# coder — code inlined in a LaTeX document[*]

Jérôme LAURENS[†]

Released 2022/02/07

**Abstract**

Usually, documentation is put inside the code, coder allows to work the other way round by putting code inside the documentation. This is particularly interesting when different code files share some logic and should be documented all at once. The file `coder-manual.pdf` gives different examples. Here is the implementation of the package.
This LaTeX package requires LuaTeX and may use syntax coloring based on the pygments[1] package.

## 1 Package dependencies

datetime2, xcolor, fancyvrb and dependencies of these packages.

## 2 Similar technologies

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

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

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

## 3 Known bugs and limitations

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

[*]This file describes version 1.0a, last revised 2022/02/07.
[†]E-mail: jerome.laurens@u-bourgogne.fr
[1]The coder package has been tested with pygments version 2.11.2

## 4 Presentation

coder is a triptych of three complementary components

1. coder.sty, on the 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 instert code snippets as blocks. Moreover, block code chunks can be exported to files, once declared with `\CDRExport` command. The `\CDRSet` command is used to set various parameters, including display engines declared with either `\CDRCodeEngineNew` or `\CDRBlockEngineNew`[2].

### 4.1 Code flow

The normal code flow is

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

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

### 4.2 File exportation

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

[2]Work in progress

### 4.3 Display engine

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

### 4.4 LaTeX user interface

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

### 4.5 Properties and inheritance

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

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

## 5 Namespace and conventions

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

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

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

## 6 Options

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

### 6.1 fancyvrb

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

**`formatcom=`**⟨*command*⟩ execute before printing verbatim text. Initially empty. Ignored in `code` mode.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## 6.2 pygments options

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

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

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

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

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

🚫 `outencoding` Overrides `encoding` if given.

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

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

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

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

🚫 `linenostep=`⟨*integer*⟩ If set to a number n > 1, only every nth line number is printed. Ignored in `code` mode. Additional options given to the Verbatim environment (see the fancyvrb docs for possible values). Initially empty.

🚫 `verboptions` Forbidden.

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

🛑 `texcomments[=true|false]` If set to `true`, enables LaTeX comment lines. That is, LaTeX markup in comment tokens is not escaped so that LaTeX can render it. Initially `false`. Ignored in code mode.

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

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

⚙ `envname=`⟨*name*⟩ Allows you to pick an alternative environment name replacing `Verbatim`. The alternate environment still has to support `Verbatim`'s option syntax. Initially `Verbatim`.

## 6.3 LaTeX

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

🛑 `tags` `clist` of tag names, used for line numbering.

🛑 `inline` `true` when inline code is concerned, false otherwise.

🛑 `sty_template` LaTeX source text where `<placeholder:style_defs>` must be replaced by the style definitions provided by pygments. It may include the style name.

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

**`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

```
%<*lua>
local lfs   = _ENV.lfs
local tex   = _ENV.tex
local token = _ENV.token
local md5   = _ENV.md5
local kpse  = _ENV.kpse
local rep   = string.rep
local lpeg  = require("lpeg")
local P, Cg, Cp, V = lpeg.P, lpeg.Cg, lpeg.Cp, lpeg.V
local json  = require('lualibs-util-jsn')
```

## 3 General purpose material

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

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

(*End definition for `CDR_PY_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. On return, print `true` or `false` in the TeX stream to indicate whether pygments is available.

```
local function set_python_path(self, path_var)
  local path, mode, _, __
  if path_var then
    path = assert(token.get_macro(path_var))
    mode,_,__ = lfs.attributes(path,'mode')
    print('**** CDR mode', mode)
    assert(mode == 'file' or mode == 'link')
  else
    path = io.popen([[which python]]):read('a'):match("^%s*(.-)%s*$")
  end
  self.PYTHON_PATH = path
  print('**** CDR python path', self.PYTHON_PATH)
  path = path:match("^(.+/)")..'pygmentize'
  mode,_,__ = lfs.attributes(path,'mode')
  print('**** CDR path, mode', path, mode)
  if mode == 'file' or mode == 'link' then
    self.PYGMENTIZE_PATH = path
    tex.print('true')
  else
    self.PYGMENTIZE_PATH = ''
    tex.print('false')
  end
end
```

`JSON_boolean_true`
`JSON_boolean_false`

Special marker to encode booleans in JSON files. These are table which `__cls__` field is either `BooleanTrue` or `BooleanFalse`.

(*End definition for* `JSON_boolean_true` *and* `JSON_boolean_false`. *These variables are documented on page* ??.)

```
local JSON_boolean_true = {
  __cls__ = 'BooleanTrue',
}
local JSON_boolean_false = {
  __cls__ = 'BooleanFalse',
}
```

`is_truthy`

```
if CDR.is_truthy(⟨what⟩) then
⟨true code⟩
else
⟨false code⟩
end
```

Execute ⟨*true code*⟩ if ⟨*what*⟩ is `JSON_boolean_true` or the string "true", ⟨*false code*⟩ otherwise.

```
local function is_truthy(s)
  return s == JSON_boolean_true or s == 'true'
end
```

`escape`

```
⟨variable⟩ = CDR.escape(⟨string⟩)
```

🛑 Escape the given string to be used by the shell.

```
local function escape(s)
  s = s:gsub(' ','\\ ')
  s = s:gsub('\\','\\\\')
  s = s:gsub('\r','\\r')
  s = s:gsub('\n','\\n')
  s = s:gsub('"','\\"')
  s = s:gsub("'","\\'")
  return s
end
```

`make_directory` ⟨*variable*⟩ = `CDR.make_directory(`⟨*string path*⟩`)`

Make a directory at the given path.

```
local function make_directory(path)
  local mode,_,__ = lfs.attributes(path,"mode")
  if mode == "directory" then
    return true
  elseif mode ~= nil then
    return nil,path.." exist and is not a directory",1
  end
  if os["type"] == "windows" then
    path = path:gsub("/", "\\")
    _,_,__ = os.execute(
      "if not exist "  .. path .. "\\nul " .. "mkdir " .. path
    )
  else
    _,_,__ = os.execute("mkdir -p " .. path)
  end
  mode = lfs.attributes(path,"mode")
  if mode == "directory" then
    return true
  end
  return nil,path.." exist and is not a directory",1
end
```

`dir_p` The directory where the auxiliary pygments related files are saved, in general ⟨*jobname*⟩`.pygd/`.

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

`json_p` The path of the JSON file used to communicate with coder-tool.py, in general ⟨*jobname*⟩`.pygd/`⟨*jobname*⟩

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

```
local dir_p, json_p
local jobname = tex.jobname
dir_p = './'..jobname..'.pygd/'
if make_directory(dir_p) == nil then
  dir_p = './'
  json_p = dir_p..jobname..'.pyg.json'
else
  json_p = dir_p..'input.pyg.json'
end
```

print_file_content CDR.print_file_content(⟨*macro name*⟩)

The command named ⟨*macro name*⟩ contains the path to a file. Read the content of that file and print the result to the TeX stream.

```lua
local function print_file_content(name)
  local p = token.get_macro(name)
  local fh = assert(io.open(p, 'r'))
  local s = fh:read('a')
  fh:close()
  tex.print(s)
end
```

safe_equals ⟨*variable*⟩ = safe_equals(⟨*string*⟩)

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

```lua
local eq_pattern = P({ Cp() * P('=')^1 * Cp() + P(1) * V(1) })
local function safe_equals(s)
  local i, j = 0, 0
  local max = 0
  while true do
    i, j = eq_pattern:match(s, j)
    if i == nil then
      return rep('=', max + 1)
    end
    i = j - i
    if i > max then
      max = i
    end
  end
end
```

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

Class method. Loads the given ⟨*lua code chunk*⟩ and execute it. On error, messages are printed.

```lua
local function load_exec(self, chunk)
  local env = setmetatable({ self = self, tex = tex }, _ENV)
  local func, err = load(chunk, 'coder-tool', 't', env)
  if func then
    local ok
    ok, err = pcall(func)
    if not ok then
      print("coder-util.lua Execution error:", err)
      print('chunk:', chunk)
    end
  else
    print("coder-util.lua Compilation error:", err)
    print('chunk:', chunk)
  end
end
```

`load_exec_output` `CDR:load_exec_output(⟨lua code chunk⟩)`

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

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

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

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

```
local parse_pattern
do
  local tag = P('!') + '*' + '?'
  local stp = '>>>>>'
  local cmd = (P(1) - stp)^0
  parse_pattern = P({
    P('<<<<<') * Cg(tag) * 'LUA:' * Cg(cmd) * stp * Cp() + 1 * V(1)
  })
end
local function load_exec_output(self, s)
  local i, tag, cmd
  i = 1
  while true do
    tag, cmd, i = parse_pattern:match(s, i)
    if tag == '!' then
      self:load_exec(cmd)
    elseif tag == '*' then
      local eqs = safe_equals(cmd)
      cmd = '['..eqs..'['..cmd..']'..eqs..']'
      tex.print([[%
\directlua{CDR:load_exec(]]..cmd..[[)}%
]])
    elseif tag == '?' then
      print('\nDEBUG/coder: '..cmd)
    else
      return
    end
  end
end
```

## 4 Properties

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

## 5 Hiligting

### 5.1 Common

hilight_set

`CDR:hilight_set(...)`

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

```
local function hilight_set(self, key, value)
  local args = self['.arguments']
  local t = args
  if t[key] == nil then
    t = args.pygopts
    if t[key] == nil then
      t = args.texopts
      if t[key] == nil then
        t = args.fv_opts
        assert(t[key] ~= nil)
      end
    end
  end
  if t[key] == JSON_boolean_true or t[key] == JSON_boolean_false then
    t[key] = value == true and JSON_boolean_true or JSON_boolean_false
  else
    t[key] = value
  end
end

local function hilight_set_var(self, key, var)
  self:hilight_set(key, assert(token.get_macro(var or 'l_CDR_tl')))
end
```

hilight_source

`CDR:hilight_source(`⟨*src*⟩`, `⟨*sty*⟩`)`

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

```
local function hilight_source(self, sty, src)
  local args = self['.arguments']
  local texopts = args.texopts
  local pygopts = args.pygopts
  local inline = self.is_truthy(texopts.is_inline)
  local use_cache = self.is_truthy(args.cache)
  local use_py = false
  local cmd = self.PYTHON_PATH..' '..self.CDR_PY_PATH
  local debug = args.debug
  local pyg_sty_p
  if sty then
    pyg_sty_p = self.dir_p..pygopts.style..'.pyg.sty'
    token.set_macro('l_CDR_pyg_sty_tl', pyg_sty_p)
    texopts.pyg_sty_p = pyg_sty_p
    local mode,_,__ = lfs.attributes(pyg_sty_p, 'mode')
    if not mode or not use_cache then
      use_py = true
      if debug then
        print('PYTHON STYLE:')
      end
      cmd = cmd..(' --create_style')
    end
    self:cache_record(pyg_sty_p)
  end
  local pyg_tex_p
  if src then
    local source
    if inline then
      source = args.source
    else
      local ll = self['.lines']
      source = table.concat(ll, '\n')
    end
    local hash = md5.sumhexa( ('%s:%s:%s'
      ):format(
        source,
        inline and 'code' or 'block',
        pygopts.style
      )
    )
    local base = self.dir_p..hash
    pyg_tex_p = base..'.pyg.tex'
    token.set_macro('l_CDR_pyg_tex_tl', pyg_tex_p)
    local mode,_,__ = lfs.attributes(pyg_tex_p,'mode')
    if not mode or not use_cache then
      use_py = true
      if debug then
        print('PYTHON SOURCE:', inline)
      end
      if not inline then
        local tex_p = base..'.tex'
        local f = assert(io.open(tex_p, 'w'))
        local ok, err = f:write(source)
        f:close()
        if not ok then
          print('File error('..tex_p..'): '..err)
        end
        if debug then
          print('OUTPUT: '..tex_p)
        end
      end
      cmd = cmd..(' --base=%q'):format(base)
    end
  end
  if use_py then
    local json_p = self.json_p
    local f = assert(io.open(json_p, 'w'))
    local ok, err = f:write(json.tostring(args, true))
    f:close()
    if not ok then
      print('File error('..json_p..'): '..err)
    end
    cmd = cmd..('  %q'):format(json_p)
    if debug then
      print('CDR>'..cmd)
    end
    local o = io.popen(cmd):read('a')
    self:load_exec_output(o)
    if debug then
      print('PYTHON', o)
    end
  end
  self:cache_record(
    sty and pyg_sty_p or nil,
    src and pyg_tex_p or nil
  )
end
```

## 5.2 Code

`hilight_code_setup` `CDR:hilight_code_setup()`

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

```
local function hilight_code_setup(self)
  self['.arguments'] = {
    __cls__ = 'Arguments',
    source  = '',
    cache   = JSON_boolean_true,
    debug   = JSON_boolean_false,
    pygopts = {
      __cls__ = 'PygOpts',
      lang    = 'tex',
      style   = 'default',
      mathescape   = JSON_boolean_false,
      escapeinside = '',
    },
    texopts = {
      __cls__ = 'TeXOpts',
      tags    = '',
      is_inline = JSON_boolean_true,
      pyg_sty_p = '',
    },
    fv_opts = {
      __cls__ = 'FVOpts',
    }
  }
  self.hilight_json_written = false
end
```

## 5.3 Block

`hilight_block_setup` CDR:hilight_block_setup(⟨*tags clist var*⟩)

Records the contents of the ⟨*tags clist var*⟩ LaTeX variable to prepare block hilighting.

```
local function hilight_block_setup(self, tags_clist_var)
  local tags_clist = assert(token.get_macro(assert(tags_clist_var)))
  self['.tags clist'] = tags_clist
  self['.lines'] = {}
  self['.arguments'] = {
    __cls__ = 'Arguments',
    cache   = JSON_boolean_false,
    debug   = JSON_boolean_false,
    source  = nil,
    pygopts = {
      __cls__ = 'PygOpts',
      lang = 'tex',
      style = 'default',
      texcomments  = JSON_boolean_false,
      mathescape   = JSON_boolean_false,
      escapeinside = '',
    },
    texopts = {
      __cls__ = 'TeXOpts',
      tags    = tags_clist,
      is_inline = JSON_boolean_false,
      pyg_sty_p = '',
    },
    fv_opts = {
      __cls__ = 'FVOpts',
      firstnumber = 1,
      stepnumber  = 1,
    }
  }
  self.hilight_json_written = false
end
```

`record_line` CDR:record_line(⟨*line variable name*⟩)

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

```
local function record_line(self, line_variable_name)
  local line = assert(token.get_macro(assert(line_variable_name)))
  local ll = assert(self['.lines'])
  ll[#ll+1] = line
end
```

hilight_block_teardown `CDR:hilight_block_teardown()`

Records the contents of the ⟨*tags clist var*⟩ LaTeX variable to prepare block hilighting.

```
local function hilight_block_teardown(self)
  local ll = assert(self['.lines'])
  if #ll > 0 then
    local records = self['.records'] or {}
    self['.records'] = records
    local t = {
      already = {},
      code = table.concat(ll,'\n')
    }
    for tag in self['.tags clist']:gmatch('([^,]+)') do
      local tt = records[tag] or {}
      records[tag] = tt
      tt[#tt+1] = t
    end
  end
end
```

## 6 Exportation

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

export_file `CDR:export_file(⟨file name var⟩)`

This is called at export time. ⟨*file name var*⟩ is the name of an `str` variable containing the file name.

