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` gives different examples. Here is the implementation of the package.
This LaTeX package requires LuaTeX and may use syntax coloring based on pygment.

1 Package dependencies

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

2 Similar technologies

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

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

3 Known bugs and limitations

- coder does not play well with docstrip.

4 Namespace

LaTeX identifiers related to coder start with `CDR`, including both commands and evironments. 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.

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

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

5 Presentation

coder is a triptych of three complementary components

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

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, both 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 `\CDRNewCodeEngine` or `\CDRNewBlockEngine`.

5.1 Code flow

The normal code flow is

1. from coder.sty, LaTeX parses a code snippet as `\CDRCode` argument of `CDRBlock` environment body, somehow stores it, and calls either `CDR:process_code` or `CDR:process_block`,
2. coder-util.lua reads the content of some command, and store it in a `json` file, together with informations to process this code snippet properly,
3. coder-tool.py is asked by coder-util.lua to read the `json` file and eventually uses pygment 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-tool.py gives to coder-util.lua some LaTeX instructions to both input the `*.pyg.sty` and the `*.pyg.tex` file, these are finally executed and the code is displayed with colors. coder-tool.py is also partially responsible of code line numbering.

5.2 File exports

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

5.3 Display engine

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

5.4 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.

5.5 Properties and inheritance

Properties cover various informations, from the language of the code, to the color and font. They are uniquely identified by a path, and may be defined at the *global* level or at the *tag* level.

the *global* level is set by `\CDRSet` and `\CDRExport`, it consists of global variables,

the *tag* level is set by `\CDRSet`, `\CDRCode` and `CDRBlock` environment.

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

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 rep   = string.rep
local lpeg  = require("lpeg")
local P, Cg, Cp, V = lpeg.P, lpeg.Cg, lpeg.Cp, lpeg.V
require("lualibs.lua")
local json   = _ENV.utilities.json
```

3 General purpose material

`CDR_PY_PATH` Location of the coder-tool.py utility.

```
local CDR_PY_PATH = io.popen(
  [[kpsewhich coder-tool.py]]
):read('a'):match("^%s*(.-)%s*$")
```

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

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

Escape the given string. NEVER USED.

```
local function escape(s)
  s = s:gsub('\\','\\\\')
  s = s:gsub('\r','\\r')
  s = s:gsub('\n','\\n')
  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 pygment 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.

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

```
local function load_exec(chunk)
  local func, err = load(chunk)
  if func then
    local 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
```

safe_equals ⟨*variable*⟩ `= CDR.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*⟩`]`.

```
local eq_pattern = P({ Cp() * P('=')^1 * Cp() + 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_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:process_code`.

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

4 Properties

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

options_reset CDR:options_reset()

Instance method. This is called by coder.sty's \CDR_to_lua:.

```
local function options_reset(self)
  self['.options'] = {}
end
```

option_add CDR:option_add(⟨*key*⟩, ⟨*value name*⟩)

Instance method. This is called by coder.sty's \CDR_to_lua:.

```
local function option_add(self, key, value_name)
  local p = self['.options']
  p[key] = token.get_macro(assert(value_name))
end
```

5 Exportation

export_file CDR:export(⟨*file name var*⟩)

This is called at export time. ⟨*file name var*⟩ is the TEX variable containing the file name.

```
local function export_file(self, file_name)
  self['.name'] = assert(tex.token(assert(file_name)))
  self['.export'] = {}
end
```

export_file_info CDR:export_file_info(⟨*key*$_n$*ame*⟩, ⟨*value name*⟩)

This is called at export time.

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

export_complete CDR:export_complete()

This is called at export time.

