:Nx \l_CDR_options_tl {
939       \CDR_tag_get:c { \l_CDR_engine_tl\space engine~options } ,
940     }
941   }
942   \exp_args:NnV
943   \use:c { \CDR_code_engine:V \l_CDR_engine_tl } \l_CDR_options_tl {
944     \CDR_tag_get:c { format }
945     #1
946   }
947 }

```

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

Create a L^AT_EX environment uniquely named after *<engine name>*, which must be a non void string once expanded. The *<begin instructions>* and *<end instructions>* are list of instructions which may refer to the unique argument as #1, which is the value given to CDRBlock environment for key *<engine name>* engine options. Various options are available with the \CDRGetOption function. *Implementation detail:* the third argument is parsed by \NewDocumentEnvironment.

```

948 \NewDocumentCommand \CDRBlockEngineNew { mm } {
949   \NewDocumentEnvironment { \CDR_block_engine:c { #1 } } { m } {
950     \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
951     #2
952   }
953 }

954 \NewDocumentCommand \CDRBlockEngineRenew { mm } {
955   \tl_if_empty:NTF { #1 } {
956     \PackageWarning
957       { coder }
958       { The~engine~cannot~be~void. }
959     \use_none:n
960   } {
961     \RenewDocumentEnvironment { \CDR_block_engine:c { #1 } } { m } {
962       \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
963       #2
964     }
965   }
966 }

```

13.3 Conditionals

`\CDR_if_code_engine:cTF` ★ `\CDR_if_code_engine:cTF {⟨engine name⟩} {⟨true code⟩} {⟨false code⟩}`

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

```
967 \prg_new_conditional:Nnn \CDR_if_code_engine:c { p, T, F, TF } {
968   \cs_if_exist:cTF { \CDR_code_engine:c { #1 } } {
969     \prg_return_true:
970   } {
971     \prg_return_false:
972   }
973 }
974 \prg_new_conditional:Nnn \CDR_if_code_engine:V { p, T, F, TF } {
975   \cs_if_exist:cTF { \CDR_code_engine:V #1 } {
976     \prg_return_true:
977   } {
978     \prg_return_false:
979   }
980 }
```

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

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

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

13.4 Default code engine

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

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

13.5 Default block engine

The default block engine does nothing.

```
996 \CDRBlockEngineNew { default } { } { }
```

13.6 efbox code engine

```
997 \AtBeginDocument {  
998   \@ifpackageloaded{efbox} {  
999     \CDRCodeEngineNew {efbox} {  
1000       \efbox[#1]{#2}%  
1001     }  
1002   }  
1003 }
```

13.7 Block mode default engine

```
1004 \CDRBlockEngineNew {} {  
1005 } {  
1006 }
```

13.8 tcolorbox related engine

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

14 \CDRCode function

14.1 API

\CDR@Sp

\CDR@Sp

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

```
1007 \cs_new:Npn \CDR@DefineSp {  
1008   \CDR_if_tag_truthy:cTF { showspace } {  
1009     \cs_set:Npn \CDR@Sp {{\FancyVerbSpace}}  
1010   } {  
1011     \cs_set_eq:NN \CDR@Sp \space  
1012   }  
1013 }
```

\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.

```
1014 \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.

```
1015 \CDR_tag_keys_define:nn { __code } {
```

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

```
1016   tag .tl_set:N = \l_CDR_tag_tl,
1017   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.

```
1018   engine~options .code:n = \CDR_tag_set:,
1019   engine~options .value_required:n = true,
```

⬢ **__initialize** initialize

```
1020   __initialize .meta:n = {
1021     tag = default,
1022     engine~options = ,
1023   },
1024   __initialize .value_forbidden:n = true,
1025 }
```

14.4 Implementation

```
\CDR_code_format: \CDR_code_format:
```

Private utility to setup the formatting.