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

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

export_complete `CDR:export_complete()`

This is called at export time.

```
local function export_complete(self)
  local name    = self['.name']
  local export  = self['.export']
  local records = self['.records']
  local raw = export.raw == 'true'
  local tt = {}
  local s
  if not raw then
    s = export.preamble
    if s and #s>0  then
      tt[#tt+1] = s
    end
  end
  for tag in string.gmatch(export.tags, '([^,]+)') do
    local Rs = records[tag]
    if Rs then
      for _,R in ipairs(Rs) do
        if not R.already[name] or not once then
          tt[#tt+1] = R.code
        end
        if once then
          R.already[name] = true
        end
      end
    end
  end
  if not raw then
    s = export.postamble
    if s and #s>0  then
      tt[#tt+1] = s
    end
  end
  if #tt>0 then
    local fh = assert(io.open(name,'w'))
    fh:write(table.concat(tt, '\n'))
    fh:close()
  end
  self['.name'] = nil
  self['.export'] = nil
end
```

## 7 Caching

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

cache_clean_all
cache_record
cache_clean_unused

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

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

```
local function cache_clean_all(self)
  local to_remove = {}
  for f in lfs.dir(self.dir_p) do
    to_remove[f] = true
  end
  for k,_ in pairs(to_remove) do
    os.remove(self.dir_p .. k)
  end
end
local function cache_record(self, pyg_sty_p, pyg_tex_p)
  if pyg_sty_p then
    self['.style_set']  [pyg_sty_p] = true
  end
  if pyg_tex_p then
    self['.colored_set'][pyg_tex_p] = true
  end
end
local function cache_clean_unused(self)
  local to_remove = {}
  for f in lfs.dir(self.dir_p) do
    f = self.dir_p .. f
    if not self['.style_set'][f] and not self['.colored_set'][f] then
      to_remove[f] = true
    end
  end
  for f,_ in pairs(to_remove) do
    os.remove(f)
  end
end
```

_DESCRIPTION Short text description of the module.

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

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

## 8 Return the module

```
return {
```

Known fields are

```
  _DESCRIPTION       = _DESCRIPTION,
```

**`_VERSION`** to store ⟨*version string*⟩,

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

**`date`** to store ⟨*date string*⟩,

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

**Various paths** ,

```
  CDR_PY_PATH        = CDR_PY_PATH,
  set_python_path    = set_python_path,
```

**`is_truthy`**

```
  is_truthy          = is_truthy,
```

**`escape`**

```
  escape             = escape,
```

**`make_directory`**

```
  make_directory     = make_directory,
```

**`load_exec`**

```
  load_exec          = load_exec,
```

```
  load_exec_output   = load_exec_output,
```

**`record_line`**

```
  record_line        = record_line,
```

**`hilight common`**

```
  hilight_set        = hilight_set,
  hilight_set_var    = hilight_set_var,
  hilight_source     = hilight_source,
```

**`hilight code`**

```
  hilight_code_setup = hilight_code_setup,
```

**`hilight_block_setup`**

```
  hilight_block_setup    = hilight_block_setup,
  hilight_block_teardown = hilight_block_teardown,
```

`cache`

```
  cache_clean_all    = cache_clean_all,
  cache_record       = cache_record,
  cache_clean_unused = cache_clean_unused,
```

`Internals`

```
  ['.style_set']     = {},
  ['.colored_set']   = {},
  ['.options']       = {},
  ['.export']        = {},
  ['.name']          = nil,
```

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

```
  already            = false,
```

`Other`

```
  dir_p              = dir_p,
  json_p             = json_p,
```

`Exportation`

```
  export_file        = export_file,
  export_file_info   = export_file_info,
  export_complete    = export_complete,
```

```
}
```

```
%</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.

```
%<*py>
#! /usr/bin/env python3
# -*- coding: utf-8 -*-
%</py>
```

## 1 Usage

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

## 2 Header and global declarations

```
%<*py>
__version__ = '0.10'
__YEAR__  = '2022'
__docformat__ = 'restructuredtext'

import sys
import os
import argparse
import re
from pathlib import Path
import json
from pygments import highlight as hilight
from pygments.formatters.latex import LatexEmbeddedLexer, LatexFormatter
from pygments.lexers import get_lexer_by_name
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.

```
class BaseOpts(object):
```

```
  def __init__(self, d={}):
    for k, v in d.items():
      setattr(self, k, v)
```

### 3.1 `TeXOpts` class

```
class TeXOpts(BaseOpts):
  tags      = ''
  is_inline  = True
  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

```
  sty_template=r'''% !TeX root=...
\makeatletter
\CDR@StyleDefine{<placeholder:style_name>} {%
  <placeholder:style_defs>}%
\makeatother'''
  def __init__(self, *args, **kvargs):
    super().__init__(*args, **kvargs)
    self.pyg_sty_p = Path(self.pyg_sty_p or '')
```

### 3.2 `PygOpts`class

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.

```
class PygOpts(BaseOpts):
  style = 'default'
  nobackground = False
  linenos = False
  linenostart = 1
  linenostep = 1
  commandprefix = 'Py'
  texcomments = False
  mathescape =  False
  escapeinside = ""
  envname = 'Verbatim'
  lang = 'tex'
  def __init__(self, *args, **kvargs):
    super().__init__(*args, **kvargs)
    self.linenostart = abs(int(self.linenostart))
    self.linenostep  = abs(int(self.linenostep))
```

## 3.3 FVclass

```
class FVOpts(BaseOpts):
  gobble = 0
  tabsize = 4
  linenosep = '0pt'
  commentchar = ''
  frame = 'none'
  framerule = '0.4pt',
  framesep = r'\fboxsep',
  rulecolor = 'black',
  fillcolor = '',
  label = ''
  labelposition = 'none'
  numbers = 'left'
  numbersep = '1ex'
  firstnumber = 'auto'
  stepnumber = 1
  numberblanklines = True
  firstline = ''
  lastline = ''
  baselinestretch = 'auto'
  resetmargins = True
  xleftmargin = '0pt'
  xrightmargin = '0pt'
  hfuzz = '2pt'
  vspace = r'\topsep'
  samepage = False
  def __init__(self, *args, **kvargs):
    super().__init__(*args, **kvargs)
    self.gobble  = abs(int(self.gobble))
    self.tabsize = abs(int(self.tabsize))
    if self.firstnumber != 'auto':
      self.firstnumber = abs(int(self.firstnumber))
    self.stepnumber = abs(int(self.stepnumber))
```

## 3.4 Argumentsclass

```
class Arguments(BaseOpts):
  cache  = False
  debug  = False
  source = ""
  style  = "default"
  json   = ""
  directory = "."
  texopts = TeXOpts()
  pygopts = PygOpts()
  fv_opts = FVOpts()
```

# 4 Controller main class

```
class Controller:
```

## 4.1 Static methods

`object_hook` Helper for json parsing.

```
  @staticmethod
  def object_hook(d):
    __cls__ = d.get('__cls__', 'Arguments')
    if __cls__ == 'PygOpts':
      return PygOpts(d)
    elif __cls__ == 'FVOpts':
      return FVOpts(d)
    elif __cls__ == 'TeXOpts':
      return TeXOpts(d)
    elif __cls__ == 'BooleanTrue':
      return True
    elif __cls__ == 'BooleanFalse':
      return False
    else:
      return Arguments(d)
```

`lua_command`
`lua_command_now`
`lua_debug`

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

Wraps the given command between markers. It will be in the output of the coder-tool.py, further captured by coder-util.lua and either forwarded to TEX ot executed synchronously.

```
  @staticmethod
  def lua_command(cmd):
    print(f'<<<<<*LUA:{cmd}>>>>>')
  @staticmethod
  def lua_command_now(cmd):
    print(f'<<<<<!LUA:{cmd}>>>>>')
  @staticmethod
  def lua_debug(msg):
    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.

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

```
  _json_p = None
  @property
  def json_p(self):
    p = self._json_p
    if p:
      return p
    else:
      p = self.arguments.json
      if p:
        p = Path(p).resolve()
    self._json_p = p
    return p
```

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

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

```
  @property
  def parser(self):
    parser = argparse.ArgumentParser(
      prog=sys.argv[0],
      description='''
Writes to the output file a set of LaTeX macros describing
the syntax hilighting of the input file as given by pygments.
'''
    )
    parser.add_argument(
      "-v", "--version",
      help="Print the version and exit",
      action='version',
      version=f'coder-tool version {__version__},'
      ' (c) {__YEAR__} by Jérôme LAURENS.'
    )
    parser.add_argument(
      "--debug",
      action='store_true',
      default=None,
      help="display informations useful for debugging"
    )
    parser.add_argument(
      "--create_style",
      action='store_true',
      default=None,
      help="create the style definitions"
    )
    parser.add_argument(
      "--base",
      action='store',
      default=None,
      help="the path of the file to be colored, with no extension"
    )
    parser.add_argument(
      "json",
      metavar="<json data file>",
      help="""
file name with extension, contains processing information.
"""
    )
    return parser

```

## 4.3 Methods

### 4.3.1 __init__

`__init__` Constructor. Reads the command line arguments.

```
  def __init__(self, argv = sys.argv):
    argv = argv[1:] if re.match(".*coder\-tool\.py$", argv[0]) else argv
    ns = self.parser.parse_args(
      argv if len(argv) else ['-h']
    )
    with open(ns.json, 'r') as f:
      self.arguments = json.load(
        f,
        object_hook = Controller.object_hook
      )
    args = self.arguments
    args.json = ns.json
    self.texopts = args.texopts
    pygopts = self.pygopts = args.pygopts
    fv_opts = self.fv_opts = args.fv_opts
    self.formatter = LatexFormatter(
      style = pygopts.style,
      nobackground = pygopts.nobackground,
      commandprefix = pygopts.commandprefix,
      texcomments = pygopts.texcomments,
      mathescape = pygopts.mathescape,
      escapeinside = pygopts.escapeinside,
      envname = 'CDR@Pyg@Verbatim',
    )

    try:
      lexer = self.lexer = get_lexer_by_name(pygopts.lang)
    except ClassNotFound as err:
      sys.stderr.write('Error: ')
      sys.stderr.write(str(err))

    escapeinside = pygopts.escapeinside
    # When using the LaTeX formatter and the option `escapeinside` is
    # specified, we need a special lexer which collects escaped text
    # before running the chosen language lexer.
    if len(escapeinside) == 2:
      left  = escapeinside[0]
      right = escapeinside[1]
      lexer = self.lexer = LatexEmbeddedLexer(left, right, lexer)

    gobble = fv_opts.gobble
    if gobble:
      lexer.add_filter('gobble', n=gobble)
    tabsize = fv_opts.tabsize
    if tabsize:
      lexer.tabsize = tabsize
    lexer.encoding = ''
    args.base = ns.base
    args.create_style = ns.create_style
    if ns.debug:
      args.debug = True
    # IN PROGRESS: support for extra keywords
    # EXTRA_KEYWORDS = set(('foo', 'bar', 'foobar', 'barfoo', 'spam', 'eggs'))
    # def over(self, text):
    #   for index, token, value in lexer.__class__.get_tokens_unprocessed(self, text):
    #     if token is Name and value in EXTRA_KEYWORDS:
    #       yield index, Keyword.Pseudo, value
    #   else:
    #       yield index, token, value
    # lexer.get_tokens_unprocessed = over.__get__(lexer)

```

### 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.

```
  def create_style(self):
    args = self.arguments
    if not args.create_style:
      return
    texopts = args.texopts
    pyg_sty_p = texopts.pyg_sty_p
    if args.cache and pyg_sty_p.exists():
      return
    texopts = self.texopts
    style = self.pygopts.style
    formatter = self.formatter
    style_defs = formatter.get_style_defs() \
      .replace(r'\makeatletter', '') \
      .replace(r'\makeatother', '') \
      .replace('\n', '%\n')
    sty = self.texopts.sty_template.replace(
      '<placeholder:style_name>',
      style,
    ).replace(
      '<placeholder:style_defs>',
      style_defs,
    ).replace(
      '{}%',
      '{%}\n}%{'
    ).replace(
      '[}%',
      '[%]\n}%'
    ).replace(
      '{]}%',
      '{%[\n]}%'
    )
    with pyg_sty_p.open(mode='w',encoding='utf-8') as f:
      f.write(sty)
    if args.debug:
      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 hilighted by pygments.

```
  def pygmentize(self, source):
    source = hilight(source, self.lexer, self.formatter)
    m = re.match(
      r'\\begin{CDR@Pyg@Verbatim}.*?\n(.*?)\n\\end{CDR@Pyg@Verbatim}\s*\Z',
      source,
      flags=re.S
    )
    assert(m)
    hilighted = m.group(1)
    texopts = self.texopts
    if texopts.is_inline:
      return hilighted.replace(' ', r'\CDR@Sp ')+r'\ignorespaces'
    lines = hilighted.split('\n')
    ans_code = []
    last = 1
    for line in lines[1:]:
      last += 1
      ans_code.append(rf'''\CDR@Line{{{last}}}{{{line}}}''')
    if len(lines):
      ans_code.insert(0, rf'''\CDR@Line[last={last}]{{{1}}}{{{lines[0]}}}''')
    hilighted = '\n'.join(ans_code)
    return hilighted
```

### 4.3.4 `create_pygmented`

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

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

```
  def create_pygmented(self):
    args = self.arguments
    base = args.base
    if not base:
      return False
    source = args.source
    if not source:
      tex_p = Path(base).with_suffix('.tex')
      with open(tex_p, 'r') as f:
        source = f.read()
    pyg_tex_p = Path(base).with_suffix('.pyg.tex')
    hilighted = self.pygmentize(source)
    with pyg_tex_p.open(mode='w',encoding='utf-8') as f:
      f.write(hilighted)
    if args.debug:
      print('HILIGHTED', os.path.relpath(pyg_tex_p))
```

## 4.4 Main entry

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

# File III
# coder.sty implementation

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

## 1 Setup

### 1.1 Utilities

\CDR_set_conditional:Nn

`\CDR_set_conditional:Nn` ⟨*core name*⟩ {⟨*condition*⟩}

Wrapper over `\prg_set_conditional:Nnn`.

```
\cs_new:Npn \CDR_set_conditional:Nn #1 #2 {
  \bool_if:nTF { #2 } {
    \prg_set_conditional:Nnn #1 { p, T, F, TF } { \prg_return_true: }
  } {
    \prg_set_conditional:Nnn #1 { p, T, F, TF } { \prg_return_false: }
  }
}
```

\CDR_set_conditional_alt:Nn

`\CDR_set_conditional_alt:Nnnn` ⟨*core name*⟩ {⟨*condition*⟩}

Wrapper over `\prg_set_conditional:Nnn`.

```
\cs_new:Npn \CDR_set_conditional_alt:Nn #1 #2 {
  \prg_set_conditional:Nnn #1 { p, T, F, TF } {
    \bool_if:nTF { #2 } { \prg_return_true: } { \prg_return_false: }
  }
}
```

\CDR_has_pygments_p: ⋆
\CDR_has_pygments:*TF* ⋆

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

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

```
\prg_new_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } {
  \PackageError { coder } { Internal~error(pygments~path) } { Please~report~error }
}

\cs_new:Npn \CDR_pygments_setup:n #1 {
  \CDR_set_conditional:Nn \CDR_has_pygments: {
    \str_if_eq_p:nn { #1 } { true }
  }
}
\lua_now:n { CDR = require("coder-util") }
\exp_args:Nx \CDR_pygments_setup:n {
  \lua_now:n { CDR:set_python_path() }
}
\cs_new:Npn \CDR_pygments_setup: {
  \sys_get_shell:nnNTF {which~pygmentize} { \cc_select:N \c_str_cctab } \l_CDR_tl {
    \tl_if_in:NnTF \l_CDR_tl { pygmentize } {
      \prg_set_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } {
        \prg_return_true:
      }
    } {
      \prg_set_conditional:Nnn \CDR_has_pygments: { p, T, F, TF } {
        \prg_return_false:
      }
    }
  } {
    \typeout {Shell~escape~is~not~available}
  }
}
```

```
\NewDocumentCommand \CDRTest {} {
  \par\noindent
  Path~to~\textsf{python}:~\texttt{\directlua{tex.print(CDR.PYTHON_PATH)}}
  \par\noindent
  Path~to~\textsf{pygmentize}:~\texttt{\directlua{tex.print(CDR.PYGMENTIZE_PATH)}}
  \par\noindent
  \CDR_has_pygments:TF { Pygments~is~available } { Pygments~is~not~available
}:~%\CDRCode[lang=tex]|\textit{text}|
  \par\noindent
}
```

## 2 Messages

```
\msg_new:nnn { coder } { unknown-choice } {
  #1~given~value~‘#3’~not~in~#2
}
```

## 3 Constants

`\c_CDR_tag`
`\c_CDR_Tags`

Paths of L3keys modules.
These are root path components used throughout the pakage. The latter is a subpath of the former.