```
local function export_complete(self)
  local name = sef['.name']
  local export = sef['.export']
  local tt = {}
  local s = export.preamble
  if s then
```

```
    tt[#tt+1] = s
  end
  for _,tag in ipairs(export.tags) do
    s = records[tag]:concat('\n')
    tt[#tt+1] = s
    records[tag] = { [1] = s }
  end
  s = export.postamble
  if s then
    tt[#tt+1] = s
  end
  if #tt>0 then
    local fh = assert(io.open(name,'w'))
    fh:write(tt:concat('\n'))
    fh:close()
  end
  self['.file'] = nil
  self['.exportation'] = nil
end
```

6 Caching

We save some computation time by pygmentizing files only when necessary. The codertool.py is expected to create a `*.pygd.sty` file for a style and a `*.pygd.tex` file for colored 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 colored code files created.

`cache_clean_all`	`CDR:cache_clean_all()`
`cache_record`	`CDR:cache_record(`⟨*style name.pygd.sty*⟩`, `⟨*digest.pygd.tex*⟩`)`
`cache_clean_unused`	`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.pygd.sty*⟩ and ⟨*digest.pygd.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(dir_p) do
    to_remove[f] = true
  end
  for k,_ in pairs(to_remove) do
    os.remove(dir_p .. k)
  end
end
local function cache_record(self, style, colored)
  self['.style_set'][style] = true
  self['.colored_set'][colored] = true
end
```

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

7 Return the module

Known fields are

`date` to store ⟨*date string*⟩,

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

`dir_p` is the path to the directory where all

Known methods are

escape

make_directory

load_exec

record_start

record_stop

record_line

process_code

cache_clean_all

cache_record

cache_clean_unused

`pygment` related material is stored,

`json_p` is the path to the JSON file used by coder-tool.py utility.

`.style_set` the set of style names used

`.colored_set` the set of "colored" names used

.records the ⟨*tag name*⟩-->⟨*line array*⟩ table

.fields the ⟨*field name*⟩-->⟨*domain*⟩ --> ⟨*key*⟩ --> ⟨*value*⟩ table

.exports the ⟨*file name*⟩-->⟨*info table*⟩ table

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

field_group_begin begin a group,

field_group_end end a group,

field_put put a field value,

field_get get a field value,

field_print get a field value,

```
return {
  _DESCRIPTION       = _DESCRIPTION,
  _VERSION           = token.get_macro('fileversion'),
  date               = token.get_macro('filedate'),
  CDR_PY_PATH        = CDR_PY_PATH,
  escape             = escape,
  make_directory     = make_directory,
  load_exec          = load_exec,
  load_exec_output   = load_exec_output,
  record_start       = record_start,
  record_stop        = record_stop,
  record_line        = function(self,line) end,
  process_code       = process_code,
  cache_clean_all    = cache_clean_all,
  cache_record       = cache_record,
  cache_clean_unused = cache_clean_unused,
  options_reset      = options_reset,
  option_add         = option_add,
  ['.style_set']     = {},
  ['.colored_set']   = {},
  ['.options']       = {},
  ['.export']        = {},
  ['.name']          = nil,
  already            = false,
}
%</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 Header and global declarations

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

from posixpath import split
import sys
import argparse
import re
from pathlib import Path
import hashlib
import json
from pygments import highlight
from pygments.formatters.latex import LatexEmbeddedLexer, LatexFormatter
from pygments.lexers import get_lexer_by_name
from pygments.util import ClassNotFound
from pygments.util import guess_decode
```

2 `Controller` main class

The first class variables are string formats. They are used to let coder-tool.py talk back to TEX through coder-util.lua.

```
class Controller:

  @staticmethod
  def ensure_bool(x):
    if x == True or x == False: return x
    x = x[0:1]
    return x == 'T' or x == 't'
```

2.1 `Object` nested class

```
  class Object(object):
    def __new__(cls, d={}, *args, **kvargs):
      __cls__ = d.get('__cls__', 'arguments')
      if __cls__ == 'options':
        return super(Controller.Object, cls)['__new__'](
          Controller.Options, *args, **kvargs
        )
      elif __cls__ == 'FV':
        return super(Controller.Object, cls)['__new__'](
          Controller.FV, *args, **kvargs
        )
      else:
```

```
      return super(Controller.Object, cls)['__new__'](
        Controller.Arguments, *args, **kvargs
      )
  def __init__(self, d={}):
    for k, v in d.items():
      if type(v) == str:
        if v.lower() == 'true':
          setattr(self, k, True)
          continue
        elif v.lower() == 'false':
          setattr(self, k, False)
          continue
      setattr(self, k, v)
  def __repr__(self):
    return f"{object['__repr__'](self)}: {self['__dict__']}"
```