```
1026 \cs_new:Npn \CDR_brace_if_contains_comma:n #1 {
1027   \tl_if_in:nnTF { #1 } { , } { { #1 } } { #1 }
1028 }
1029 \cs_generate_variant:Nn \CDR_brace_if_contains_comma:n { V }
1030 \cs_new:Npn \CDR_code_format: {
1031   \frenchspacing
1032   \CDR_tag_get:cN { baselinestretch } \l_CDR_tl
1033   \tl_if_eq:NnF \l_CDR_tl { auto } {
1034     \exp_args:NNV
1035     \def \baselinestretch \l_CDR_tl
1036   }
1037   \CDR_tag_get:cN { fontfamily } \l_CDR_tl
1038   \tl_if_eq:NnT \l_CDR_tl { tt } { \tl_set:Nn \l_CDR_tl { lmtt } }
1039   \exp_args:NV
1040   \fontfamily \l_CDR_tl
1041   \clist_map_inline:nn { series, shape } {
1042     \CDR_tag_get:cN { font##1 } \l_CDR_tl
1043     \tl_if_eq:NnF \l_CDR_tl { auto } {
1044       \exp_args:NnV
1045       \use:c { font##1 } \l_CDR_tl
1046     }
1047   }
```

```

1047 }
1048 \CDR_tag_get:cN { fontsize } \l_CDR_tl
1049 \tl_if_eq:NnF \l_CDR_tl { auto } {
1050   \tl_use:N \l_CDR_tl
1051 }
1052 \selectfont
1053 % \@noligs ?? this is in fancyvrb but does not work here as is
1054 }

```

\CDR_code:n \CDR_code:n <delimiter>

Main utility used by \CDRCode.

```

1055 \cs_new:Npn \CDR_code:n #1 {
1056   \CDR_if_tag_truthy:cTF {pygments} {
1057     \cs_set:Npn \CDR@StyleUseTag {
1058       \CDR@StyleUse { \CDR_tag_get:c { style } }
1059       \cs_set_eq:NN \CDR@StyleUseTag \prg_do_nothing:
1060     }
1061     \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1062       __fancyvrb,
1063     }
1064     \CDR_tag_keys_set:nV { __local } \l_CDR_keyval_tl
1065     \DefineShortVerb { #1 }
1066     \SaveVerb [
1067       aftersave = {
1068         \exp_args:Nx \UndefineShortVerb { #1 }
1069         \lua_now:n { CDR:highlight_code_setup() }
1070         \CDR_tag_get:cN {lang} \l_CDR_tl
1071         \lua_now:n { CDR:highlight_set_var('lang') }
1072         \CDR_tag_get:cN {cache} \l_CDR_tl
1073         \lua_now:n { CDR:highlight_set_var('cache') }
1074         \CDR_tag_get:cN {debug} \l_CDR_tl
1075         \lua_now:n { CDR:highlight_set_var('debug') }
1076         \CDR_tag_get:cN {style} \l_CDR_tl
1077         \lua_now:n { CDR:highlight_set_var('style') }
1078         \lua_now:n { CDR:highlight_set_var('source', 'FV@SV@CDR@Source') }
1079         \FV@UseKeyValues
1080         \frenchspacing
1081         % \FV@SetupFont Break
1082         \FV@DefineWhiteSpace
1083         \FancyVerbDefineActive
1084         \FancyVerbFormatCom
1085         \CDR_code_format:
1086         \CDR@DefineSp
1087         \CDR_tag_get:c { format }
1088         \CDR@DefineSp
1089         \CDR@CodeEngineApply {
1090           \CDR@StyleIfExist { \l_CDR_tl } {
1091             \CDR@StyleUseTag
1092             \lua_now:n { CDR:highlight_source(false, true) }
1093           } {
1094             \lua_now:n { CDR:highlight_source(true, true) }
1095             \input { \l_CDR_pyg_sty_tl }

```