```
\str_const:Nn \c_CDR_Tags { CDR@Tags }
\str_const:Nx \c_CDR_tag { \c_CDR_Tags / tag }
```

(*End definition for* `\c_CDR_tag` *and* `\c_CDR_Tags`*. These variables are documented on page* **??**.)

`\c_CDR_tag_get` Root identifier for tag properties, used throughout the pakage.

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

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

## 4 Implementation details

As far as possible, macro making assignments to variables are protected. All variables following expl3 naming conventions are implementation details and therefore must be considered private.

Many functions have useful hooks for debugging or testing.

\CDR@Debug

`\CDR@Debug {⟨argument⟩}`

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

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

## 5 Variables

### 5.1 Internal scratch variables

These local variables are used in a very limited scope.

`\l_CDR_bool` Local scratch variable.

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

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

`\l_CDR_tl` Local scratch variable.

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

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

`\l_CDR_str` Local scratch variable.

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

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

`\l_CDR_seq` Local scratch variable.

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

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

`\l_CDR_prop` Local scratch variable.

```
\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.

```
\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

```
\ior_new:N \l_CDR_ior
```

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

`\l_CDR_iow` Output file identifier

```
\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.

```
\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.

```
\prop_new:N \g_CDR_source_prop
```

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

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

```
\tl_new:N \g_CDR_chunks_tl
\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.

```
\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.

```
\tl_new:N \g_CDR_hook_tl
```

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

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

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

## 5.4 Local variables

`\l_CDR_kv_clist` keyval storage.

```
\clist_new:N \l_CDR_kv_clist
```

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

`\l_CDR_opts_tl` options storage.

```
\tl_new:N \l_CDR_opts_tl
```

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

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

```
\tl_new:N \l_CDR_recorded_tl
```

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

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

```
\tl_new:N \l_CDR_count_tl
```

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

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

```
\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.

```
\tl_new:N \l_CDR_line_tl
```

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

`\l_CDR_lineno_tl` Token list for lineno display.

```
\tl_new:N \l_CDR_lineno_tl
```

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

`\l_CDR_name_tl` Token list for chunk name display.

```
\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.

```
\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`⟩.

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

`default` Generic and named line number counter.

`__n` `__line`

```
\CDR_int_new:cn { default } { 1 }
\CDR_int_new:cn { __n } { 1 }
\CDR_int_new:cn { __i } { 1 }
\CDR_int_new:cn { __line } { 1 }
```

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

`\CDR_int:c` ⋆

`\CDR_int:c {`⟨*tag name*⟩`}`

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

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

`\CDR_int_use:c` ⋆

`\CDR_int_use:n {`⟨*tag name*⟩`}`

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

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

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

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

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

```
\prg_new_conditional:Nnn \CDR_int_if_exist:c { p, T, F, TF } {
  \int_if_exist:cTF { CDR@int.#1 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

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

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

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

```
\prg_new_conditional:Nnn \CDR_int_compare:cNn { p, T, F, TF } {
  \int_compare:nNnTF { \CDR_int:c { #1 } } #2 { #3 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

`\CDR_int_set:cn`
`\CDR_int_gset:cn`

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

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

```
\cs_new:Npn \CDR_int_set:cn #1 #2 {
  \int_set:cn { CDR@int.#1 } { #2 }
}
\cs_new:Npn \CDR_int_gset:cn #1 #2 {
  \int_gset:cn { CDR@int.#1 } { #2 }
}
```

`\CDR_int_set:cc`
`\CDR_int_gset:cc`

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

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

```
\cs_new:Npn \CDR_int_set:cc #1 #2 {
  \CDR_int_set:cn { #1 } { \CDR_int:c { #2 } }
}
\cs_new:Npn \CDR_int_gset:cc #1 #2 {
  \CDR_int_gset:cn { #1 } { \CDR_int:c { #2 } }
}
```

`\CDR_int_add:cn`
`\CDR_int_gadd:cn`

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

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

```
\cs_new:Npn \CDR_int_add:cn #1 #2 {
  \int_add:cn { CDR@int.#1 } { #2 }
}
\cs_new:Npn \CDR_int_gadd:cn #1 #2 {
  \int_gadd:cn { CDR@int.#1 } { #2 }
}
```

`\CDR_int_add:cc`
`\CDR_int_gadd:cc`

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

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

```
\cs_new:Npn \CDR_int_add:cc #1 #2 {
  \CDR_int_add:cn { #1 } { \CDR_int:c { #2 } }
}
\cs_new:Npn \CDR_int_gadd:cc #1 #2 {
  \CDR_int_gadd:cn { #1 } { \CDR_int:c { #2 } }
}
```

`\CDR_int_sub:cn`
`\CDR_int_gsub:cn`

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

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

```
\cs_new:Npn \CDR_int_sub:cn #1 #2 {
  \int_sub:cn { CDR@int.#1 } { #2 }
}
\cs_new:Npn \CDR_int_gsub:cn #1 #2 {
  \int_gsub:cn { CDR@int.#1 } { #2 }
}
```

## 5.6 Utilities

`\g_CDR_tags_clist`
`\g_CDR_all_tags_clist`
`\g_CDR_last_tags_clist`

Store the current list of tags used by `\CDRCode` and the `CDRBlock` environment, or declared by `\CDRExport`. All the tags are recorded, if there is an only one, it is not shown in block code chunks. The `\g_CDR_last_tags_clist` variable contains the last list of tags that was displayed.

```
\clist_new:N \g_CDR_tags_clist
\clist_new:N \g_CDR_all_tags_clist
\clist_new:N \g_CDR_last_tags_clist
\AddToHook { shipout/before } {
  \clist_gclear:N \g_CDR_last_tags_clist
}
```

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

```
\prg_new_conditional:Nnn \CDR_clist_if_eq:NN { p, T, F, TF } {
  \tl_if_eq:NNTF #1 #2 {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

# 6 Tag properties

The tag properties concern the code chunks. They are set from different paths, such that `\l_keys_path_str` must be properly parsed for that purpose. Commands in this section and the next ones contain `CDR_tag`.

The ⟨*tag names*⟩ starting with a double underscore are reserved by the package.

## 6.1 Helpers

`\CDR_tag_get_path:cc` ⋆
`\CDR_tag_get_path:c` ⋆

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

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

```
\cs_new:Npn \CDR_tag_get_path:cc #1 #2 {
  \c_CDR_tag_get @ #1 / #2
}
\cs_new:Npn \CDR_tag_get_path:c {
  \CDR_tag_get_path:cc { __local }
}
```

## 6.2 Set

\CDR_tag_set:ccn
\CDR_tag_set:ccV

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

Store ⟨*value*⟩, which is further retrieved with the instruction `\CDR_tag_get:cc {⟨tag name⟩} {⟨relative key path⟩}`. Only ⟨*tag name*⟩ and ⟨*relative key path*⟩ containing no `@` character are supported. All the affectations are made at the current TEX group level. *Nota Bene:* `\cs_generate_variant:Nn` is buggy when there is a 'c' argument.

```
\cs_new_protected:Npn \CDR_tag_set:ccn #1 #2 #3 {
  \cs_set:cpn { \CDR_tag_get_path:cc { #1 } { #2 } } { \exp_not:n { #3 } }
}
\cs_new_protected:Npn \CDR_tag_set:ccV #1 #2 #3 {
  \exp_args:NnnV
  \CDR_tag_set:ccn { #1 } { #2 } #3
}
```

\c_CDR_tag_regex To parse a l3keys full key path.

```
\tl_set:Nn \l_CDR_tl { /([^/]*)/(.*)$ } \use_none:n { $ }
\tl_put_left:NV \l_CDR_tl \c_CDR_tag
\tl_put_left:Nn \l_CDR_tl { ^ }
\exp_args:NNV
\regex_const:Nn \c_CDR_tag_regex \l_CDR_tl
```

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

\CDR_tag_set:n

`\CDR_tag_set:n {⟨value⟩}`

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

```
\cs_new_protected:Npn \CDR_tag_set:n {
  \exp_args:NnV
  \regex_extract_once:NnNTF \c_CDR_tag_regex
      \l_keys_path_str \l_CDR_seq {
    \CDR_tag_set:ccn
      { \seq_item:Nn \l_CDR_seq 2 }
      { \seq_item:Nn \l_CDR_seq 3 }
  } {
    \PackageWarning
      { coder }
      { Unexpected~key~path~'\l_keys_path_str' }
    \use_none:n
  }
}
```

\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.

```
\cs_new_protected:Npn \CDR_tag_set: {
  \exp_args:NV
  \CDR_tag_set:n \l_keys_value_tl
}
```

`\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.

```
\cs_new:Npn \CDR_tag_set:cn #1 {
  \exp_args:NnV
  \regex_extract_once:NnNTF \c_CDR_tag_regex
      \l_keys_path_str \l_CDR_seq {
    \CDR_tag_set:ccn
      { \seq_item:Nn \l_CDR_seq 2 }
      { #1 }
  } {
    \PackageWarning
      { coder }
      { Unexpected~key~path~`\l_keys_path_str' }
    \use_none:n
  }
}
```

`\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.

```
\prg_generate_conditional_variant:Nnn \str_if_eq:nn { Vn } { p, T, F, TF }

\regex_const:Nn \c_CDR_root_regex { ^(.*)/.*$ } \use_none:n { $ }
\cs_new:Npn \CDR_tag_choices: {
  \str_if_eq:nnT \l_keys_key_tl \l_keys_choice_tl {
    \exp_args:NnV
    \regex_extract_once:NnNT \c_CDR_root_regex
        \l_keys_path_str \l_CDR_seq {
      \str_set:Nx \l_keys_path_str {
        \seq_item:Nn \l_CDR_seq 2
      }
    }
  }
}
```

`\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.

```
\cs_new_protected:Npn \CDR_tag_choices_set: {
  \CDR_tag_choices:
  \exp_args:NV
  \CDR_tag_set:n \l_keys_choice_tl
}
```

`\CDR_if_tag_truthy_p:cc` ⋆
`\CDR_if_tag_truthy:cc`*TF* ⋆
`\CDR_if_tag_truthy_p:c` ⋆
`\CDR_if_tag_truthy:c`*TF* ⋆

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

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`.

```
\prg_new_conditional:Nnn \CDR_if_tag_truthy:cc { p, T,  F, TF } {
  \exp_args:Ne
  \str_compare:nNnTF {
    \exp_args:Ne \str_lowercase:n { \CDR_tag_get:cc { #1 } { #2 } }
  } = { true } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
\prg_new_conditional:Nnn \CDR_if_tag_truthy:c { p, T,  F, TF } {
  \exp_args:Ne
  \str_compare:nNnTF {
    \exp_args:Ne \str_lowercase:n { \CDR_tag_get:c { #1 } }
  } = { true } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

`\CDR_if_tag_eq_p:ccn` ⋆
`\CDR_if_tag_eq:ccn`*TF* ⋆
`\CDR_if_tag_eq_p:cn` ⋆
`\CDR_if_tag_eq:cn`*TF* ⋆

`\CDR_if_tag_eq:ccnTF {⟨tag name⟩} {⟨relative key path⟩} {⟨value⟩} {⟨true code⟩} {⟨false code⟩}`
`\CDR_if_tag_eq:cnTF {⟨relative key path⟩} {⟨value⟩} {⟨true code⟩} {⟨false code⟩}`

Execute ⟨*true code*⟩ when the property for ⟨*tag name*⟩ and ⟨*relative key path*⟩ is equal to {⟨*value*⟩}, ⟨*false code*⟩ otherwise. The comparison is based on `\str_compare:...`. In the second version, the ⟨*tag name*⟩ is not provided and set to `__local`.

```
\prg_new_conditional:Nnn \CDR_if_tag_eq:ccn { p, T,  F, TF } {
  \exp_args:Nf
  \str_compare:nNnTF { \CDR_tag_get:cc { #1 } { #2 } } = { #3 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
\prg_new_conditional:Nnn \CDR_if_tag_eq:cn { p, T,  F, TF } {
  \exp_args:Nf
  \str_compare:nNnTF { \CDR_tag_get:cc { __local } { #1 } } = { #2 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

\CDR_if_truthy_p:n ⋆
\CDR_if_truthy:n*TF* ⋆

`\CDR_if_truthy:nTF {⟨token list⟩} {⟨true code⟩} {⟨false code⟩}`

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".

```
\prg_new_conditional:Nnn \CDR_if_truthy:n { p, T,  F, TF } {
  \exp_args:Ne
  \str_compare:nNnTF { \exp_args:Ne \str_lowercase:n { #1 } } = { true } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

\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.

```
\cs_new_protected:Npn \CDR_tag_boolean_set:n #1 {
  \CDR_if_truthy:nTF { #1 } {
    \CDR_tag_set:n { true }
  } {
    \CDR_tag_set:n { false }
  }
}
\cs_generate_variant:Nn \CDR_tag_boolean_set:n { x }
```

## 6.3 Retrieving tag properties

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

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

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

For text block code chunks, this module inherits from

1. `\c_CDR_tag_get/`⟨*name*$_1$⟩, ..., `\c_CDR_tag_get/`⟨*name*$_n$⟩ for each tag name of the ordered tags list
2. `\c_CDR_tag_get/default.block`
3. `\c_CDR_tag_get/default`
4. `\c_CDR_tag_get/__pygments`
5. `\c_CDR_tag_get/__pygments.block`
6. `\c_CDR_tag_get/__fancyvrb`
7. `\c_CDR_tag_get/__fancyvrb.block`
8. `\c_CDR_tag_get/__fancyvrb.all` when no using pygments

`\CDR_if_tag_exist_here_p:cc` ⋆
`\CDR_if_tag_exist_here:cc`*TF* ⋆

`\CDR_if_tag_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.

```
\prg_new_conditional:Nnn \CDR_if_tag_exist_here:cc { p, T, F, TF } {
  \cs_if_exist:cTF { \CDR_tag_get_path:cc { #1 } { #2 } } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

`\CDR_if_tag_exist_p:cc` ⋆
`\CDR_if_tag_exist:cc`*TF* ⋆
`\CDR_if_tag_exist_p:c` ⋆
`\CDR_if_tag_exist:c`*TF* ⋆

`\CDR_if_tag_exist:ccTF {`⟨*tag name*⟩`}` ⟨*relative key path*⟩ `{`⟨*true code*⟩`} {`⟨*false code*⟩`}`
`\CDR_if_tag_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`.

```
\prg_new_conditional:Nnn \CDR_if_tag_exist:cc { p, T, F, TF } {
  \cs_if_exist:cTF { \CDR_tag_get_path:cc { #1 } { #2 } } {
    \prg_return_true:
  } {
    \seq_if_exist:cTF { \CDR_tag_parent_seq:c { #1 } } {
      \seq_map_tokens:cn
        { \CDR_tag_parent_seq:c { #1 } }
        { \CDR_if_tag_exist_f:cn { #2 } }
    } {
      \prg_return_false:
    }
  }
}
\prg_new_conditional:Nnn \CDR_if_tag_exist:c { p, T, F, TF } {
  \cs_if_exist:cTF { \CDR_tag_get_path:c { #1 } } {
    \prg_return_true:
  } {
    \seq_if_exist:cTF { \CDR_tag_parent_seq:c { __local } } {
      \seq_map_tokens:cn
        { \CDR_tag_parent_seq:c { __local } }
        { \CDR_if_tag_exist_f:cn { #1 } }
    } {
      \prg_return_false:
    }
  }
}
\cs_new:Npn \CDR_if_tag_exist_f:cn #1 #2 {
  \quark_if_no_value:nTF { #2 } {
    \seq_map_break:n {
      \prg_return_false:
    }
  } {
    \CDR_if_tag_exist:ccT { #2 } { #1 } {
      \seq_map_break:n {
        \prg_return_true:
      }
    }
  }
}
```

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

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

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

```
\cs_new:Npn \CDR_tag_get:cc #1 #2 {
  \CDR_if_tag_exist_here:ccTF { #1 } { #2 } {
    \use:c { \CDR_tag_get_path:cc { #1 } { #2 } }
  } {
    \seq_if_exist:cT { \CDR_tag_parent_seq:c { #1 } } {
      \seq_map_tokens:cn
        { \CDR_tag_parent_seq:c { #1 } }
        { \CDR_tag_get_f:cn { #2 } }
    }
  }
}
\cs_new:Npn \CDR_tag_get_f:cn #1 #2 {
  \quark_if_no_value:nF { #2 } {
    \CDR_if_tag_exist_here:ccT { #2 } { #1 } {
      \seq_map_break:n {
        \use:c { \CDR_tag_get_path:cc { #2 } { #1 } }
      }
    }
  }
}
\cs_new:Npn \CDR_tag_get:c {
  \CDR_tag_get:cc { __local }
}
```

\CDR_tag_get:ccN
\CDR_tag_get:cN

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

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

```
\cs_new_protected:Npn \CDR_tag_get:ccN #1 #2 #3 {
  \tl_set:Nf #3 { \CDR_tag_get:cc { #1 } { #2 } }
}
\cs_new_protected:Npn \CDR_tag_get:cN {
  \CDR_tag_get:ccN { __local }
}
```

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

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

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

```
\prg_new_protected_conditional:Nnn \CDR_tag_get:ccN { T, F, TF } {
  \CDR_if_tag_exist:ccTF { #1 } { #2 } {
    \CDR_tag_get:ccN { #1 } { #2 } #3
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
\prg_new_protected_conditional:Nnn \CDR_tag_get:cN { T, F, TF } {
  \CDR_if_tag_exist:cTF { #1 } {
    \CDR_tag_get:cN { #1 } #2
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

### 6.4 Inheritance

When a child inherits from a parent, all the keys of the parent that are not inherited are made available to the child (inheritance does not jump over generations).