2.2 `Options` nested class

```
class Options(Object):
  docclass = 'article'
  style = 'autumn'
  preamble = ''
  lang = 'tex'
  escapeinside = ""
  gobble = 0
  tabsize = 4
  style = 'default'
  already_style = False
  texcomments = False
  mathescape =  False
  linenos = False
  linenostart = 1
  linenostep = 1
  linenosep = '0pt'
  encoding = 'guess'
  verboptions = ''
  nobackground = False
  commandprefix = 'Py'
```

2.3 `Arguments` nested class

```
class FV(Object):
  pass
```

2.4 `Arguments` nested class

```
class Arguments(Object):
  cache = False
  debug = False
  code = ""
  json = ""
  options = None
  directory = ""
```

2.5 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.directory_p` The full path to the directory containing the various output files related to pygment. When not given inside the `json` file, this is the directory of the `json` file itself. The directory is created when missing.

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

```
_directory_p = None
@property
def directory_p(self):
  p = self._directory_p
  if p:
    return p
  p = self.arguments.directory
  if p:
    p = Path(p)
  else:
    p = self.json_p
    if p:
      p = p.parent / p.stem
    else:
      p = Path('SHARED')
  if p:
    p = p.resolve().with_suffix(".pygd")
    p.mkdir(exist_ok=True)
  self._directory_p = p
  return p
```

`self.colored_p` The full path to the file where colored commands created by pygment should be stored.

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

```
_colored_p = None
@property
def colored_p(self):
  p = self._colored_p
  if p:
    return p
```

```
    p = self.arguments.output
    if p:
      p = Path(p).resolve()
    else:
      p = self.json_p
      if p:
        p = p.with_suffix(".pyg.tex")
    self._colored_p = p
    return p
```

`self.sty_p` The full path to the style file with definition created by pygment.

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

```
  @property
  def sty_p(self):
    return (self.directory_p / self.options.style).with_suffix(".pyg.sty")
```

`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 highlighting 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",
      default=None,
      help="display informations useful for debugging"
    )
    parser.add_argument(
      "json",
      metavar="json data file",
      help="""
file name with extension of information to specify which processing is required
"""
    )
    return parser

```

2.6 Static methods

`Controller.tex_command`	`self.tex_command(`⟨*asynchronous tex command*⟩`)`
`Controller.lua_command`	`self.lua_command(`⟨*asynchronous lua command*⟩`)`
`Controller.lua_command_now`	`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 tex_command(cmd):
  print(f'<<<<<?TEX:{cmd}>>>>>')
@staticmethod
def lua_command(cmd):
  print(f'<<<<<?LUA:{cmd}>>>>>')
@staticmethod
def lua_command_new(cmd):
  print(f'<<<<<!LUA:{cmd}>>>>>')
```

2.7 Methods

2.7.1 `__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
    )
  options = self.options = self.arguments.options
  formatter = self.formatter = LatexFormatter(style=options.style)
  formatter.docclass = options.docclass
  formatter.preamble = options.preamble
  formatter.linenos = self.ensure_bool(options.linenos)
  formatter.linenostart = abs(options.linenostart)
  formatter.linenostep = abs(options.linenostep)
  formatter.verboptions = options.verboptions
  formatter.nobackground = self.ensure_bool(options.nobackground)
  formatter.commandprefix = options.commandprefix
  formatter.texcomments = self.ensure_bool(options.texcomments)
  formatter.mathescape = self.ensure_bool(options.mathescape)
  formatter.envname = u'CDR@Pyg@Verbatim'

  try:
    lexer = self.lexer = get_lexer_by_name(self.arguments.lang)
  except ClassNotFound as err:
```

```
    sys.stderr.write('Error: ')
    sys.stderr.write(str(err))

  escapeinside = options.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 = abs(int(self.gobble))
  if gobble:
    lexer.add_filter('gobble', n=gobble)
  tabsize = abs(int(self.tabsize))
  if tabsize:
    lexer.tabsize = tabsize
  lexer.encoding = ''

```

2.7.2 get_tex_p

get_tex_p ⟨*variable*⟩ = self.get_tex_p(⟨*digest string*⟩)

The full path of the file where the colored commands created by `pygment` are stored. The digest allow to uniquely identify the code initially colored such that caching is easier.