```

1096         \CDR@StyleUseTag
1097     }
1098     \makeatletter
1099     \input { \l_CDR_pyg_tex_tl }
1100     \makeatother
1101 }
1102 \group_end:
1103 }
1104 ] { CDR@Source } #1
1105 } {
1106     \exp_args:NV \fvset \l_CDR_keyval_tl
1107     \DefineShortVerb { #1 }
1108     \SaveVerb [
1109         aftersave = {
1110             \UndefinedShortVerb { #1 }
1111             \cs_set_eq:NN \CDR@FormattingPrep \FV@FormattingPrep
1112             \cs_set:Npn \FV@FormattingPrep {
1113                 \CDR@FormattingPrep
1114                 \CDR_tag_get:c { format }
1115             }
1116             \CDR@CodeEngineApply { \mbox {
1117                 \FV@UseKeyValues
1118                 \FV@FormattingPrep
1119                 \FV@SV@CDR@Code
1120             } }
1121             \group_end:
1122         }
1123     ] { CDR@Code } #1
1124 }
1125 }

1126 \NewDocumentCommand \CDRCode { O{} } {
1127     \group_begin:
1128     \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
1129         \prg_return_false:
1130     }
1131     \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1132         __code, default.code, __pygments, default,
1133     }
1134     \CDR_tag_keys_set_known:nnN { __local } { #1 } \l_CDR_keyval_tl
1135     \CDR_tag_provide_from_keyval:V \l_CDR_keyval_tl
1136     \CDR_tag_keys_set_known:nVN { __local } \l_CDR_keyval_tl \l_CDR_keyval_tl
1137     \exp_args:NNV
1138     \def \FV@KeyValues \l_CDR_keyval_tl
1139     \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1140         __fancyvrb,
1141     }
1142     \CDR_tag_keys_set:nV { __local } \l_CDR_keyval_tl
1143     \CDR_tag_inherit:cf { __local } {
1144         \tl_if_empty:NF \l_CDR_tag_tl { \l_CDR_tag_tl, }
1145         __code, default.code, __pygments, default, __fancyvrb,
1146     }
1147     \CDR_code:n
1148 }
1149 \cs_set:Npn \CDR_code:n #1 {

```

```

1150 \CDR_if_tag_truthy:cTF {pygments} {
1151   \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1152     __fancyvrb,
1153   }
1154   \CDR_tag_keys_set:nV { __local } \l_CDR_keyval_tl
1155   \DefineShortVerb { #1 }
1156   \SaveVerb [
1157     aftersave = {
1158       \exp_args:Nx \UndefineShortVerb { #1 }
1159       \lua_now:n { CDR:highlight_code_setup() }
1160       \CDR_tag_get:cN {lang} \l_CDR_tl
1161       \lua_now:n { CDR:highlight_set_var('lang') }
1162       \CDR_tag_get:cN {cache} \l_CDR_tl
1163       \lua_now:n { CDR:highlight_set_var('cache') }
1164       \CDR_tag_get:cN {debug} \l_CDR_tl
1165       \lua_now:n { CDR:highlight_set_var('debug') }
1166       \CDR_tag_get:cN {style} \l_CDR_tl
1167       \lua_now:n { CDR:highlight_set_var('style') }
1168       \lua_now:n { CDR:highlight_set_var('source', 'FV@SV@CDR@Source') }
1169       \exp_args:NNV
1170       \def \FV@KeyValues \l_CDR_keyval_tl
1171       \FV@UseKeyValues
1172       \frenchspacing
1173       % \FV@SetupFont Break
1174       \FV@DefineWhiteSpace
1175       \FancyVerbDefineActive
1176       \FancyVerbFormatCom
1177       \CDR@DefineSp
1178       \CDR_code_format:
1179       \CDR_tag_get:c { format }
1180       \CDR@CodeEngineApply {
1181         \CDR@StyleIfExist { \CDR_tag_get:c {style} } {
1182           \CDR@StyleUseTag
1183           \lua_now:n { CDR:highlight_source(false, true) }
1184         } {
1185           \lua_now:n { CDR:highlight_source(true, true) }
1186           \input { \l_CDR_pyg_sty_tl }
1187           \CDR@StyleUseTag
1188         }
1189         \makeatletter
1190         \input { \l_CDR_pyg_tex_tl }
1191         \makeatother
1192       }
1193     \group_end:
1194   }
1195 ] { CDR@Source } #1
1196 } {
1197   \DefineShortVerb { #1 }
1198   \SaveVerb [
1199     aftersave = {
1200       \UndefineShortVerb { #1 }
1201       \cs_set_eq:NN \CDR@FormattingPrep \FV@FormattingPrep
1202       \cs_set:Npn \FV@FormattingPrep {
1203         \CDR@FormattingPrep

```