\CDR_tag_parent_seq:c ⋆

\CDR_tag_parent_seq:c {⟨*tag name*⟩}

Return the name of the sequence variable containing the list of the parents. Each child has its own sequence of parents assigned locally.

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

\CDR_get_inherit:cn
\CDR_get_inherit:cf
\CDR_get_inherit:n
\CDR_get_inherit:f

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

Set the parents of ⟨*child name*⟩ to the given list. When the ⟨*child name*⟩ is not provided, it defaults to `__local`.

```
\cs_new:Npn \CDR_get_inherit:cn #1 #2 {
  \seq_set_from_clist:cn { \CDR_tag_parent_seq:c { #1 } } { #2 }
  \seq_remove_duplicates:c \l_CDR_tl
  \seq_remove_all:cn \l_CDR_tl {}
  \seq_put_right:cn \l_CDR_tl { \q_no_value }
}
\cs_new:Npn \CDR_get_inherit:cf {
  \exp_args:Nnf \CDR_get_inherit:cn
}
\cs_new:Npn \CDR_tag_parents:c #1 {
  \seq_map_inline:cn { \CDR_tag_parent_seq:c { #1 } } {
    \quark_if_no_value:nF { ##1 } {
      ##1,
    }
  }
}
\cs_new:Npn \CDR_get_inherit:n {
  \CDR_get_inherit:cn { __local }
}
\cs_new:Npn \CDR_get_inherit:f {
  \CDR_get_inherit:cf { __local }
}
```

## 7 Cache management

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

```
\AddToHook { begindocument/before } {
  \IfFileExists {./\jobname.aux} {} {
   \lua_now:n {CDR:cache_clean_all()}
  }
}
```

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

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

## 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*⟩.

```
\cs_new:Npn \CDR_clist_map_inline:Nnn #1 #2 {
  \clist_if_empty:NTF #1 {
    #2
    \use_none:n
  } {
    \clist_map_inline:Nn #1
  }
}
```

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

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

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

```
\prg_new_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
  \PackageError
    { coder }
    { Conditional~not~available }
    { Internal~error:~report~bug }
}
```

`\CDR_process_record:`

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

```
\cs_new:Npn \CDR_process_record: {}
```

## 9 l3keys modules for code chunks

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

## 9.1 Utilities

\CDR_tag_module:n ⋆

\CDR_tag_module:n {⟨*module base*⟩}

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

```
\cs_set:Npn \CDR_tag_module:n #1 {
  \str_if_eq:nnTF { #1 } { .. } {
    \c_CDR_Tags
  } {
    \tl_if_empty:nTF { #1 } { \c_CDR_Tags / tag } { \c_CDR_Tags / tag / #1 }
  }
}
```

\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.

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

\CDR_tag_keys_if_exist:nn*TF* ⋆

\CDR_tag_keys_if_exist:nnTF {⟨*module base*⟩} {⟨*key*⟩} {⟨*true code*⟩} {⟨*false code*⟩}

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

```
\prg_new_conditional:Nnn \CDR_tag_keys_if_exist:nn { p, T, F, TF } {
  \exp_args:Nf
  \keys_if_exist:nnTF { \CDR_tag_module:n { #1 } } { #2 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

\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.

```
\cs_new_protected:Npn \CDR_tag_keys_set:nn #1 {
  \exp_args:Nf
  \keys_set:nn { \CDR_tag_module:n { #1 } }
}
\cs_generate_variant:Nn \CDR_tag_keys_set:nn { nV }
```

`\CDR_tag_keys_set:nn`

`\CDR_tag_keys_set:nn {⟨module base⟩} {⟨keyval list⟩}`

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

```
\cs_new_protected:Npn \CDR_local_set:n {
  \CDR_tag_keys_set:nn { __local }
}
\cs_generate_variant:Nn \CDR_local_set:n { V }
```

### 9.1.1 Handling unknown tags

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

`\CDR_tag_keys_inherit:nn`

`\CDR_tag_keys_inherit:nn {⟨tag name⟩} {⟨parents comma list⟩}`

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

```
\cs_new_protected_nopar:Npn \CDR_tag_keys_inherit__:nnn #1 #2 #3 {
  \keys_define:nn { #1 } { #2 .inherit:n = { #1 / #3 } }
}
\cs_new_protected_nopar:Npn \CDR_tag_keys_inherit_:nnn #1 #2 #3 {
  \exp_args:Nnx
  \use:n { \CDR_tag_keys_inherit__:nnn { #1 } { #2 } } {
    \clist_use:nn { #3 } { ,#1/ }
  }
}
\cs_new_protected_nopar:Npn \CDR_tag_keys_inherit:nn {
  \exp_args:Nf
  \CDR_tag_keys_inherit_:nnn { \CDR_tag_module:n { } }
}
```

`\CDR_local_inherit:n`

`Wrapper over \CDR_tag_keys_inherit:nn where ⟨tag name⟩ is given by \CDR_tag_module:n{__local}.`

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

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

`\CDR_tag_keys_set_known:nnN`
`\CDR_tag_keys_set_known:nVN`
`\CDR_tag_keys_set_known:nN`
`\CDR_tag_keys_set_known:N`

`\CDR_tag_keys_set_known:nnN {⟨tag name⟩} {⟨key[=value] items⟩} ⟨clist var⟩`
`\CDR_tag_keys_set_known:nN {⟨tag name⟩} ⟨clist var⟩`

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

```
\cs_new_protected_nopar:Npn \CDR_tag_keys_set_known__:nnN #1 #2 {
  \keys_set_known:nnnN { #1 } { #2 } { #1 }
}
\cs_new_protected_nopar:Npn \CDR_tag_keys_set_known:nnN #1 {
  \exp_args:Nf
  \CDR_tag_keys_set_known__:nnN { \CDR_tag_module:n { #1 } }
}
\cs_generate_variant:Nn \CDR_tag_keys_set_known:nnN { nV }
\cs_new_protected_nopar:Npn \CDR_tag_keys_set_known:nN #1 #2 {
  \CDR_tag_keys_set_known:nVN { #1 } #2 #2
}
```

`\CDR_tag_keys_set_known:nnN`
`\CDR_tag_keys_set_known:nVN`
`\CDR_tag_keys_set_known:nN`
`\CDR_tag_keys_set_known:N`

`\CDR_local_set_known:nN {⟨key[=value] items⟩} ⟨clist var⟩`
`\CDR_local_set_known:N ⟨clist var⟩`

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

```
\cs_new_protected_nopar:Npn \CDR_local_set_known:nN {
  \CDR_tag_keys_set_known:nnN { __local }
}
\cs_generate_variant:Nn \CDR_local_set_known:nN { V }
\cs_new_protected_nopar:Npn \CDR_local_set_known:N #1 {
  \CDR_local_set_known:VN #1 #1
}
```

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

```
\tl_set:Nn \l_CDR_tl { /([^/]*)(?:/(.*))?$ } \use_none:n { $ }
\exp_args:NNf
\tl_put_left:Nn \l_CDR_tl { \CDR_tag_module:n {} }
\tl_put_left:Nn \l_CDR_tl { ^ }
\exp_args:NNV
\regex_const:Nn \c_CDR_provide_regex \l_CDR_tl
```

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

`\@CDR@TEST`
`\CDR_tag_provide_from_kv:n`

`\CDR_tag_provide:n {⟨deep comma list⟩}`
`\CDR_tag_provide_from_kv:n {⟨key-value list⟩}`

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

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

```
\regex_const:Nn \c_CDR_engine_regex { ^[^/]+\sengine\soptions$ } \use_none:n { $ }
\cs_new_protected_nopar:Npn \CDR_tag_provide:n #1 {
\CDR@Debug { \string\CDR_tag_provide:n: #1 }
  \exp_args:NNf
  \regex_extract_once:NnNTF \c_CDR_provide_regex {
    \CDR_tag_module:n { .. } / #1
  } \l_CDR_seq {
    \tl_set:Nx \l_CDR_tl { \seq_item:Nn \l_CDR_seq 3 }
    \exp_args:Nx
    \clist_map_inline:nn {
      \seq_item:Nn \l_CDR_seq 2
    } {
      \CDR_tag_keys_if_exist:nnF { } { ##1 } {
        \CDR_tag_keys_inherit:nn { ##1 } {
          __pygments, __pygments.block,
          default.block, default.code, default, __tags, __engine,
          __fancyvrb, __fancyvrb.block, __fancyvrb.frame,
          __fancyvrb.number, __fancyvrb.all,
        }
        \CDR_tag_keys_define:nn { } {
          ##1 .code:n = \CDR_tag_keys_set:nn { ##1 } { ####1 },
          ##1 .value_required:n = true,
        }
\CDR@Debug{\string\CDR_tag_provide:n \CDR_tag_module:n {##1} = ...}
      }
      \exp_args:NnV
      \CDR_tag_keys_if_exist:nnF { ##1 } \l_CDR_tl {
        \exp_args:NNV
        \regex_match:NnT \c_CDR_engine_regex
            \l_CDR_tl {
          \exp_args:Nnf
          \CDR_tag_keys_define:nn { ##1 } {
            \use:n { \l_CDR_tl } .code:n = \CDR_tag_set:n { ####1 },
          }
          \exp_args:Nnf
          \CDR_tag_keys_define:nn { ##1 } {
            \use:n { \l_CDR_tl } .value_required:n = true,
          }
\CDR@Debug{\string\CDR_tag_provide:n: \CDR_tag_module:n { ##1 } / \l_CDR_tl = ...}
        }
      }
    }
  } {
    \regex_match:NnTF \c_CDR_engine_regex { #1 } {
      \CDR_tag_keys_define:nn { default } {
        #1 .code:n = \CDR_tag_set:n { ##1 },
        #1 .value_required:n = true,
      }
\CDR@Debug{\string\CDR_tag_provide:n.C:\CDR_tag_module:n { default } / #1 = ...}
    } {
\CDR@Debug{\string\CDR_tag_provide:n\space did~nothing~new.}
    }
  }
}
\cs_new:Npn \CDR_tag_provide:nn #1 #2 {
  \CDR_tag_provide:n { #1 }
}
\cs_new:Npn \CDR_tag_provide_from_kv:n {
  \keyval_parse:nnn {
    \CDR_tag_provide:n
  } {
    \CDR_tag_provide:nn
  }
}
\cs_generate_variant:Nn \CDR_tag_provide_from_kv:n { V }
```

### 9.2 pygments

These are pygments's `LatexFormatter` options, that are not covered by `__fancyvrb`. They are made available at the end user level, but may not be relevant when pygments is nor used.

#### 9.2.1 `__pygments` l3keys module

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

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

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

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

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

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

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

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

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

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

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

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

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

● `__initialize` Initializer.

```
  __initialize .meta:n = {
    lang = tex,
    pygments = \CDR_has_pygments:TF { true } { false },
    style = default,
    commandprefix = PY,
    mathescape = false,
    escapeinside = ,
  },
  __initialize .value_forbidden:n = true,

}
\AtBeginDocument{
  \CDR_tag_keys_set:nn { __pygments } { __initialize }
}
```

### 9.2.2 `__pygments.block` l3keys module

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

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

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

● `__initialize` Initializer.

```
  __initialize .meta:n = {
    texcomments = false,
  },
  __initialize .value_forbidden:n = true,

}
\AtBeginDocument{
  \CDR_tag_keys_set:nn { __pygments.block } { __initialize }
}
```

## 9.3 Specifc to coder

### 9.3.1 `default` l3keys module

```
\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.

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

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

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

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

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

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

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

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

**default options=⟨*default options*⟩** to specify the coder options that should apply when the default engine is selected.setup_tags

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

**⟨*engine name*⟩ engine options=⟨*engine options*⟩** to specify the options for the named engine,

**⟨*engine name*⟩ options=⟨*coder options*⟩** to specify the coder options that should apply when the named engine is selected.

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

```
  __initialize .meta:n = {
    format = ,
    cache = true,
    debug = false,
    post~processor = ,
    default~engine~options = ,
    default~options = ,
  },
  __initialize .value_forbidden:n = true,

}
\AtBeginDocument{
  \CDR_tag_keys_set:nn { default } { __initialize }
}
```

### 9.3.2 default.code l3keys module

Void for the moment.

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

Known keys include:

**mbox[=true|false]** When set to `true`, put the argument inside a LaTeX `mbox` to prevent the code chunk to spread over different lines. Initially `true`.

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

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

```
  __initialize .meta:n = {
    mbox = true,
  },
  __initialize .value_forbidden:n = true,
```

```
}
\AtBeginDocument{
  \CDR_tag_keys_set:nn { default.code } { __initialize }
}
```

### 9.3.3 __tags l3keys module

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

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

Known keys include:

**tags=⟨*comma list of tag names*⟩** to enable/disable the display of the code chunks tags. Initially `empty`.

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

```
  tags .code:n = {
    \clist_set:Nx \l_CDR_clist { #1 }
    \clist_remove_duplicates:N \l_CDR_clist
    \exp_args:NV
    \CDR_tag_set:n \l_CDR_clist
  },
  tags .value_required:n = true,
```

**__initialize** Initialization.

```
  __initialize .meta:n = {
    tags = ,
  },
  __initialize .value_forbidden:n = true,
}
\AtBeginDocument{
  \CDR_tag_keys_set:nn { __tags } { __initialize }
}
```

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

```
\CDR_tag_keys_define:nn { __no_tags } {
  tags .code:n = {
    \PackageError
      { coder }
      { Key~'tags'~is~forbidden~for~engines }
      { See~the~coder~manual }
  }
}
```

#### 9.3.4 `__engine` l3keys module

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

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

Known keys include:

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

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

🛑 **`__initialize`** Initialization.

```
  __initialize .meta:n = {
    engine = default,
  },
  __initialize .value_forbidden:n = true,

}
\AtBeginDocument{
  \CDR_tag_keys_set:nn { __engine } { __initialize }
}
```

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

```
\CDR_tag_keys_define:nn { __no_engine } {
  engine .code:n = {
    \PackageError
      { coder }
      { Key~'engine'~is~forbidden~for~engines }
      { See~the~coder~manual }
  }
}
```

### 9.3.5 `default.block` l3keys module

```
\CDR_tag_keys_define:nn { default.block } {
```

Known keys include:

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

```
  tags~format .code:n = \CDR_tag_set:,
  tags~format .value_required:n = true,
```

🛑 **`numbers format=`**⟨***`format commands`***⟩ the format used to display line numbers (mainly font, size and color).