```
def get_tex_p(self, digest):
  return (self.directory_p / digest).with_suffix(".pyg.tex")
```

2.7.3 process

self.process self.process()

Main entry point.

```
def process(self):
  self.create_style()
  self.create_pygmented()
  print('create_tool.py: done')
  return 0
```

2.7.4 create_style

self.create_style self.create_style()

Where the ⟨*code*⟩ is pygmentized.

```
def create_style(self):
  options = self.options
  formatter = self.formatter
```

```
    style = None
    if not self.ensure_boolean(options.already_style):
      style = formatter.get_style_defs() \
        .replace(r'\makeatletter', '') \
        .replace(r'\makeatother', '') \
        .replace('\n', '%\n')
      style = re.sub(
        r"\expandafter\def\csname\s*(.*?)\endcsname",
        r'\cs_new:cpn{\1}',
        style,
        flags=re.M
      )
      style = re.sub(
        r"\csname\s*(.*?)\endcsname",
        r'\use:c{\1}',
        style,
        flags=re.M
      )
      style = fr'''%
\ExplSyntaxOn
\makeatletter
\CDR_style_gset:nn {{{options.style}}} {{%
{style}%
}}%
\makeatother
\ExplSyntaxOff
'''
    sty_p = self.sty_p
    if self.arguments.cache and sty_p.exists():
      print("Already available:", sty_p)
    else:
      with sty_p.open(mode='w',encoding='utf-8') as f:
        f.write(style)
```

2.7.5 pygmentize

These are pygment's `LatexFormatter` options, only used internally by coder.sty to talk to pygment. This is here for the record.

`style=`⟨*name*⟩ the pygment style to use. Initially `default`.

`full` Tells the formatter to output a `full` document, i.e. a complete self-contained document (default: "False"). choose a pygment style. Forbidden.

`title` If 'full' is true, the title that should be used to caption the document (default: """). Forbidden.

`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: """). 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.

linenostep=⟨*integer*⟩ 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`.

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`.

`self.pygmentize` ⟨*code variable*⟩, ⟨*style variable*⟩ = `self.pygmentize(`⟨*code*⟩`[, inline=`⟨*yorn*⟩`])`

Where the ⟨*code*⟩ is pygmentized.

```
  def pygmentize(self, code, inline=True):
    options = self.options
    formatter = self.formatter
    mode = 'Code' if inline else 'Block'
    envname = formatter.envname = rf'CDR@Pyg@{mode}'
    code = highlight(code, self.lexer, formatter)
    m = re.match(
      rf'(\begin{{{envname}}}.*?\n)(.*?)(\n'
      rf'\end{{{envname}}}\s*)\Z',
      code,
      flags=re.S
    )
    assert(m)
    if inline:
      ans_code = rf'''\bgroup
\CDRCode@Prepare:n {{{options.style}}}%
{m.group(2)}%
\egroup
'''
    else:
      ans_code = []
      linenos = options.linenos
```

```
      linenostart = abs(int(options.linenostart))
      linenostep = abs(int(options.linenostep))
      numbers = []
      lines = []
      counter = linenostart
      all_lines = m.group(2).split('\n')
      for line in all_lines:
        line = re.sub(r'^ ', r'\vphantom{Xy}~', line)
        line = re.sub(r' ', '~', line)
        if linenos:
          if (counter - linenostart) % linenostep == 0:
            line = rf'\CDR_lineno:n{{{counter}}}' + line
            numbers.append(str(counter))
          counter += 1
        lines.append(line)
      ans_code.append(fr'''%
\begin{{CDR@Block/engine/{options.style}}}
\CDRBlock@linenos@used:n {{{',.join(numbers)}}}%
{m.group(1)}{'\n'.join(lines)}{m.group(3)}%
\end{{CDR@Block/engine/{options.style}}}
''' )
      ans_code = "".join(ans_code)
    return ans_code
```

2.7.6 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):
    code = self.arguments.code
    if not code:
      return False
    code = self.pygmentize(code, self.ensure_bool(self.arguments.inline))
    h = hashlib.md5(str(code).encode('utf-8'))
    out_p = self.get_tex_p(h.hexdigest())
    if self.arguments.cache and out_p.exists():
      print("Already available:", out_p)
    else:
      with out_p.open(mode='w',encoding='utf-8') as f:
        f.write(r'''% -*- mode: latex -*-
\makeatletter
''')
        f.write(code)
        f.write(r'''\makeatother
''')
    self.tex_command( rf'''%
\CDR_remove:n {{colored:}}%
\input {{ \tl_to_str:n {{{out_p}}} }}%
\CDR:n {{colored:}}%
''')
    sty_p = self.sty_p
    if sty_p.parent.stem != 'SHARED':
```

```
      self.lua_command_now( fr'''
CDR:cache_record({sty_p.name}),{out_p.name})
''' )
    print("PREMATURE EXIT")
    exit(1)
```