```

1204         \CDR_tag_get:c { format }
1205     }
1206     \CDR@CodeEngineApply { A \mbox { a
1207         \exp_args:NNV
1208         \def \FV@KeyValues \l_CDR_keyval_tl
1209         \FV@UseKeyValues
1210         \FV@FormattingPrep
1211         \@nameuse{FV@SV@CDR@Code}
1212     z } Z }
1213     \group_end:
1214 }
1215 ] { CDR@Code } #1
1216 }
1217 }

1218 \RenewDocumentCommand \CDRCode { O{} } {
1219     \group_begin:
1220     \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
1221         \prg_return_false:
1222     }
1223     \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1224         __code, default.code, __pygments, default,
1225     }
1226     \CDR_tag_keys_set_known:nnN { __local } { #1 } \l_CDR_keyval_tl
1227     \CDR_tag_provide_from_keyval:V \l_CDR_keyval_tl
1228     \CDR_tag_keys_set_known:nVN { __local } \l_CDR_keyval_tl \l_CDR_keyval_tl
1229     \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1230         __fancyvrb,
1231     }
1232     \CDR_tag_keys_set:nV { __local } \l_CDR_keyval_tl
1233     \CDR_tag_inherit:cf { __local } {
1234         \tl_if_empty:NF \l_CDR_tag_tl { \l_CDR_tag_tl, }
1235         __code, default.code, __pygments, default, __fancyvrb,
1236     }
1237     \fvset{showspaces}
1238     \CDR_code:n
1239 }

```

15 CDRBlock environment

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

15.1 Storage

`\l_CDR_block_prop`

1240 `\prop_new:N \l_CDR_block_prop`

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

15.2 __block l3keys module

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

```

1241 \CDR_tag_keys_define:nn { __block } {
  no export[=true|false] to ignore this code chunk at export time.

1242   no-export .code:n = \CDR_tag_boolean_set:x { #1 },
1243   no-export .default:n = true,

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

1244   no-export~format .code:n = \CDR_tag_set:,
1245   no-export~format .value_required:n = true,

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

1246   test .code:n = \CDR_tag_boolean_set:x { #1 },
1247   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.

1248   engine-options .code:n = \CDR_tag_set:,
1249   engine-options .value_required:n = true,

  __initialize initialize

1250   __initialize .meta:n = {
1251     no-export = false,
1252     no-export~format = ,
1253     test = false,
1254     engine-options = ,
1255   },
1256   __initialize .value_forbidden:n = true,
1257 }

```

15.3 Implementation

We start by saving some fancyvrb macros that we further want to extend. The unique mandatory argument of these macros will eventually be recorded to be saved later on.

```

1258 \clist_map_inline:nn { i, ii, iii, iv } {
1259   \cs_set_eq:cc { CDR@ListProcessLine@ #1 } { FV@ListProcessLine@ #1 }
1260 }
1261 \cs_new:Npn \CDR_process_line:n #1 {
1262   \str_set:Nn \l_CDR_str { #1 }
1263   \lua_now:n {CDR:record_line('l_CDR_str')}
1264 }

1265 \def\FVB@CDRBlock {
1266   \@sphack
1267   \group_begin:
1268   \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
1269     \prg_return_true:
1270   }
1271   \CDR_tag_keys_set:nn { __block } { __initialize }

```

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