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

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

```
  show~tags .choices:nn =
    { none, left, right, numbers, mirror, dry }
    { \CDR_tag_choices_set: },
  show~tags .default:n = numbers,
```

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

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

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

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

🛑 **`__initialize`** Initialization.

```
  __initialize .meta:n = {
    show~tags = numbers,
    only~top = true,
    use~margin = true,
    numbers~format = {
      \sffamily
      \scriptsize
      \color{gray}
    },
    tags~format = {
      \bfseries
    },
  },
  __initialize .value_forbidden:n = true,

}
\AtBeginDocument{
  \CDR_tag_keys_set:nn { default.block } { __initialize }
}
```

## 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**__initialize** Initialization.

```
  __initialize .meta:n = {
    formatcom = ,
    fontfamily = tt,
    fontsize = auto,
    fontseries = auto,
    fontshape = auto,
    showspaces = false,
    showtabs = false,
    obeytabs = false,
    tabsize = 2,
    defineactive = ,
    reflabel = ,
  },
  __initialize .value_forbidden:n = true,

}
\AtBeginDocument{
  \CDR_tag_keys_set:nn { __fancyvrb } { __initialize }
}
```

### 9.4.2 `__fancyvrb.frame` l3keys module

Block specific options, frame related.

```
\CDR_tag_keys_define:nn { __fancyvrb.frame } {
```

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

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

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

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

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

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

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

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

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

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

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

✅ **__initialize** Initialization.

```
__initialize .meta:n = {
  frame = none,
  framerule = 0.4pt,
  framesep = \fboxsep,
  rulecolor = black,
  fillcolor = ,
  labelposition = none,% auto?
},
__initialize .value_forbidden:n = true,

}
\AtBeginDocument{
  \CDR_tag_keys_set:nn { __fancyvrb.frame } { __initialize }
}
```

### 9.4.3 `__fancyvrb.block` l3keys module

Block specific options, except numbering.

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

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

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

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

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

```
  baselinestretch .code:n = \CDR_tag_set:,
  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.

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

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

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

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

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

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

```
  hfuzz .code:n = \CDR_tag_set:,
  hfuzz .value_required:n = true,
```

🛑 `vspace=`⟨*dimension*⟩ the amount of vertical space added to `\parskip` before and after blocks. Initially `\topsep`.

```
  vspace .code:n = \CDR_tag_set:,
  vspace .value_required:n = true,
```

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

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

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

```
  label .code:n = \CDR_tag_set:,
  label .value_required:n = true,
```

✅ `__initialize` Initialization.

```
  __initialize .meta:n = {
    commentchar = ,
    gobble = 0,
    baselinestretch = auto,
    resetmargins = true,
    xleftmargin = 0pt,
    xrightmargin = 0pt,
    hfuzz = 2pt,
    vspace = \topset,
    samepage = false,
    label = ,
  },
  __initialize .value_forbidden:n = true,

}
\AtBeginDocument{
  \CDR_tag_keys_set:nn { __fancyvrb.block } { __initialize }
}
```

### 9.4.4 `__fancyvrb.number` l3keys module

Block line numbering.

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

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

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

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

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

```
firstnumber .code:n = {
  \regex_match:NnTF \c_CDR_integer_regex { #1 } {
    \CDR_tag_set:
  } {
    \str_case:nnF { #1 } {
      { auto } { \CDR_tag_set: }
      { last } { \CDR_tag_set: }
    } {
      \PackageWarning
        { CDR }
        { Value~‘#1’~not~in~auto,~last. }
    }
  }
},
firstnumber .value_required:n = true,
```

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

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

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

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

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

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

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

✅ **`__initialize`** Initialization.

```
  __initialize .meta:n = {
    numbers = left,
    numbersep = 1ex,
    firstnumber = auto,
    stepnumber = 1,
    numberblanklines = true,
    firstline = ,
    lastline = ,
  },
  __initialize .value_forbidden:n = true,

}
\AtBeginDocument{
  \CDR_tag_keys_set:nn { __fancyvrb.number } { __initialize }
}
```

### 9.4.5 `__fancyvrb.all` l3keys module

Options available when pygments is not used.

```
\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.

```
  commandchars .code:n = \CDR_tag_set:,
  commandchars .value_required:n = true,
```

🔴 **`codes=⟨macro⟩`** to specify catcode changes. For instance, this allows us to include formatted mathematics in verbatim text. Initially empty. Ignored in pygments mode.

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

✅ **`__initialize`** Initialization.

```
  __initialize .meta:n = {
    commandchars = ,
    codes = ,
  },
  __initialize .value_forbidden:n = true,

}
\AtBeginDocument{
  \CDR_tag_keys_set:nn { __fancyvrb.all } { __initialize }
}
```

## 10 \CDRSet

\CDRSet

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

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

### 10.1 CDR@Set l3keys module

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

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

```
  only~description .choices:nn = { false, true, {} } {
    \int_compare:nNnTF \l_keys_choice_int = 1 {
      \prg_set_conditional:Nnn \CDR_if_only_description: { p, T, F, TF } { \prg_return_true: }
    } {
      \prg_set_conditional:Nnn \CDR_if_only_description: { p, T, F, TF } { \prg_return_false: }
    }
  },
  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`.

```
  python~path .code:n = {
    \str_set:Nn \l_CDR_str { #1 }
    \exp_args:Nx \CDR_pygments_setup:n {
      \lua_now:n { CDR:set_python_path('l_CDR_str') }
    }
  },

}
```

### 10.2 Branching

\CDR_if_only_description_p: ⋆
\CDR_if_only_description:*TF* ⋆

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

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

## 10.3 Implementation

\CDRBlock_preflight:n

\CDR_set_preflight:n {⟨*CDR@Set kv list*⟩}

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

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

```
\NewDocumentCommand \CDRSet { m } {
\CDR@Debug{\string\CDRSet}
  \CDR_set_preflight:n { #1 }
  \keys_set_known:nnnN { CDR@Set } { #1 } { CDR@Set } \l_CDR_kv_clist
  \clist_map_inline:nn {
    __pygments, __pygments.block,
    __tags, __engine, default.block, default.code, default,
    __fancyvrb, __fancyvrb.frame, __fancyvrb.block, __fancyvrb.number, __fancyvrb.all
  } {
    \CDR_tag_keys_set_known:nN { ##1 } \l_CDR_kv_clist
\CDR@Debug{ Debug.CDRSet.1:##1/\l_CDR_kv_clist/ }
  }
  \CDR_tag_keys_set_known:nN { .. } \l_CDR_kv_clist
\CDR@Debug{ Debug.CDRSet.2:\CDR_tag_module:n { .. }//\l_CDR_kv_clist/ }
  \CDR_tag_provide_from_kv:V \l_CDR_kv_clist
\CDR@Debug{ Debug.CDRSet.2a:\CDR_tag_module:n { .. }//\l_CDR_kv_clist/ }
  \CDR_tag_keys_set_known:nN { .. } \l_CDR_kv_clist
\CDR@Debug{ Debug.CDRSet.3:\CDR_tag_module:n { .. }//\l_CDR_kv_clist/ }
  \CDR_tag_keys_set:nV { default } \l_CDR_kv_clist
\CDR@Debug{ Debug.CDRSet.4:\CDR_tag_module:n { default } /\l_CDR_kv_clist/ }
  \keys_define:nn { CDR@Set@tags } {
    tags .code:n = {
      \clist_set:Nx \g_CDR_tags_clist { ##1 }
      \clist_remove_duplicates:N \g_CDR_tags_clist
    },
  }
  \keys_set_known:nn { CDR@Set@tags } { #1 }
  \ignorespaces
}
```

# 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.

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

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

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

Store ⟨*value*⟩, which is further retrieved with the instruction \CDR_get_get:cc {⟨*file name*⟩} {⟨*relative key path*⟩}. All the affectations are made at the current TEX group level.

```
\cs_new_protected:Npn \CDR_export_set:ccn #1 #2 #3 {
  \cs_set:cpn { \CDR_export_get_path:cc { #1 } { #2 } } { \exp_not:n { #3 } }
}
\cs_new_protected:Npn \CDR_export_set:Vcn #1 {
  \exp_args:NV
  \CDR_export_set:ccn { #1 }
}
\cs_new_protected:Npn \CDR_export_set:VcV #1 #2 #3 {
  \exp_args:NnV
  \use:n {
    \exp_args:NV \CDR_export_set:ccn #1 { #2 }
  } #3
}
```

\CDR_export_if_exist:cc*TF* ⋆

\CDR_export_if_exist:ccTF {⟨*file name*⟩} ⟨*relative key path*⟩ {⟨*true code*⟩} {⟨*false code*⟩}

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

```
\prg_new_conditional:Nnn \CDR_export_if_exist:cc { p, T, F, TF } {
  \cs_if_exist:cTF { \CDR_export_get_path:cc { #1 } { #2 } } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

\CDR_export_get:cc ⋆

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

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

```
\cs_new:Npn \CDR_export_get:cc #1 #2 {
  \CDR_export_if_exist:ccT { #1 } { #2 } {
    \use:c { \CDR_export_get_path:cc { #1 } { #2 } }
  }
}
```

\CDR_export_get:ccN*TF*

\CDR_export_get:ccNTF {⟨*file name*⟩} {⟨*relative key path*⟩} ⟨*tl var*⟩ {⟨*true code*⟩} {⟨*false code*⟩}

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

```
\prg_new_protected_conditional:Nnn \CDR_export_get:ccN { T, F, TF } {
  \CDR_export_if_exist:ccTF { #1 } { #2 } {
    \tl_set:Nx #3 { \CDR_export_get:cc { #1 } { #2 } }
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

## 11.2 Storage

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

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

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

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

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

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

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

```
\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.

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

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

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

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

```
  tags .code:n = {
    \clist_set:Nx \l_CDR_clist { #1 }
    \clist_remove_duplicates:N \l_CDR_clist
    \prop_put:NVV \l_CDR_export_prop \l_keys_key_str \l_CDR_clist
  },
  tags .value_required:n = true,
```

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

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

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

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

🛑 **`postamble`** the added postamble. Initially empty.

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

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

```
  raw .choices:nn = { false, true, {} } {
    \prop_put:NVx \l_CDR_export_prop \l_keys_key_str {
      \int_compare:nNnTF
        \l_keys_choice_int = 1 { false } { true }
    }
  },
```

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

```
  once .choices:nn = { false, true, {} } {
    \prop_put:NVx \l_CDR_export_prop \l_keys_key_str {
      \int_compare:nNnTF
        \l_keys_choice_int = 1 { false } { true }
    }
  },
```

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

```
  __initialize .meta:n = {
    __initialize_prop = #1,
    file =,
    tags =,
    lang = tex,
    preamble =,
    postamble =,
    raw = false,
    once = true,
  },
  __initialize .default:n = \l_CDR_export_prop,
```

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

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

```
}
```

## 11.4 Implementation

```
\NewDocumentCommand \CDRExport { m } {
  \keys_set:nn { CDR@Export } { __initialize }
  \keys_set:nn { CDR@Export } { #1 }
  \tl_if_empty:NTF \l_CDR_file_tl {
    \PackageWarning
      { coder }
      { Missing~export~key~‘file’ }
  } {
    \CDR_export_set:VcV \l_CDR_file_tl { file } \l_CDR_file_tl
    \prop_map_inline:Nn \l_CDR_export_prop {
      \CDR_export_set:Vcn \l_CDR_file_tl { ##1 } { ##2 }
    }
```

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.

```
    \prop_get:NnNTF \l_CDR_export_prop { tags } \l_CDR_clist {
      \tl_if_empty:NTF \l_CDR_clist {
        \PackageWarning
          { coder }
          { Missing~export~key~‘tags’ }
      } {
        \clist_set_eq:NN \g_CDR_tags_clist \l_CDR_clist
        \clist_remove_duplicates:N \g_CDR_tags_clist
        \clist_put_left:NV \g_CDR_all_tags_clist \l_CDR_clist
        \clist_remove_duplicates:N \g_CDR_all_tags_clist
```

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

```
        \exp_args:NV
        \CDR_export_get:ccNT \l_CDR_file_tl { lang } \l_CDR_tl {
          \clist_map_inline:Nn \g_CDR_tags_clist {
            \CDR_tag_set:ccV { ##1 } { lang } \l_CDR_tl
          }
        }
      }
      \seq_put_left:NV \g_CDR_export_seq \l_CDR_file_tl
    } {
      \PackageWarning
        { coder }
        { Missing~export~key~‘tags’ }
    }
  }
  \ignorespaces
}
```

Files are created at the end of the typesetting process.

```
\AddToHook { enddocument / end } {
  \seq_map_inline:Nn \g_CDR_export_seq {
    \str_set:Nx \l_CDR_str { #1 }
    \lua_now:n { CDR:export_file(’l_CDR_str’) }
    \clist_map_inline:nn {
      tags, raw, once, preamble, postamble
    } {
      \CDR_export_get:ccNT { #1 } { ##1 } \l_CDR_tl {
        \exp_args:NNx
        \str_set:Nn \l_CDR_str { \l_CDR_tl }
        \lua_now:n {
          CDR:export_file_info('##1','l_CDR_str')
        }
      }
    }
    \lua_now:n { CDR:export_complete() }
  }
}
```

## 12 Style

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

`\CDR@StyleDefine`

`\CDR@StyleDefine {⟨pygments style name⟩} {⟨definitions⟩}`

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

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

`\CDR@StyleUse`
`CDR@StyleUseTag`

`\CDR@StyleUse {⟨pygments style name⟩}`
`\CDR@StyleUseTag`

Use the definitions for the given ⟨*pygments style name*⟩. No safe check is made. The `\CDR@StyleUseTag` version finds the ⟨*pygments style name*⟩ from the context.

```
\cs_set:Npn \CDR@StyleUse #1 {
  \tl_use:c { g_CDR@Style/#1 }
}
\cs_set:Npn \CDR@StyleUseTag {
  \CDR@StyleUse { \CDR_tag_get:c { style } }
}
```

`\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.

```
\prg_new_conditional:Nnn \CDR@StyleIfExist:c { TF } {
  \tl_if_exist:cTF { g_CDR@Style/#1 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
\cs_set_eq:NN \CDR@StyleIfExist \CDR@StyleIfExist:cTF
```

## 13 Creating display engines

### 13.1 Utilities

```
\CDRCode_engine:c  ⋆
\CDRCode_engine:V  ⋆
\CDRBlock_engine:c ⋆
\CDRBlock_engine:V ⋆
```

`\CDRCode_engine:c {⟨engine name⟩}`
`\CDRBlock_engine:c {⟨engine name⟩}`

`\CDRCode_engine:c` builds a command sequence name based on ⟨*engine name*⟩. `\CDRBlock_engine:c` builds an environment name based on ⟨*engine name*⟩.

```
\cs_new:Npn \CDRCode_engine:c #1 {
  CDR@colored/code/#1:nn
}
\cs_new:Npn \CDRBlock_engine:c #1 {
  CDR@colored/block/#1
}
\cs_new:Npn \CDRCode_engine:V {
  \exp_args:NV \CDRCode_engine:c
}
\cs_new:Npn \CDRBlock_engine:V {
  \exp_args:NV \CDRBlock_engine:c
}
```

```
\CDRCode_options:c  ⋆
\CDRCode_options:V  ⋆
\CDRBlock_options:c ⋆
\CDRBlock_options:V ⋆
```

`\CDRCode_options:c {⟨engine name⟩}`
`\CDRBlock_options:c {⟨engine name⟩}`

`\CDRCode_options:c` builds a command sequence name based on ⟨*engine name*⟩ used to store the comma list of key value options. `\CDRBlock_options:c` builds a command sequence name based on ⟨*engine name*⟩ used to store the comma list of key value options.

```
\cs_new:Npn \CDRCode_options:c #1 {
  CDR@colored/code~options/#1:nn
}
\cs_new:Npn \CDRBlock_options:c #1 {
  CDR@colored/block~options/#1
}
\cs_new:Npn \CDRCode_options:V {
  \exp_args:NV \CDRCode_options:c
}
\cs_new:Npn \CDRBlock_options:V {
  \exp_args:NV \CDRBlock_options:c
}
```

```
\CDRCode_options_use:c  ⋆
\CDRCode_options_use:V  ⋆
\CDRBlock_options_use:c ⋆
\CDRBlock_options_use:V ⋆
```

`\CDRCode_options_use:c {⟨engine name⟩}`
`\CDRBlock_options_use:c {⟨engine name⟩}`

`\CDRCode_options_use:c` builds a command sequence name based on ⟨*engine name*⟩ and use it. `\CDRBlock_options:c` builds a command sequence name based on ⟨*engine name*⟩ and use it.

```
\cs_new:Npn \CDRCode_options_use:c #1 {
  \CDRCode_if_options:cT { #1 } {
    \use:c { \CDRCode_options:c { #1 } }
  }
}
\cs_new:Npn \CDRBlock_options_use:c #1 {
  \CDRBlock_if_options:cT { #1 } {
    \use:c { \CDRBlock_options:c { #1 } }
  }
}
\cs_new:Npn \CDRCode_options_use:V {
  \exp_args:NV \CDRCode_options_use:c
}
\cs_new:Npn \CDRBlock_options_use:V {
  \exp_args:NV \CDRBlock_options_use:c
}
```

`\l_CDR_engine_tl` Storage for an engine name.

```
\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

`\CDRCodeEngineNew`
`\CDRCodeEngineRenew`

```
\CDRCodeEngineNew  {⟨engine name⟩}{⟨engine body⟩}
\CDRCodeEngineRenew{⟨engine name⟩}{⟨engine body⟩}
```

⟨*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.