2.8 Main entry

```
if __name__ == '__main__':
  try:
    ctrl = Controller()
    sys.exit(ctrl.process())
  except KeyboardInterrupt:
    sys.exit(1)
%</py>
```

File III
coder.sty implementation

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

1 Installation test

```
\NewDocumentCommand \CDRTest {} {
  \sys_if_shell:TF {
    \_CDR_has_pygment:F {
      \msg_warning:nnn
        { coder }
        { :n }
        { No~"pygmentize"~found. }
    }
  } {
    \msg_warning:nnn
      { coder }
      { :n }
      { No~unrestricted~shell~escape~for~"pygmentize".}
  }
}
```

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.

```
\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
\c_CDR_slash

Root identifier for tag properties, used throughout the pakage.

```
\str_const:Nn \c_CDR_tag_get { CDR@tag@get }
\str_const:Nx \c_CDR_slash { \tl_to_str:n {/} }
```

(*End definition for* \c_CDR_tag_get *and* \c_CDR_slash*. These variables are documented on page* ??.)

4 Implementation details

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

5 Variables

5.1 Internal scratch variables

These local variables are used in a very limited scope.

\l_CDR_bool

Local scratch variable.

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

(*End definition for* \l_CDR_bool*. 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_in` Input file identifier

```
\ior_new:N \l_CDR_in
```

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

`\l_CDR_out` Output file identifier

```
\iow_new:N \l_CDR_out
```

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

5.3 Global variables

Line number counter for the code chunks.

`\g_CDR_code_int` Chunk number counter.

```
\int_new:N \g_CDR_code_int
```

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

`\g_CDR_code_prop` Global code property list.

```
\prop_new:N \g_CDR_code_prop
```

(*End definition for* `\g_CDR_code_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_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* ??.)

`\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* ??.)

6 Tag properties

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

6.1 Helpers

`\g_CDR_tag_path_seq` Global variable to store relative key path. Used for automatic management to know what has been defined explicitly.

```
\seq_new:N \g_CDR_tag_path_seq
```

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

`\CDR_tag_get_path:nn` ⋆ `\CDR_tag_get_path:nn {⟨tag name⟩} {⟨relative key path⟩}`

Internal: return a unique key based on the arguments. Used to store and retrieve values.

```
\cs_new:Npn \CDR_tag_get_path:nn #1 #2 {
  \c_CDR_tag_get @ #1 @ #2 :
}
```

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:nn {⟨*tag name*⟩} {⟨*relative key path*⟩}. Only ⟨*tag name*⟩ and ⟨*relative key path*⟩ containing no @ character are supported. Record the relative key path (the part after the tag name) of the current full key path in g_CDR_tag_path_seq. All the affectations are made at the current TEX group level.

```
\cs_new_protected:Npn \CDR_tag_set:ccn #1 #2 #3 {
  \seq_put_left:Nx \g_CDR_tag_path_seq { #2 }
  \cs_set:cpn { \CDR_tag_get_path:nn { #1 } { #2 } } { \exp_not:n { #3 } }
}
\cs_generate_variant:Nn \CDR_tag_set:ccn { ccV }
```

\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 CDR@tag/⟨*tag name*⟩/⟨*relative key path*⟩, an exception is raised on the contrary. This is meant to be call from \keys_define:nn argument. Implementation detail: the last argument is parsed by the last command.