```

1272 \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1273   __block, __pygments.block, default.block,
1274   __pygments, default,
1275 }
1276 \CDR_tag_keys_set_known:nVN { __local } \FV@KeyValues \l_CDR_keyval_tl
1277 \CDR_tag_provide_from_keyval:V \l_CDR_keyval_tl
1278 \CDR_tag_keys_set_known:nVN { __local } \l_CDR_keyval_tl \l_CDR_keyval_tl

```

By default, this code chunk will have the same list of tags as the last code block or last `\CDRExport` stored in `\g_CDR_tags_clist`. This can be overwritten with the `tags=...` user interface. At least one tag must be provided.

```

1279 \CDR_tag_inherit:cn { __local } { default.block }
1280 \CDR_tag_get:cn { tags } \l_CDR_clist
1281 \clist_if_empty:NTF \l_CDR_clist {
1282   \clist_if_empty:NT \g_CDR_tags_clist {
1283     \PackageWarning
1284       { coder }
1285       { No~(default)~tags~provided. }
1286   }
1287 } {
1288   \clist_gset_eq:NN \g_CDR_tags_clist \l_CDR_clist
1289 }
1290 \lua_now:n {
1291   CDR:highlight_block_setup('g_CDR_tags_clist')
1292 }

```

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

```

1293 \bool_set_false:N \l_CDR_pygments_bool
1294 \clist_if_empty:NTF \g_CDR_tags_clist {
1295   \bool_set:Nn \l_CDR_pygments_bool {
1296     \CDR_if_tag_truthy_p:c { pygments }
1297   }
1298 } {
1299   \bool_if:NF \l_CDR_pygments_bool {
1300     \clist_map_inline:Nn \g_CDR_tags_clist {
1301       \CDR_if_tag_truthy:ccT { ##1 } { pygments } {
1302         \clist_map_break:n {
1303           \bool_set_true:N \l_CDR_pygments_bool
1304         }
1305       }
1306     }
1307   }
1308 }

```

Now we setup the full inheritance tree.

```

1309 \CDR_tag_inherit:cf { __local } {
1310   \g_CDR_tags_clist,
1311   __block, default.block, __pygments.block, __fancyvrb.block, __fancyvrb.number,
1312   __pygments, default, __fancyvrb,

```

```

1313 }
1314 \bool_if:NTF \l_CDR_pygments_bool {
1315   \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1316     __fancyvrb.number
1317   }
1318   \CDR_tag_keys_set_known:nVN { __local } \l_CDR_keyval_tl \l_CDR_keyval_tl
1319   \exp_args:NV \fvset \l_CDR_keyval_tl
1320   \CDR_keys_inherit:Vnn \c_CDR_tag { __local } {
1321     __fancyvrb, __fancyvrb.block
1322   }
1323   \exp_args:NnV
1324   \CDR_tag_keys_set:nn { __local } \l_CDR_keyval_tl
1325   \exp_args:NNV
1326   \def \FV@KeyValues \l_CDR_keyval_tl

```

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