```
\cs_new:Npn \CDR_forbidden:n #1 {
  \group_begin:
  \CDR_local_inherit:n { __no_tag, __no_engine }
  \CDR_local_set_known:nN { #1 } \l_CDR_kv_clist
  \group_end:
}
\NewDocumentCommand \CDRCodeEngineNew { mO{}m } {
  \exp_args:Nx
  \tl_if_empty:nTF { #1 } {
    \PackageWarning
      { coder }
      { The~engine~cannot~be~void. }
  } {
    \CDR_forbidden:n { #2 }
    \cs_set:cpn { \CDRCode_options:c { #1 } } { \exp_not:n { #2 } }
    \cs_new:cpn { \CDRCode_engine:c {#1} } ##1 ##2 {
      \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
      #3
    }
    \ignorespaces
  }
}
```

`\CDR_forbidden_keys:n`

`\CDR_forbidden_keys:n {⟨key[=value] items⟩}`

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

```
\cs_new:Npn \CDR_forbidden_keys:n #1 {
  \group_begin:
  \CDR_local_inherit:n { __no_tags, __no_engine }
  \CDR_local_set_known:nN { #1 } \l_CDR_kv_clist
  \group_end:
}
```

```
\NewDocumentCommand \CDRCodeEngineRenew { mO{}m } {
  \exp_args:Nx
  \tl_if_empty:nTF { #1 } {
    \PackageWarning
      { coder }
      { The~engine~cannot~be~void. }
      \use_none:n
  } {
    \cs_if_exist:cTF { \CDRCode_engine:c { #1 } } {
      \CDR_forbidden:n { #2 }
      \cs_set:cpn { \CDRCode_options:c { #1 } } { \exp_not:n { #2 } }
      \cs_set:cpn { \CDRCode_engine:c { #1 } } ##1 ##2 {
        \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
        #3
      }
    } {
      \PackageWarning
        { coder }
        { No~code~engine~#1.}
    }
    \ignorespaces
  }
}
```

`\CDR@CodeEngineApply`

`\CDR@CodeEngineApply {⟨source⟩}`

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

```
\cs_new_protected:Npn \CDR@CodeEngineApply {
  \CDRCode_if_engine:cF { \CDR_tag_get:c { engine } } {
    \PackageError
      { coder }
      { \CDR_tag_get:c { engine }~code~engine~unknown,~replaced~by~'default' }
      { See~\CDRCodeEngineNew~in~the~coder~manual }
    \CDR_tag_set:cn { engine } { default }
  }
  \CDR_tag_get:c { format }
  \exp_args:Nnx
  \use:c { \CDRCode_engine:c { \CDR_tag_get:c { engine } } } {
    \CDR_tag_get:c { \CDR_tag_get:c { engine }~engine~options },
    \CDR_tag_get:c { engine~options }
  }
}
```

`\CDRBlockEngineNew`
`\CDRBlockEngineRenew`

`\CDRBlockEngineNew {⟨`*`engine name`*`⟩} [⟨`*`options`*`⟩] {⟨`*`begin instructions`*`⟩} {⟨`*`end instructions`*`⟩}`
`\CDRBlockEngineRenew {⟨`*`engine name`*`⟩} [⟨`*`options`*`⟩] {⟨`*`begin instructions`*`⟩} {⟨`*`end instructions`*`⟩}`

Create a LaTeX environment uniquely named after ⟨*engine name*⟩, which must be a non void string once expanded. The ⟨*begin instructions*⟩ and ⟨*end instructions*⟩ are lists of instructions which may refer to the name as `#1`, which is the value given to `CDRBlock` environment for key ⟨*engine name*⟩ `engine options`. Various options are available with the `\CDRGetOption` function. *Implementation detail*: the fourth argument is parsed by `\NewDocumentEnvironment`.

```
\NewDocumentCommand \CDRBlockEngineNew { mO{}m } {
  \CDR_forbidden:n { #2 }
  \cs_set:cpn { \CDRBlock_options:c { #1 } } { \exp_not:n { #2 } }
  \NewDocumentEnvironment { \CDRBlock_engine:c { #1 } } { m } {
    \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
    #3
  }
}
```

```
\NewDocumentCommand \CDRBlockEngineRenew { mO{}m } {
  \tl_if_empty:nTF { #1 } {
    \PackageError
      { coder }
      { The~engine~cannot~be~void. }
      { See~\string\CDRBlockEngineNew~in~the~coder~manual }
      \use_none:n
  } {
    \cs_if_exist:cTF { \CDRBlock_engine:c { #1 } } {
      \CDR_forbidden:n { #2 }
      \cs_set:cpn { \CDRBlock_options:c { #1 } } { \exp_not:n { #2 } }
      \RenewDocumentEnvironment { \CDRBlock_engine:c { #1 } } { m } {
        \cs_set_eq:NN \CDRGetOption \CDR_tag_get:c
        #3
      }
    } {
      \PackageError
        { coder }
        { No~block~engine~#1.}
        { See~\string\CDRBlockEngineNew~in~the~coder~manual }
    }
  }
}
```

\CDRBlock_engine_begin:
\CDR@Block_engine_end:

\CDRBlock_engine_begin:
\CDRBlock_engine_end:

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

```
\cs_new:Npn \CDRBlock_engine_begin: {
  \CDRBlock_if_engine:cF { \CDR_tag_get:c { engine } } {
    \PackageError
      { coder }
      { \CDR_tag_get:c { engine }~block~engine~unknown,~replaced~by~‘default’ }
      {See~\CDRBlockEngineNew~in~the~coder~manual}
    \CDR_tag_set:cn { engine } { default }
  }
  \exp_args:Nnx
  \use:c { \CDRBlock_engine:c \CDR_tag_get:c { engine } } {
    \CDR_tag_get:c { \CDR_tag_get:c { engine }~engine~options },
    \CDR_tag_get:c { engine~options },
  }
}
\cs_new:Npn \CDRBlock_engine_end: {
  \use:c { end \CDRBlock_engine:c \CDR_tag_get:c { engine } }
}
%    \begin{MacroCode}
%
% \subsection{Conditionals}
%
% \begin{function}[EXP,TF]{\CDRCode_if_engine:c}
% \begin{syntax}
% \cs{CDRCode_if_engine:cTF} \Arg{engine name} \Arg{true code} \Arg{false code}
% \end{syntax}
% If there exists a code engine with the given \metatt{engine name},
% execute \metatt{true code}.
% Otherwise, execute \metatt{false code}.
% \end{function}
%    \begin{MacroCode}[OK]
\prg_new_conditional:Nnn \CDRCode_if_engine:c { p, T, F, TF } {
  \cs_if_exist:cTF { \CDRCode_engine:c { #1 } } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
\prg_new_conditional:Nnn \CDRCode_if_engine:V { p, T, F, TF } {
  \cs_if_exist:cTF { \CDRCode_engine:V #1 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

\CDRBlock_if_engine:c*TF* ⋆

\CDRBlock_if_engine:c {⟨*engine name*⟩} {⟨*true code*⟩} {⟨*false code*⟩}

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

```
\prg_new_conditional:Nnn \CDRBlock_if_engine:c { p, T, F, TF } {
  \cs_if_exist:cTF { \CDRBlock_engine:c { #1 } } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
\prg_new_conditional:Nnn \CDRBlock_if_engine:V { p, T, F, TF } {
  \cs_if_exist:cTF { \CDRBlock_engine:V #1 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

`\CDRCode_if_options:cTF` ⋆

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

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

```
\prg_new_conditional:Nnn \CDRCode_if_options:c { p, T, F, TF } {
  \cs_if_exist:cTF { \CDRCode_options:c { #1 } } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
\prg_new_conditional:Nnn \CDRCode_if_options:V { p, T, F, TF } {
  \cs_if_exist:cTF { \CDRCode_options:V #1 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

`\CDRBlock_if_options:cTF` ⋆

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

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

```
\prg_new_conditional:Nnn \CDRBlock_if_options:c { p, T, F, TF } {
  \cs_if_exist:cTF { \CDRBlock_options:c { #1 } } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
\prg_new_conditional:Nnn \CDRBlock_if_options:V { p, T, F, TF } {
  \cs_if_exist:cTF { \CDRBlock_options:V #1 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

### 13.3 Default code engine

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

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

### 13.4 efbox code engine

```
\AtBeginDocument {
  \@ifpackageloaded{efbox} {
    \CDRCodeEngineNew {efbox} {
      \efbox[#1]{#2}
    }
  } {}
}
```

### 13.5 Block mode default engine

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

### 13.6 tcolorbox related engine

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

## 14 `\CDRCode` function

### 14.1 API

`\CDR@Sp` `\CDR@Sp`

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

```
\cs_new:Npn \CDR@DefinePygSp {
  \CDR_if_tag_truthy:cTF { showspaces } {
    \cs_set:Npn \CDR@Sp {{\FancyVerbSpace}}
  } {
    \cs_set_eq:NN \CDR@Sp \space
  }
}
```

`\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.

```
\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.

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

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

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

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

● `__initialize` initialize

```
  __initialize .meta:n = {
    tag = default,
    engine~options = ,
  },
  __initialize .value_forbidden:n = true,
```

```
}
```

### 14.4 Implementation

```
\NewDocumentCommand \CDRCode { O{} } {
  \group_begin:
  \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
    \prg_return_false:
  }
  \clist_set:Nn \l_CDR_kv_clist { #1 }
  \CDRCode_tags_setup:N \l_CDR_kv_clist
  \CDRCode_engine_setup:N \l_CDR_kv_clist
  \CDR_local_inherit:n {
    __code, default.code, __pygments, default,
  }
  \CDR_local_set_known:N \l_CDR_kv_clist
  \CDR_tag_provide_from_kv:V \l_CDR_kv_clist
  \CDR_local_set_known:N \l_CDR_kv_clist
  \CDR_local_inherit:n {
    __fancyvrb,
  }
  \CDR_local_set:V \l_CDR_kv_clist
  \CDRCode:n
}
```

`\CDRCode_tags_setup:N`
`\CDRCode_engine_setup:N`

`\CDRCode_tags_setup:N {`⟨*clist var*⟩`}`
`\CDRCode_engine_setup:N {`⟨*clist var*⟩`}`

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

```
\cs_new_protected_nopar:Npn \CDRCode_tags_setup:N #1 {
\CDR@Debug{\string \CDRCode_tags_setup:N, \string #1 }
  \CDR_local_inherit:n { __tags }
  \CDR_local_set_known:N #1
  \CDR_if_tag_exist_here:ccT { __local } { tags } {
    \CDR_tag_get:cN { tags } \l_CDR_clist
    \clist_if_empty:NF \l_CDR_clist {
      \clist_gset_eq:NN \g_CDR_tags_clist \l_CDR_clist
    }
  }
  \clist_if_empty:NT \g_CDR_tags_clist {
    \PackageWarning
      { coder }
      { No~(default)~tags~provided. }
  }
\CDR@Debug {CDRCode_tags_setup:N\space\g_CDR_tags_clist}
```

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

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

Now setup the engine options if any.

```
\cs_new_protected_nopar:Npn \CDRCode_engine_setup:N #1 {
\CDR@Debug{\string \CDRCode_engine_setup:N, \string #1}
  \CDR_local_inherit:n { __engine }
  \CDR_local_set_known:N #1
  \CDR_tag_get:cNT { engine } \l_CDR_tl {
    \clist_put_left:Nx #1 { \CDRCode_options_use:V \l_CDR_tl }
  }
}
```

`\CDRCode:n`

`\CDRCode:n` ⟨*delimiter*⟩

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

```
\cs_new_protected_nopar:Npn \CDRCode:n #1 {
  \bool_if:nTF { \CDR_has_pygments_p: && \CDR_if_tag_truthy_p:c {pygments}} {
    \cs_set:Npn \CDR@StyleUseTag {
      \CDR@StyleUse { \CDR_tag_get:c { style } }
      \cs_set_eq:NN \CDR@StyleUseTag \prg_do_nothing:
    }
    \DefineShortVerb { #1 }
    \SaveVerb [
      aftersave = {
        \exp_args:Nx \UndefineShortVerb { #1 }
        \lua_now:n { CDR:hilight_code_setup() }
        \CDR_tag_get:cN {lang} \l_CDR_tl
        \lua_now:n { CDR:hilight_set_var('lang') }
        \CDR_tag_get:cN {cache} \l_CDR_tl
        \lua_now:n { CDR:hilight_set_var('cache') }
        \CDR_tag_get:cN {debug} \l_CDR_tl
        \lua_now:n { CDR:hilight_set_var('debug') }
        \CDR_tag_get:cN {escapeinside} \l_CDR_tl
        \lua_now:n { CDR:hilight_set_var('escapeinside') }
        \CDR_tag_get:cN {mathescape} \l_CDR_tl
        \lua_now:n { CDR:hilight_set_var('mathescape') }
        \CDR_tag_get:cN {style} \l_CDR_tl
        \lua_now:n { CDR:hilight_set_var('style') }
        \lua_now:n { CDR:hilight_set_var('source', 'FV@SV@CDR@Source') }
        \clist_set_eq:NN \FV@KeyValues \l_CDR_kv_clist
        \FV@UseKeyValues
        \frenchspacing
        \FV@BaseLineStretch
        \FV@FontSize
        \FV@FontFamily
        \FV@FontSeries
        \FV@FontShape
        \selectfont
        \FV@DefineWhiteSpace
        \FancyVerbDefineActive
        \FancyVerbFormatCom
        \CDR@DefinePygSp
        \CDR_tag_get:c { format }
        \CDR@CodeEngineApply {
          \CDR@StyleIfExist { \CDR_tag_get:c { style } } { } {
            \lua_now:n { CDR:hilight_source(true, false) }
            \input { \l_CDR_pyg_sty_tl }
          }
          \CDR@StyleUseTag
          \lua_now:n { CDR:hilight_source(false, true) }
          \makeatletter
          \lua_now:n {
            CDR.synctex_tag = tex.get_synctex_tag();
            CDR.synctex_line = tex.inputlineno;
            tex.set_synctex_mode(1)
          }
          \CDR_if_tag_truthy:cT { mbox } { \mbox } {
            \input { \l_CDR_pyg_tex_tl }\ignorespaces
          }
          \lua_now:n {
            tex.set_synctex_mode(0)
          }
          \makeatother
        }
        \group_end:
      }
    ] { CDR@Source } #1
  } {
    \DefineShortVerb { #1 }
    \SaveVerb [
      aftersave = {
        \UndefineShortVerb { #1 }
        \cs_set_eq:NN \CDR@FormattingPrep \FV@FormattingPrep
```

```
        \cs_set:Npn \FV@FormattingPrep {
          \CDR@FormattingPrep
          \CDR_tag_get:c { format }
        }
        \CDR@CodeEngineApply { \CDR_if_tag_truthy:cT { mbox } { \mbox } {
          \clist_set_eq:NN \FV@KeyValues \l_CDR_kv_clist
          \FV@UseKeyValues
          \FV@FormattingPrep
          \FV@SV@CDR@Code
        } }
        \group_end:
      }
    ] { CDR@Code } #1
  }
}
```

## 15 CDRBlock environment

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

### 15.1 __block l3keys module

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

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

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

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

● `no export format=⟨format commands⟩` a format appended to `format`, `tags format` and `numbers format` when `no export` is `true`.. Initially empty.

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

● `dry numbers[=true|false]` Initially `false`.

```
  dry~numbers .code:n = \CDR_tag_boolean_set:x { #1 },
  dry~numbers .default:n = true,
```

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

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

● `engine options=⟨engine options⟩` options forwarded to the engine. They are appended to the options given with key `⟨engine name⟩ engine options`. Mainly a convenient user interface shortcut.