```
\cs_new:Npn \CDR_tag_set:n {
  \exp_args:NnV
  \regex_extract_once:nnNTF {
    ^CDR@tags/tag/([^/]*)/(.*)$
  } \l_keys_path_str \l_CDR_seq { \use_none:n {$}
    \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. Both are guessed from \l_keys_path_str or \l_keys_value_tl. More precisely, \l_keys_path_str is expected to read something like CDR@tag/⟨*tag name*⟩/⟨*relative key path*⟩, an exception is raised on the contrary. This is meant to be call from \keys_define:nn argument.

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

\CDR_tag_set:cc

\CDR_tag_set:cc {⟨*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:cc #1 {
  \exp_args:NnV
  \regex_extract_once:nnNTF {
    ^CDR@tags/tag/([^/]*)/.*$
  } \l_keys_path_str \l_CDR_seq { \use_none:n { $ }
    \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.

```
\cs_new:Npn \CDR_tag_choices: {
  \exp_args:NVV
  \str_if_eq:nnT \l_keys_key_tl \l_keys_choice_tl {
    \exp_args:NnV
    \regex_extract_once:nnNT {
      ^(.*)/.*$
    } \l_keys_path_str \l_CDR_seq { \use_none:n { $ }
      \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:Npn \CDR_tag_choices_set: {
  \CDR_tag_choices:
  \exp_args:NV
  \CDR_tag_set:n \l_keys_choice_tl
}
```

\CDR_tag_boolean_set:

\CDR_tag_boolean_set:

Calls \CDR_tag_set:n with false if the first item is selected, true otherwise. Before, ensure that the \l_keys_path_str is set properly.

```
\cs_new:Npn \CDR_tag_boolean_set: {
  \CDR_tag_choices:
  \exp_args:Nx
  \CDR_tag_set:n {
    \int_compare:nNnTF \l_keys_index_tl = 1 { false } { true }
  }
}
```

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

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

\CDR_tag_if_exist:nn*TF* ⋆

\CDR_tag_if_exist:nnTF {⟨*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.

```
\prg_new_conditional:Nnn \CDR_tag_if_exist:nn { T, F, TF } {
  \cs_if_exist:cTF { \CDR_tag_get_path:nn { #1 } { #2 } } {
    \prg_return_true:
  } {
    \seq_if_exist:cTF { \CDR_tag_parent_seq:n { #1 } } {
      \seq_map_tokens:cn
        { \CDR_tag_parent_seq:n { #1 } }
        { \CDR_tag_if_exist_f:nn { #2 } }
    } {
      \prg_return_false:
    }
  }
```

```
}
\cs_new:Npn \CDR_tag_if_exist_f:nn #1 #2 {
  \quark_if_no_value:nTF { #2 } {
    \seq_map_break:n {
      \prg_return_false:
    }
  } {
    \CDR_tag_if_exist:nnT { #2 } { #1 } {
      \seq_map_break:n {
        \prg_return_true:
      }
    }
  }
}
```

\CDR_tag_get:nn ⋆ \CDR_tag_get:nn {⟨*tag name*⟩} {⟨*relative key path*⟩}

The property value stored for ⟨*tag name*⟩ and ⟨*relative key path*⟩. Takes care of inheritance.

```
\cs_new:Npn \CDR_tag_get:nn #1 #2 {
  \CDR_tag_if_exist_here:nnTF { #1 } { #2 } {
    \use:c { \CDR_tag_get_path:nn { #1 } { #2 } }
  } {
    \seq_if_exist:cT { \CDR_tag_parent_seq:n { #1 } } {
      \seq_map_tokens:cn
        { \CDR_parent_seq:n { #1 } }
        { \CDR_tag_get_f:nn { #2 } }
    }
  }
}
\cs_new:Npn \CDR_tag_get_f:nn #1 #2 {
  \quark_if_no_value:nF { #2 } {
    \CDR_if_exist_here:nnT { #2 } { #1 } {
      \seq_map_break:n {
        \use:c { \CDR_tag_get_path:nn { #2 } { #1 } }
      }
    }
  }
}
```