```

1327   \CDR_tag_get:cN {lang} \l_CDR_tl
1328   \lua_now:n { CDR:highlight_set_var('lang') }
1329   \CDR_tag_get:cN {cache} \l_CDR_tl
1330   \lua_now:n { CDR:highlight_set_var('cache') }
1331   \CDR_tag_get:cN {debug} \l_CDR_tl
1332   \lua_now:n { CDR:highlight_set_var('debug') }
1333   \CDR_tag_get:cN {style} \l_CDR_tl
1334   \lua_now:n { CDR:highlight_set_var('style') }
1335   \CDR@StyleIfExist { \l_CDR_tl } { } {
1336     \lua_now:n { CDR:highlight_source(true, false) }
1337     \input { \l_CDR_pyg_sty_tl }
1338   }
1339   \CDR@StyleUseTag
1340   \CDR_if_tag_truthy:cTF {no~export} {
1341     \clist_map_inline:nn { i, ii, iii, iv } {
1342       \cs_set:cpn { FV@ListProcessLine@ ##1 } #####1 {
1343         \tl_set:Nn \l_CDR_tl { #####1 }
1344         \lua_now:n { CDR:record_line('l_CDR_tl') }
1345       }
1346     }
1347   } {
1348     \clist_map_inline:nn { i, ii, iii, iv } {
1349       \cs_set:cpn { FV@ListProcessLine@ ##1 } #####1 {
1350         \tl_set:Nn \l_CDR_tl { #####1 }
1351         \lua_now:n { CDR:record_line('l_CDR_tl') }
1352       }
1353     }
1354   }
1355   \CDR_tag_get:cN { engine } \l_CDR_engine_tl
1356   \CDR_if_code_engine:VF \l_CDR_engine_tl {
1357     \PackageError
1358     { coder }
1359     { \l_CDR_engine_tl\space block~engine~unknown,~replaced~by~'default' }
1360     { See~\CDRBlockEngineNew~in~the~coder~manual }
1361     \tl_set:Nn \l_CDR_engine_tl { default }
1362   }
1363   \CDR_tag_get:cN { \l_CDR_engine_tl~engine~options } \l_CDR_options_tl
1364   \exp_args:NnV

```

```

1365 \use:c { \CDR_block_engine:V \l_CDR_engine_tl } \l_CDR_options_tl
1366
1367 \def\FV@ProcessLine ##1 {
1368   \tl_set:Nn \l_CDR_tl { ##1 }
1369   \lua_now:n { CDR:record_line('l_CDR_tl') }
1370 }
1371 } {
1372   \exp_args:NNV
1373   \def \FV@KeyValues \l_CDR_keyval_tl
1374   \CDR_if_tag_truthy:cF {no~export} {
1375     \clist_map_inline:nn { i, ii, iii, iv } {
1376       \cs_set:cpn { FV@ListProcessLine@ ##1 } #####1 {
1377         \tl_set:Nn \l_CDR_tl { #####1 }
1378         \lua_now:n { CDR:record_line('l_CDR_tl') }
1379         \use:c { CDR@ListProcessLine@ ##1 } { #####1 }
1380       }
1381     }
1382   }
1383   \exp_args:NnV
1384   \use:c { \CDR_block_engine:V \l_CDR_engine_tl } \l_CDR_options_tl
1385   \FV@VerbatimBegin
1386 }
1387 \FV@Scan
1388 }
1389 \def\FVE@CDRBlock {
1390   \bool_if:NT \l_CDR_pygments_bool {
1391     \CDR_tag_get:c { format }
1392     \fvset{ commandchars=\\{\} }
1393     \CDR@DefineSp
1394     \FV@VerbatimBegin
1395     \lua_now:n { CDR:highlight_source(false, true) }
1396     \makeatletter
1397     \input{ \l_CDR_pyg_tex_tl }
1398     \makeatother
1399   }
1400   \FV@VerbatimEnd
1401   \use:c { end \CDR_block_engine:V \l_CDR_engine_tl }
1402   \group_end:
1403   \@esphack
1404 }
1405 \DefineVerbatimEnvironment{CDRBlock}{CDRBlock}{-}
1406

```

16 Management

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

```
1407 \bool_new:N \g_CDR_in_impl_bool
```

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

`\CDR_if_show_code:TF` `\CDR_if_show_code:TF {⟨true code⟩} {⟨false code⟩}`
 Execute *⟨true code⟩* when code should be printed, *⟨false code⟩* otherwise.