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

🛑 **`__initialize`** initialize

```
  __initialize .meta:n = {
    no~export = false,
    no~export~format = ,
    dry~numbers = false,
    test = false,
    engine~options = ,
  },
  __initialize .value_forbidden:n = true,

}
```

## 15.2 Implementation

### 15.2.1 Storage

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

**`__start`** for the first index

**`__step`** for the step, defaults to 1

**`__last`** for the last index, included

```
\CDR_int_new:cn { __start } { 0 }
\CDR_int_new:cn { __step  } { 0 }
\CDR_int_new:cn { __last  } { 0 }
```

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

### 15.2.2 Preparation

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

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

`\CDRBlock_preflight:n` `\CDRBlock_preflight:n {⟨CDR@Block kv list⟩}`

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

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

### 15.2.3 Main environment

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

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

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

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

```
\cs_new:Npn \FVB@CDRBlock {
  \@bsphack
  \exp_args:NV \CDRBlock_preflight:n \FV@KeyValues
  \begingroup
  \lua_now:n {
    CDR.synctex_tag = tex.get_synctex_tag();
    CDR.synctex_line = tex.inputlineno;
    tex.set_synctex_mode(1)
  }
  \seq_clear:N \l_CDR_vrb_seq
  \cs_set_protected_nopar:Npn \FV@ProcessLine ##1 {
    \seq_put_right:Nn \l_CDR_vrb_seq { ##1 }
  }
  \FV@Scan
}
```

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

```
\cs_new:Npn \FVE@CDRBlock {
  \CDRBlock_setup:
  \CDR_if_no_export:F {
    \seq_map_inline:Nn \l_CDR_vrb_seq {
      \tl_set:Nn \l_CDR_tl { ##1 }
      \lua_now:n { CDR:record_line('l_CDR_tl') }
    }
  }
  \CDRBlock_engine_begin:
  \tl_clear:N \FV@ListProcessLastLine
  \CDR_if_pygments:TF {
    \CDRBlock@Pyg
  } {
    \CDRBlock@FV
  }
  \lua_now:n {
    tex.set_synctex_mode(0);
    CDR.synctex_line = 0;
  }
  \CDRBlock_engine_end:
  \CDRBlock_teardown:
  \endgroup
  \@esphack
  \noindent
}
\DefineVerbatimEnvironment{CDRBlock}{CDRBlock}{}
%    \begin{MacroCode}
\cs_new_protected_nopar:Npn \CDRBlock_setup: {
\CDR@Debug { \string \CDRBlock_setup: , \FV@KeyValues }
  \prg_set_conditional:Nnn \CDR_if_block: { p, T, F, TF } {
    \prg_return_true:
  }
  \CDR_tag_keys_set:nn { __block } { __initialize }
```

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

```
  \CDRBlock_tags_setup:N \FV@KeyValues
  \CDRBlock_engine_setup:N \FV@KeyValues
  \CDR_local_inherit:n {
    __block, __pygments.block, default.block,
    __pygments, default
  }
  \CDR_local_set_known:N \FV@KeyValues
  \CDR_tag_provide_from_kv:V \FV@KeyValues
  \CDR_local_set_known:N \FV@KeyValues
 \CDR@Debug{\string \CDRBlock_setup:.KV1:\l_CDR_kv_clist}
```

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

```
  \CDR_local_inherit:n {
    \CDR_if_tag_eq:cnF { engine } { default } {
      __fancyvrb.frame,
    },
    __fancyvrb.number,
  }
  \CDR_local_set_known:N \FV@KeyValues
```

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

```
  \CDR_local_inherit:n {
    __fancyvrb.block,
    __fancyvrb,
  }
  \CDR_local_set_known:VN \FV@KeyValues \l_CDR_kv_clist
  \lua_now:n {
    CDR:hilight_block_setup('g_CDR_tags_clist')
  }
  \CDR_set_conditional:Nn \CDR_if_pygments:
    { \CDR_has_pygments_p: && \CDR_if_tag_truthy_p:c { pygments } }
  \CDR_set_conditional:Nn \CDR_if_no_export:
    { \CDR_if_tag_truthy_p:c { no~export } }
  \CDR_set_conditional:Nn \CDR_if_numbers_dry:
    { \CDR_if_tag_truthy_p:c { dry~numbers } }
  \CDR_set_conditional:Nn \CDR_if_dry_tags:
    { \CDR_if_tag_eq_p:cn { show~tags } { dry } }
  \CDR_set_conditional:Nn \CDR_if_number_on:
    { ! \CDR_if_tag_eq_p:cn { numbers } { none } }
  \CDR_set_conditional:Nn \CDR_if_already_tags: {
    \CDR_if_tag_truthy_p:c { only~top } &&
    \CDR_clist_if_eq_p:NN \g_CDR_tags_clist \g_CDR_last_tags_clist
  }
  \CDR_if_number_on:T {
    \clist_map_inline:Nn \g_CDR_tags_clist {
      \CDR_int_if_exist:cF { ##1 } {
        \CDR_int_new:cn { ##1 } { 1 }
      }
    }
  }
}
```

\CDRBlock_teardown:

\CDRBlock_teardown:

Update the stored line numbers and send the hilight_block_teardown message to CDR.

```
\cs_new_protected_nopar:Npn \CDRBlock_teardown: {
  \bool_if:nT { \CDR_if_number_on_p: && !\CDR_if_numbers_dry_p: } {
    \tl_set:Nx \l_CDR_tl { \seq_count:N \l_CDR_vrb_seq }
    \clist_map_inline:Nn \g_CDR_tags_clist {
      \CDR_int_gadd:cn { ##1 } { \l_CDR_tl }
    }
  }
  \lua_now:n {
    CDR:hilight_block_teardown()
  }
  \CDR_if_dry_tags:F {
    \clist_gset_eq:NN \g_CDR_last_tags_clist \g_CDR_tags_clist
  }
}
```

### 15.2.4 pygments only

Parts of CDRBlock environment specific to pygments.

\CDRBlock@Pyg

\CDRBlock@Pyg

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

```
\cs_set_protected:Npn \CDRBlock@Pyg {
\CDR@Debug { \string\CDRBlock@Pyg / \the\inputlineno }
  \CDR_tag_get:cN {lang} \l_CDR_tl
  \lua_now:n { CDR:hilight_set_var('lang') }
  \CDR_tag_get:cN {cache} \l_CDR_tl
  \lua_now:n { CDR:hilight_set_var('cache') }
  \CDR_tag_get:cN {debug} \l_CDR_tl
  \lua_now:n { CDR:hilight_set_var('debug') }
  \CDR_tag_get:cN {texcomments} \l_CDR_tl
  \lua_now:n { CDR:hilight_set_var('texcomments') }
  \CDR_tag_get:cN {escapeinside} \l_CDR_tl
  \lua_now:n { CDR:hilight_set_var('escapeinside') }
  \CDR_tag_get:cN {mathescape} \l_CDR_tl
  \lua_now:n { CDR:hilight_set_var('mathescape') }
  \CDR_tag_get:cN {style} \l_CDR_tl
  \lua_now:n { CDR:hilight_set_var('style') }
  \cctab_select:N \c_document_cctab
  \CDR@StyleIfExist { \l_CDR_tl } { } {
    \lua_now:n { CDR:hilight_source(true, false) }
    \input { \l_CDR_pyg_sty_tl }
  }
  \CDR@StyleUseTag
  \CDR@DefinePygSp
  \lua_now:n { CDR:hilight_source(false, true) }
  \fvset{ commandchars=\\\{\} }
  \FV@UseVerbatim {
    \CDR_tag_get:c { format }
    \CDR_if_no_export:T {
      \CDR_tag_get:c { no~export~format }
    }
    \makeatletter
    \input{ \l_CDR_pyg_tex_tl }\ignorespaces
    \makeatother
  }
}
```

### Info

```
\cs_new:Npn \CDR@NumberFormat {
  \CDR_tag_get:c { numbers~format }
}
\cs_new:Npn \CDR@NumberSep {
  \hspace{ \CDR_tag_get:c { numbersep } }
}
\cs_new:Npn \CDR@TagsFormat {
  \CDR_tag_get:c { tags~format }
}
```

`\CDR_info_N_L:n`
`\CDR_info_N_R:n`
`\CDR_info_T_L:n`
`\CDR_info_T_R:n`

`\CDR_info_N_L:n {⟨line number⟩}`
`\CDR_info_T_L:n {⟨line number⟩}`

Core methods to display the left and right information. The `T` variants contain tags informations, they are only used on the first line eventually. The `N` variants are for line numbers only.

```
\cs_new:Npn \CDR_info_N_L:n #1 {
  \hbox_overlap_left:n {
    \cs_set:Npn \baselinestretch { 1 }
    { \CDR@NumberFormat
      #1
    }
    \CDR@NumberSep
  }
}
\cs_new:Npn \CDR_info_T_L:n #1 {
  \hbox_overlap_left:n {
    \cs_set:Npn \baselinestretch { 1 }
    \CDR@NumberFormat
    \smash{
    \parbox[b]{\marginparwidth}{
      \raggedleft
        { \CDR@TagsFormat \g_CDR_tags_clist :}
      }
      #1
    }
    \CDR@NumberSep
  }
}
\cs_new:Npn \CDR_info_N_R:n #1 {
  \hbox_overlap_right:n {
    \CDR@NumberSep
    \cs_set:Npn \baselinestretch { 1 }
    \CDR@NumberFormat
    #1
  }
}
\cs_new:Npn \CDR_info_T_R:n #1 {
  \hbox_overlap_right:n {
    \cs_set:Npn \baselinestretch { 1 }
    \CDR@NumberSep
    \CDR@NumberFormat
    \smash {
      \parbox[b]{\marginparwidth}{
        \raggedright
        #1:
        {\CDR@TagsFormat \space \g_CDR_tags_clist}
      }
    }
  }
}
```

`\CDR_number_alt:n` First line.

```
\cs_set:Npn \CDR_number_alt:n #1 {
  \use:c { CDRNumber
    \CDR_if_number_main:nTF { #1 } { Main } { Other }
  } { #1 }
}
\cs_set:Npn \CDR_number_alt: {
\CDR@Debug{ALT: \CDR_int_use:c { __n } }
  \CDR_number_alt:n { \CDR_int_use:c { __n } }
}
```

`\CDRNumberMain`
`\CDRNumberOther`
`\CDRIfLR`

`\CDRNumberMain  {`⟨*integer expression*⟩`}`
`\CDRNumberOther {`⟨*integer expression*⟩`}`
`\CDRIfLR {`⟨*left commands*⟩`} {`⟨*right commands*⟩`}`

This is used when typesseting line numbers. The default `...Other` function just gobble one argument. The ⟨*integer expression*⟩ is exactly what will be displayed. The `\cs{CDRIfLR}` allows to format the numbers differently on the left and on the right.

```
\cs_new:Npn \CDRNumberMain {
}
\cs_new:Npn \CDRNumberOther {
                \use_none:n
}
```

\CDR@NumberMain
\CDR@NumberOther

\CDR@NumberMain
\CDR@NumberOther

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

```
\cs_new:Npn \CDR@NumberMain {
                \CDRNumberMain { \CDR_int_use:c { __n } }
}
\cs_new:Npn \CDR@NumberOther {
                \CDRNumberOther { \CDR_int_use:c { __n } }
}
```

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

\CDR_line_[LRNSO]_[LRN]:nn

\CDR_line_[LRNSO]_[LRN]:nn {⟨*line number*⟩} {⟨*line content*⟩}

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

```
\cs_new:Npn \CDR_line_N_N:n {
\CDR@Debug {Debug.CDR_line_N_N:n}
  \CDR_line_box_N:n
}

\cs_new:Npn \CDR_line_L_N:n #1 {
\CDR@Debug {Debug.CDR_line_L_N:n}
  \CDR_line_box:nnn { \CDR_info_T_L:n { } } { #1 } { }
}

\cs_new:Npn \CDR_line_R_N:n #1 {
\CDR@Debug {Debug.CDR_line_R_N:n}
  \CDR_line_box:nnn { } { #1 } { \CDR_info_T_R:n { } }
}

\cs_new:Npn \CDR_line_S_N:n {
\CDR@Debug {Debug.CDR_line_S_N:n}
  \CDR_line_box_N:n
}

\cs_new:Npn \CDR_line_O_N:n {
\CDR@Debug {STEP:CDR_line_O_N:n}
  \CDR_line_box_N:n
}

\cs_new:Npn \CDR_line_N_L:n #1 {
```

```
\CDR@Debug {STEP:CDR_line_N_L:n}
  \CDR_if_no_number:TF {
    \CDR_line_box:nnn {
      \CDR_info_N_L:n { \CDR@NumberMain }
    } { #1 } {}
  } {
    \CDR_if_number_main:nTF { \CDR_int:c { __n } + 1 } {
      \CDR_line_box_L:n { #1 }
    } {
      \CDR_line_box:nnn {
        \CDR_info_N_L:n { \CDR@NumberMain }
      } { #1 } {}
    }
  }
}

\cs_new:Npn \CDR_line_L_L:n #1 {
\CDR@Debug {STEP:CDR_line_L_L:n}
  \CDR_if_number_single:TF {
    \CDR_line_box:nnn {
      \CDR_info_T_L:n { \space \CDR@NumberMain }
    } { #1 } {}
  } {
    \CDR_if_no_number:TF {
      \cs_set:Npn \CDR@@Line {
        \cs_set:Npn \CDR@@Line {
          \CDR_line_box_L:nn { \CDR_info_N_L:n { \CDR@NumberOther } }
        }
        \CDR_line_box_L:nn { \CDR_info_N_L:n { \CDR@NumberMain } }
      }
    } {
      \cs_set:Npn \CDR@@Line {
        \CDR_line_box_L:nn { \CDR_info_N_L:n { \CDR_number_alt: } }
      }
    }
    \CDR_line_box:nnn { \CDR_info_T_L:n { } } { #1 } { }
  }
}

\cs_new:Npn \CDR_line_R_R:n #1 {
\CDR@Debug {STEP:CDR_line_R_R:n}
  \CDR_if_number_single:TF {
    \CDR_line_box:nnn { } { #1 } {
      \CDR_info_T_R:n { \CDR@NumberMain }
    }
  } {
    \CDR_if_no_number:TF {
      \cs_set:Npn \CDR@@Line {
        \cs_set:Npn \CDR@@Line {
          \CDR_line_box_R:nn { \CDR_info_N_R:n { \CDR@NumberOther } }
        }
        \CDR_line_box_R:nn { \CDR_info_N_R:n { \CDR@NumberMain } }
      }
    } {
      \cs_set:Npn \CDR@@Line {
        \CDR_line_box_R:nn { \CDR_info_N_R:n { \CDR_number_alt: } }
      }
    }
    \CDR_line_box:nnn { } { #1 } { \CDR_info_T_R:n { } }
  }
}

\cs_new:Npn \CDR_line_R_L:n #1 {
\CDR@Debug {STEP:CDR_line_R_L:n}
  \CDR_line_box:nnn {
    \CDR_if_no_number:TF {
      \CDR_info_N_L:n { \CDR@NumberMain }
    } {
      \CDR_if_number_main:nTF { \CDR_int:c { __n } + 1 } {
        \CDR_info_N_L:n { \CDR_number_alt: }
      } {
        \CDR_info_N_L:n { \CDR@NumberMain }
      }
    }
  } { #1 } {
    \CDR_info_T_R:n { }
  }
}

\cs_set_eq:NN \CDR_line_S_L:n \CDR_line_L_L:n
\cs_set_eq:NN \CDR_line_O_L:n \CDR_line_R_L:n

\cs_new:Npn \CDR_line_N_R:n #1 {
\CDR@Debug {STEP:CDR_line_N_R:n}
  \CDR_if_no_number:TF {
    \CDR_line_box:nnn {} { #1 } {
      \CDR_info_N_R:n { \CDR@NumberMain }
    }
  } {
    \CDR_if_number_main:nTF { \CDR_int:c { __n } + 1 } {
      \CDR_line_box_R:n { #1 }
    } {
      \CDR_line_box:nnn {} { #1 } {
        \CDR_info_N_R:n { \CDR@NumberMain }
      }
    }
  }
}

\cs_new:Npn \CDR_line_L_R:n #1 {
\CDR@Debug {STEP:CDR_line_L_R:n}
  \CDR_line_box:nnn {
    \CDR_info_T_L:n { }
  } { #1 } {
    \CDR_if_no_number:TF {
      \CDR_info_N_R:n { \CDR@NumberMain }
    } {
      \CDR_if_number_main:nTF { \CDR_int:c { __n } + 1 } {
        \CDR_info_N_R:n { \CDR_number_alt: }
      } {
        \CDR_info_N_R:n { \CDR@NumberMain }
      }
    }
  }
}

\cs_set_eq:NN \CDR_line_S_R:n \CDR_line_R_R:n
\cs_set_eq:NN \CDR_line_O_R:n \CDR_line_L_R:n


\cs_new:Npn \CDR_line_box_N:n #1 {
\CDR@Debug {STEP:CDR_line_box_N:n}
  \CDR_line_box:nnn { } { #1 } {}
}

\cs_new:Npn \CDR_line_box_L:n #1 {
\CDR@Debug {STEP:CDR_line_box_L:n}
  \CDR_line_box:nnn {
    \CDR_info_N_L:n { \CDR_number_alt: }
  } { #1 } {}
}

\cs_new:Npn \CDR_line_box_R:n #1 {
\CDR@Debug {STEP:CDR_line_box_R:n}
  \CDR_line_box:nnn { } { #1 } {
    \CDR_info_N_R:n { \CDR_number_alt: }
  }
}
```