\CDR_tag_get:n ⋆ \CDR_tag_get:n {⟨*relative key path*⟩}

The property value stored for the `__local` ⟨*tag name*⟩ and ⟨*relative key path*⟩. Takes care of inheritance. Implementation detail: the parameter is parsed by the last command of the expansion.

```
\cs_new:Npn \CDR_tag_get:n {
  \CDR_tag_get:nn { __local }
}
```

\CDR_tag_get:nN

\CDR_tag_get:nN {⟨*relative key path*⟩} {⟨*tl variable*⟩}

Put in ⟨*tl variable*⟩ the property value stored for the `__local` ⟨*tag name*⟩ and ⟨*relative key path*⟩.

```
\cs_new:Npn \CDR_tag_get:nN #1 #2 {
  \tl_set:Nx #2 { \CDR_tag_get:n { #1 } }
}
```

\CDR_tag_get:nnN*TF*

\CDR_tag_get:nnNTF {⟨*tag name*⟩} {⟨*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*⟩.

```
\prg_new_conditional:Nnn \CDR_tag_get:nnN { T, F, TF } {
  \CDR_tag_if_exist:nnTF { #1 } { #2 } {
    \tl_set:Nx #3 \CDR_tag_get:nn { #1 } { #2 }
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

6.4 Inherit

\CDR_tag_parent_seq:n ⋆

\CDR_tag_parent_seq:n {⟨*tag name*⟩}

Return the name of the sequence variable containing the list of the parents.

```
\cs_new:Npn \CDR_tag_parent_seq:n #1 {
  g_CDR:parent.tag @ #1 _seq
}
```

\CDR_tag_inherit:nn

\CDR_tag_inherit:nn {⟨*tag name*⟩} {⟨*parent comma list*⟩}

Set the parents of ⟨*tag name*⟩ to the given list.

```
\cs_new:Npn \CDR_tag_inherit:nn #1 #2 {
  \tl_set:Nx \l_CDR_tl { \CDR_tag_parent_seq:n { #1 } }
  \seq_set_from_clist:cn \l_CDR_tl { #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 }
}
```

6.5 Handling unknown tags

6.5.1 Utilities

`\CDR_tag:n` ⋆ `\CDR_tag:n {⟨tag name⟩}`

Build a key path.

```
\cs_new:Npn \CDR_tag:n #1 {
  \c_CDR_tag \c_CDR_slash #1
}
```

`\keys_define:on`
`\keys_define:(Vx|oo)`
`\keys_set:xn`

Various variants

```
\clist_map_inline:nn { o, Vx, oo } {
  \cs_generate_variant:Nn \keys_define:nn { #1 }
}
\cs_generate_variant:Nn \keys_set:nn { x }
```

6.5.2 Implementation

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

`\CDR_keys_set:nn`
`\CDR_keys_set_unknown:nnN`

`\CDR_keys_set:nn {⟨tag name⟩} {⟨key[=value] items⟩}`
`\CDR_keys_set_unknown:nnN {⟨tag name⟩} {⟨key[=value] items⟩} ⟨tl var⟩`

Wrappers over `\CDR_keys_set:nn` and `\CDR_keys_set_unknown:nnN` where the module is given by `\CDR_tag:n {⟨tag name⟩}`.

```
\cs_new:Npn \CDR_keys_set:nn #1 {
  \keys_set:xn { \CDR_tag:n { #1 } }
}
\cs_generate_variant:Nn \keys_set_unknown:nnnN { xnx }
\cs_new:Npn \CDR_keys_set_unknown:nnN #1 #2 {
  \keys_set_unknown:xnxN { \CDR_tag:n { #1 } } { #2 } { \CDR_tag:n { #1 } }
}
```

`\CDR_tag_provide_from_clist:n`
`\CDR_tag_provide_from_keyval:n`

`\CDR_tag_provide_from_clist:n {⟨deep comma list⟩}`
`\CDR_tag_provide_from_keyval:n {⟨key-value list⟩}`

⟨*deep comma list*⟩ has format `\c_CDR_tag/`⟨*tag name comma list*⟩. Parse the ⟨*key-value list*⟩ for full key path matching `\c_CDR_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 `\keys_parse:nnn` with two `\CDR_tag_provide:` helper.

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