```

1408 \prg_new_conditional:Nnn \CDR_if_show_code: { T, F, TF } {
1409   \bool_if:nTF {
1410     \g_CDR_in_impl_bool && !\g_CDR_with_impl_bool
1411   } {
1412     \prg_return_false:
1413   } {
1414     \prg_return_true:
1415   }
1416 }
```

`\g_CDR_with_impl_bool`

```

1417 \bool_new:N \g_CDR_with_impl_bool

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

`\CDRPreamble` `\CDRPreamble {⟨variable⟩} {⟨file name⟩}`
 Store the content of *⟨file name⟩* into the variable *⟨variable⟩*.

```

1418 \DeclareDocumentCommand \CDRPreamble { m m } {
1419   \msg_info:nnn
1420     { coder }
1421     { :n }
1422     { Reading-preamble-from-file-"#2". }
1423   \group_begin:
1424     \tl_set:Nn \l_tmpa_tl { #2 }
1425     \exp_args:NNNx
1426     \group_end:
1427     \tl_set:Nx #1 { \lua_now:n {CDR.print_file_content('l_tmpa_tl')} }
1428 }
```

17 Section separators

`\CDRImplementation` `\CDRImplementation`
`\CDRFinale` `\CDRFinale`

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

18 Finale

```

1429 \newcounter{CDR@impl@page}
1430 \DeclareDocumentCommand \CDRImplementation {} {
1431   \bool_if:NF \g_CDR_with_impl_bool {
1432     \clearpage
1433     \bool_gset_true:N \g_CDR_in_impl_bool
1434     \let\CDR@old@part\part
```

```

1435 \DeclareDocumentCommand\part{som}{}
1436 \let\CDR@old@section\section
1437 \DeclareDocumentCommand\section{som}{}
1438 \let\CDR@old@subsection\subsection
1439 \DeclareDocumentCommand\subsection{som}{}
1440 \let\CDR@old@subsubsection\subsubsection
1441 \DeclareDocumentCommand\subsubsection{som}{}
1442 \let\CDR@old@paragraph\paragraph
1443 \DeclareDocumentCommand\paragraph{som}{}
1444 \let\CDR@old@subparagraph\subparagraph
1445 \DeclareDocumentCommand\subparagraph{som}{}
1446 \cs_if_exist:NT \refsection{ \refsection }
1447 \setcounter{ CDR@impl@page }{ \value{page} }
1448 }
1449 }
1450 \DeclareDocumentCommand\CDRFinale {} {
1451 \bool_if:NF \g_CDR_with_impl_bool {
1452 \clearpage
1453 \bool_gset_false:N \g_CDR_in_impl_bool
1454 \let\part\CDR@old@part
1455 \let\section\CDR@old@section
1456 \let\subsection\CDR@old@subsection
1457 \let\subsubsection\CDR@old@subsubsection
1458 \let\paragraph\CDR@old@paragraph
1459 \let\subparagraph\CDR@old@subparagraph
1460 \setcounter { page } { \value{ CDR@impl@page } }
1461 }
1462 }
1463 \cs_set_eq:NN \CDR_line_number: \prg_do_nothing:

```

19 Finale

```

1464 %\AddToHook { cmd/FancyVerbFormatLine/before } {
1465 % \CDR_line_number:
1466 %}
1467 \AddToHook { shipout/before } {
1468 \tl_gclear:N \g_CDR_chunks_tl
1469 }

1470 % =====
1471 % Auxiliary:
1472 % finding the widest string in a comma
1473 % separated list of strings delimited by parenthesis
1474 % =====
1475
1476 % arguments:
1477 % #1) text: a comma separated list of strings
1478 % #2) formatter: a macro to format each string
1479 % #3) dimension: will hold the result
1480
1481 \cs_new:Npn \CDRWidest (#1) #2 #3 {
1482 \group_begin:
1483 \dim_set:Nn #3 { Opt }

```

```

1484 \clist_map_inline:nn { #1 } {
1485   \hbox_set:Nn \l_tmpa_box { #2{##1} }
1486   \dim_set:Nn \l_tmpa_dim { \dim_eval:n { \box_wd:N \l_tmpa_box } }
1487   \dim_compare:nNnT { #3 } < { \l_tmpa_dim } {
1488     \dim_set_eq:NN #3 \l_tm pa_dim
1489   }
1490 }
1491 \exp_args:NNNV
1492 \group_end:
1493 \dim_set:Nn #3 #3
1494 }
1495 \ExplSyntaxOff
1496