`\CDR_line_box:nnn`
`\CDR_line_box_L:nn`
`\CDR_line_box_R:nn`
`\CDR_line_box:nn`

`\CDR_line_box:nnn {⟨left info⟩} {⟨line content⟩} {⟨right info⟩}`
`\CDR_line_box_L:nn {⟨left info⟩} {⟨line content⟩}`
`\CDR_line_box_R:nn {⟨right info⟩} {⟨line content⟩}`

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

```
\cs_new:Npn \CDR_line_box:nnn #1 #2 #3 {
\CDR@Debug {\string\CDR_line_box:nnn/\tl_to_str:n{#1}/.../\tl_to_str:n{#3}/}
  \directlua {
    tex.set_synctex_tag( CDR.synctex_tag )
  }

  \lua_now:e {
    tex.set_synctex_line(CDR.synctex_line +( \CDR_int_use:c { __i }) )
  }
  \hbox to \hsize {
    \kern \leftmargin
    {
      \let\CDRIfLR\use_i:nn
      #1
    }
    \hbox to \linewidth {
      \FV@LeftListFrame
      #2
      \hss
      \FV@RightListFrame
    }
    {
      \let\CDRIfLR\use_ii:nn
      #3
    }
  }
  \ignorespaces
}
\cs_new:Npn \CDR_line_box_L:nn #1 #2 {
  \CDR_line_box:nnn { #1 } { #2 } {}
}
\cs_new:Npn \CDR_line_box_R:nn #1 #2 {
\CDR@Debug {STEP:CDR_line_box_R:nn}
  \CDR_line_box:nnn { } {#2} { #1 }
}
\cs_new:Npn \CDR_line_box_N:nn #1 #2 {
\CDR@Debug {STEP:CDR_line_box_N:nn}
  \CDR_line_box:nnn { } { #2 } {}
}
```

### Lines

```
\cs_new:Npn \CDR@Line {
\CDR@Debug {\string\CDR@Line}
  \peek_meaning_ignore_spaces:NTF [%]
  { \CDR_line:nnn } {
    \PackageError
      { coder }
      { Missing~'['%]
        ~at~first~\string\CDR@Line~call }
      { See~the~coder~developper~manual }
  }
}
```

\CDR_line:nnn

`\CDR_line:nnn {⟨CDR@Line kv list⟩} {⟨line index⟩} {⟨line content⟩}`

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

```
\keys_define:nn { CDR@Line } {
  last .code:n = \CDR_int_set:cn { __last } { #1 },
}
\cs_new:Npn \CDR_line:nnn [ #1 ] #2 {
\CDR@Debug {\string\CDR_line:nnn}
  \keys_set:nn { CDR@Line } { #1 }
  \CDR_if_number_on:TF {
    \CDR_int_set:cn { __n } { 1 }
    \CDR_int_set:cn { __i } { 1 }
```

Set the first line number.

```
     \CDR_int_set:cn { __start } { 1 }
    \CDR_if_tag_eq:cnTF { firstnumber } { last } {
      \clist_map_inline:Nn \g_CDR_tags_clist {
        \clist_map_break:n {
          \CDR_int_set:cc { __start } { ##1 }
\CDR@Debug {START: ##1=\CDR_int_use:c { ##1 } }
        }
      }
    } {
      \CDR_if_tag_eq:cnF { firstnumber } { auto } {
        \CDR_int_set:cn { __start } { \CDR_tag_get:c { firstnumber } }
      }
    }
```

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

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

`\CDR_if_visible_at_index_p:n` ⋆
`\CDR_if_visible_at_index:nTF` ⋆

`\CDR_if_visible_at_index:nTF {`⟨*relative line number*⟩`} {`⟨*true code*⟩`} {`⟨*false code*⟩`}`

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

```
    \CDR_set_conditional_alt:Nn \CDR_if_visible_at_index:n {
      \CDR_if_number_visible_p:n { ##1 + \CDR_int:c { __start } - (#2) }
    }
    \CDR_set_conditional_alt:Nn \CDR_if_number_visible:n {
      ! \CDR_int_compare_p:cNn { __last } < { ##1 }
    }
    \CDR_int_compare:cNnTF { __step } < 2 {
      \CDR_int_set:cn { __step } { 1 }
      \CDR_set_conditional_alt:Nn \CDR_if_number_main:n {
        \CDR_if_number_visible_p:n { ##1 }
      }
    } {
      \CDR_set_conditional_alt:Nn \CDR_if_number_main:n {
        \int_compare_p:nNn {
```

```
        ( ##1 ) / \CDR_int:c { __step }  * \CDR_int:c { __step }
      } = { ##1 }
      && \CDR_if_number_visible_p:n { ##1 }
    }
  }
\CDR@Debug {CDR_line:nnn:1}
```

```
    \CDR_set_conditional:Nn \CDR_if_no_number: {
      \CDR_int_compare_p:cNn { __start } > {
        \CDR_int:c { __last } / \CDR_int:c { __step } * \CDR_int:c { __step }
      }
    }
    \cs_set:Npn \CDR@Line ##1 {
\CDR@Debug {\string\CDR@Line(A), \the\inputlineno}
      \CDR_int_set:cn { __i } { ##1 }
      \CDR_int_set:cn { __n } { ##1 + \CDR_int:c { __start } - (#2) }
      \tl_set:Nx \@currentlabel { \CDR_int_use:c { __n } }
      {
        \advance\interlinepenalty\widowpenalty
        \bool_if:nT {
          \CDR_int_compare_p:cNn { __n } = { 2 }
         || \CDR_int_compare_p:cNn { __n } = { \CDR_int:c { __last } }
        } {
          \advance\interlinepenalty\clubpenalty
        }
        \penalty\interlinepenalty
      }
      \CDR@@Line
    }
    \CDR_int_set:cn { __n } { 1 + \CDR_int:c { __start } - (#2) }
    \tl_set:Nx \@currentlabel { \CDR_int_use:c { __n } }
  } {
\CDR@Debug {NUMBER~OFF}
    \cs_set:Npn \CDR@Line ##1 {
\CDR@Debug {\string\CDR@Line(B), \the\inputlineno}
      \CDR@@Line
    }
  }
\CDR@Debug {STEP_S, \CDR_int_use:c {__step}, \CDR_int_use:c {__last} }
```

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

```
  \tl_clear:N \l_CDR_tl
  \CDR_if_already_tags:TF {
    \tl_put_right:Nn \l_CDR_tl { _N }
  } {
    \exp_args:Nx
    \str_case:nnF { \CDR_tag_get:c { show~tags } } {
      { left  } { \tl_put_right:Nn \l_CDR_tl { _L } }
      { right } { \tl_put_right:Nn \l_CDR_tl { _R } }
      { none  } { \tl_put_right:Nn \l_CDR_tl { _N } }
      { dry   } { \tl_put_right:Nn \l_CDR_tl { _N } }
      { numbers } { \tl_put_right:Nn \l_CDR_tl { _S } }
      { mirror  } { \tl_put_right:Nn \l_CDR_tl { _O } }
    } { \PackageError
          { coder }
          { Unknown~show~tags~options~:~ \CDR_tag_get:c { show~tags } }
          { See~the~coder~manual }
    }
  }
```

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

```
  \exp_args:Nx
  \str_case:nnF { \CDR_tag_get:c { numbers } } {
    { left  } {
      \tl_put_right:Nn \l_CDR_tl { _L }
      \cs_set:Npn \CDR@@Line { \CDR_line_box_L:n }
    }
    { right } {
      \tl_put_right:Nn \l_CDR_tl { _R }
      \cs_set:Npn \CDR@@Line { \CDR_line_box_R:n }
    }
    { none  } {
      \tl_put_right:Nn \l_CDR_tl { _N }
      \cs_set:Npn \CDR@@Line { \CDR_line_box_N:n }
    }
  } { \PackageError
        { coder }
        { Unknown~numbers~options~:~ \CDR_tag_get:c { numbers } }
        { See~the~coder~manual }
  }
\CDR@Debug {BRANCH:CDR_line \l_CDR_tl :n}
  \use:c { CDR_line \l_CDR_tl :n }
}
```

### 15.2.5 fancyvrb only

pygments is not used, fall back to fancyvrb features.

CDRBlock@FV    \CDRBlock@Fv

```
\cs_new_protected:Npn \CDRBlock@FV {
\CDR@Debug {DEBUG.Block.FV}
  \FV@UseKeyValues
  \FV@UseVerbatim {
    \CDR_tag_get:c { format }
    \CDR_if_no_export:T {
      \CDR_tag_get:c { no~export~format }
    }
    \tl_set:Nx \l_CDR_tl { [ last=%]
      \seq_count:N \l_CDR_vrb_seq %[
    ] }
    \seq_map_indexed_inline:Nn \l_CDR_vrb_seq {
      \exp_last_unbraced:NV \CDR@Line \l_CDR_tl { ##1 } { ##2 }
      \tl_clear:N \l_CDR_tl
    }
  }
}
```

### 15.2.6 Utilities

This is put aside for better clarity.

\CDR_if_middle_column:
\CDR_if_right_column:

`\CDR_int_if_middle_column:TF {⟨true code⟩} {⟨false code⟩}`
`\CDR_int_if_right_column:TF {⟨true code⟩} {⟨false code⟩}`

Execute ⟨*true code*⟩ when in the middle or right column, ⟨*false code*⟩ otherwise.

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

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

\CDR_if_tags_visible_p:n ⋆
\CDR_if_tags_visible:n*TF* ⋆

`\CDR_if_tags_visible:nTF {⟨left|right⟩} {⟨true code⟩} {⟨false code⟩}`

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

```
\prg_set_conditional:Nnn \CDR_if_tags_visible:n { p, T, F, TF } {
  \bool_if:nTF {
    ( \CDR_if_tag_eq_p:cn { show~tags } { ##1 } ||
      \CDR_if_tag_eq_p:cn { show~tags } { numbers } &&
      \CDR_if_tag_eq_p:cn { numbers } { ##1 }
    ) && ! \CDR_if_already_tags_p:
  } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

\CDRBlock_tags_setup:N
\CDRBlock_engine_setup:N

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

```
\cs_new_protected_nopar:Npn \CDRBlock_tags_setup:N #1 {
\CDR@Debug{ \string \CDRBlock_tags_setup:N, \string #1 }
  \CDR_local_inherit:n { __tags }
  \CDR_local_set_known:N #1
  \CDR_if_tag_exist_here:ccT { __local } { tags } {
    \CDR_tag_get:cN { tags } \l_CDR_clist
    \clist_if_empty:NF \l_CDR_clist {
      \clist_gset_eq:NN \g_CDR_tags_clist \l_CDR_clist
    }
  }
  \clist_if_empty:NT \g_CDR_tags_clist {
    \PackageWarning
      { coder }
      { No~(default)~tags~provided. }
  }
\CDR@Debug {CDRBlock_tags_setup:N\space\g_CDR_tags_clist}
```

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

```
  \CDR_get_inherit:f {
    \g_CDR_tags_clist,
    __block, __tags, __engine, default.block, __pygments.block,
    __fancyvrb.block __fancyvrb.frame, __fancyvrb.number,
    __pygments, default, __fancyvrb,
  }
```

For each ⟨*tag name*⟩, create an l3int variable and initialize it to 1.

```
  \clist_map_inline:Nn \g_CDR_tags_clist {
    \CDR_int_if_exist:cF { ##1 } {
      \CDR_int_new:cn { ##1 } { 1 }
    }
  }
}
```

Now setup the engine options if any.

```
\cs_new_protected_nopar:Npn \CDRBlock_engine_setup:N #1 {
\CDR@Debug{ \string \CDRBlock_engine_setup:N, \string #1 }
  \CDR_local_inherit:n { __engine }
  \CDR_local_set_known:N #1
  \CDR_tag_get:cNT { engine } \l_CDR_tl {
    \clist_put_left:Nx #1 { \CDRBlock_options_use:V \l_CDR_tl }
  }
}
```

## 16 Management

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

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

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

`\CDR_if_show_code_p: ⋆`
`\CDR_if_show_code:TF ⋆`

`\CDR_if_show_code:TF {⟨true code⟩} {⟨false code⟩}`

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

```
\prg_new_conditional:Nnn \CDR_if_show_code: { p, T, F, TF } {
  \bool_if:nTF {
    \g_CDR_in_impl_bool && !\g_CDR_with_impl_bool
  } {
    \prg_return_false:
  } {
    \prg_return_true:
  }
}
```

\g_CDR_with_impl_bool

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

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

\CDRPreamble

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

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

```
\DeclareDocumentCommand \CDRPreamble { m m } {
  \msg_info:nnn
    { coder }
    { :n }
    { Reading~preamble~from~file~"#2". }
  \tl_set:Nn \l_CDR_tl { #2 }
  \exp_args:NNx
  \tl_set:Nx #1 { \lua_now:n {CDR.print_file_content('l_CDR_tl')} }
}
```

## 17 Section separators

\CDRImplementation
\CDRFinale

`\CDRImplementation`
`\CDRFinale`

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

## 18 Finale

```
\newcounter{CDR@impl@page}
\DeclareDocumentCommand \CDRImplementation {} {
  \bool_if:NF \g_CDR_with_impl_bool {
    \clearpage
    \bool_gset_true:N \g_CDR_in_impl_bool
    \let\CDR@old@part\part
    \DeclareDocumentCommand\part{som}{}
    \let\CDR@old@section\section
    \DeclareDocumentCommand\section{som}{}
    \let\CDR@old@subsection\subsection
    \DeclareDocumentCommand\subsection{som}{}
    \let\CDR@old@subsubsection\subsubsection
    \DeclareDocumentCommand\subsubsection{som}{}
    \let\CDR@old@paragraph\paragraph
    \DeclareDocumentCommand\paragraph{som}{}
    \let\CDR@old@subparagraph\subparagraph
    \DeclareDocumentCommand\subparagraph{som}{}
    \cs_if_exist:NT \refsection{ \refsection }
    \setcounter{ CDR@impl@page }{ \value{page} }
  }
}
\DeclareDocumentCommand\CDRFinale {} {
  \bool_if:NF \g_CDR_with_impl_bool {
    \clearpage
    \bool_gset_false:N \g_CDR_in_impl_bool
    \let\part\CDR@old@part
    \let\section\CDR@old@section
    \let\subsection\CDR@old@subsection
    \let\subsubsection\CDR@old@subsubsection
    \let\paragraph\CDR@old@paragraph
    \let\subparagraph\CDR@old@subparagraph
    \setcounter { page } { \value{ CDR@impl@page } }
  }
}
%\cs_set_eq:NN \CDR_line_number: \prg_do_nothing:
```

## 19 Finale

```
%\AddToHook { cmd/FancyVerbFormatLine/before } {
%  \CDR_line_number:
%}

\ExplSyntaxOff

```

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

```
\AtBeginDocument{
  \InputIfFileExists{coder.cfg}{}{}
}
%</sty>
```