```

20 pygmentex implementation

```

1497 % =====
1498 % fancyvrb new commands to append to a file
1499 % =====
1500
1501 % See http://tex.stackexchange.com/questions/47462/inputenc-error-with-unicode-chars-and-verbatim
1502
1503 \ExplSyntaxOn
1504
1505 \seq_new:N \l_CDR_records_seq
1506
1507 % =====
1508 % Main options
1509 % =====
1510
1511 \newif\ifCDR@left
1512 \newif\ifCDR@right
1513
1514

```

20.1 options key-value controls

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

21 Something else

```

1515
1516 % =====
1517 % pygmented commands and environments
1518 % =====
1519
1520
1521 \cs_generate_variant:Nn \exp_last_unbraced:NnNo { NxNo }
1522
1523
1524 % ERROR: JL undefined \CDR@alllinenos

```



```

1525
1526 \ProvideDocumentCommand\captionof{mm}{-}
1527 \def\CDR@alllinenos{(0)}
1528
1529 \def\FormatLineNumber#1{{\rmfamily\tiny#1}}
1530
1531 \newdimen\CDR@leftmargin
1532 \newdimen\CDR@linenosep
1533
1534 %
1535 %\newcommand\CDR@tcbox@more@options{%
1536 % nobeforeafter,%
1537 % tcbox-raise-base,%
1538 % left=0mm,%
1539 % right=0mm,%
1540 % top=0mm,%
1541 % bottom=0mm,%
1542 % boxsep=2pt,%
1543 % arc=1pt,%
1544 % boxrule=0pt,%
1545 % \CDR_options_if_in:nT {colback} {
1546 %   colback=\CDR:n {colback}
1547 % }
1548 %}
1549 %
1550 %\newcommand\CDR@mdframed@more@options{%
1551 % leftmargin=\CDR@leftmargin,%
1552 % frametitlerule=true,%
1553 % \CDR_if_in:nT {colback} {
1554 %   backgroundcolor=\CDR:n {colback}
1555 % }
1556 %}
1557 %
1558 %\newcommand\CDR@tcolorbox@more@options{%
1559 % grow-to-left-by=-\CDR@leftmargin,%
1560 % \CDR_if_in:nNT {colback} {
1561 %   colback=\CDR:n {colback}
1562 % }
1563 %}
1564 %
1565 %\newcommand\CDR@boite@more@options{%
1566 % leftmargin=\CDR@leftmargin,%
1567 % \ifcsname CDR@opt@colback\endcsname
1568 %   colback=\CDR@opt@colback,%
1569 % \fi
1570 %}
1571 %
1572 %\newcommand\CDR@mdframed@margin{%
1573 % \advance \CDR@linenosep \mdflength{outerlinewidth}%
1574 % \advance \CDR@linenosep \mdflength{middlelinewidth}%
1575 % \advance \CDR@linenosep \mdflength{innerlinewidth}%
1576 % \advance \CDR@linenosep \mdflength{innerleftmargin}%
1577 %}
1578 %

```

```

1579 %\newcommand\CDR@tcolorbox@margin{%
1580 % \advance \CDR@linenosep \kvtcb@left@rule
1581 % \advance \CDR@linenosep \kvtcb@leftupper
1582 % \advance \CDR@linenosep \kvtcb@boxsep
1583 %}
1584 %
1585 %\newcommand\CDR@boite@margin{%
1586 % \advance \CDR@linenosep \boite@leftrule
1587 % \advance \CDR@linenosep \boite@boxsep
1588 %}
1589 %
1590 %\def\CDR@global@options{}
1591 %
1592 %\newcommand\setpygmented[1]{%
1593 % \def\CDR@global@options{/CDR.cd,#1}%
1594 %}
1595
1596 \ExplSyntaxOff

1597 %</sty>

```