```
\cs_new:Npn \CDR_tag_provide_from_clist:n #1 {
  \exp_args:No
  \regex_extract_once:nnNT {
    ^\c_CDR_tag/([^/]*)(?:/(.*)$)?
  } { #1 } \l_CDR_seq {
    \tl_set:Nx \l_CDR_tl { \seq_item:Nn \l_CDR_seq 3 }
    \exp_args:Nx
    \clist_map_inline:nn {
      \seq_item:Nn \l_CDR_seq 2
    } {
      \exp_args:NV
      \keys_if_exist:nnF \c_CDR_tag { ##1 } {
        \keys_define:on \c_CDR_tag {
          ##1 .inherit:n = \c_CDR_tag / default,
          ##1 .code:n = \CDR_keys_set:nn { ##1 } { ####1 },
          ##1 .value_required:n = true,
        }
      }
      \exp_args:NoV
      \keys_if_exist:nnF { \c_CDR_tag / ##1 } \l_CDR_tl {
        \exp_args:NnV
        \regex_match:nnT {
          ^[^/]*\sengine\soptions$
        } \l_CDR_tl {
          \keys_define:oo { \c_CDR_tag / ##1 } {
            \l_CDR_tl .code:n = \exp_not:n { \CDR_tag_set:n { ####1 } },
            \l_CDR_tl .value_required:n = true,
          }
        }
      }
    }
  }
}
\cs_new:Npn \CDR_tag_provide_from_clist:nn #1 #2 {
  \CDR_tag_provide_from_clist:n { #1 }
}
\cs_new:Npn \CDR_tag_provide_from_keyval:n {
  \keys_parse:nnn {
    \CDR_tag_provide_from_clist:n
  } {
    \CDR_tag_provide_from_clist:nn
  }
}
\cs_generate_variant:Nn \CDR_tag_provide_from_keyval:n { V }
```

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

\g_CDR_has_pygment_bool

Whether pygment is available.

```
\bool_new:N \g_CDR_has_pygment_bool
\sys_get_shell:nnN {which~pygmentize} {} \l_CDR_tl
\bool_set:Nn \g_CDR_has_pygment_bool {
  \exp_args:NV
  \str_if_in_p:nn \l_CDR_tl { pygmensize }
}
```

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

\CDR_has_pygment:TF ⋆

\CDR_has_pygment:TF {⟨*true code*⟩} {⟨*false code*⟩}

Execute ⟨*true code*⟩ when pygment is available, ⟨*false code*⟩ otherwise.

```
\prg_new_conditional:Nnn \CDR_has_pygment: { T, F, TF } {
  \bool_if:NTF \g_CDR_has_pygment_bool {
    \prg_return_false:
  } {
    \prg_return_true:
  }
}
```

\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
  }
}
```

\g_CDR_block_bool

```
\bool_new:N \g_CDR_block_bool
```

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

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

Execute ⟨*true code*⟩ when inside a code block, ⟨*false code*⟩ otherwise.

```
\prg_new_conditional:Nnn \CDR_if_block: { T, F, TF } {
  \bool_if:NTF \g_CDR_block_bool {
    \prog_return_true:
  } {
    \prog_return_false:
  }
}
```

`\CDR_process_record:` Record the current line or not.

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

9 l3keys modules

9.1 `\CDR_tag:n{default}` l3keys module

```
\keys_define:on { \c_CDR_tag / default } {
```

Keys are:

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

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

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

```
  pygment .choices:nn =
    { false, true, {} } { \CDR_tag_boolean_set: },
```

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

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

- **`post processor=`⟨*command*⟩** the command for pygment 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:,
```

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

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

- **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,
```

- **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,
```

- ⟨*engine name*⟩ **engine options=⟨*engine options*⟩** to specify the options for the named engine,
- **__initialize** to initialize storage properly. We cannot use `.initial:n` actions because the `\l_keys_path_str` is not set up properly.

```
  __initialize .meta:n = {
    lang = tex,
    pygment = \CDR_has_pygment:TF { true } { false },
    style = default,
    post~processor = ,
    parskip = \the\parskip,
    engine = default,
    default~engine~options = ,
  },
  __initialize .value_forbidden:n = true,

}
```

9.2 `\CDR_tag:n{default.block}` l3keys module

```
\keys_define:xn { \CDR_tag:n { default.block } } {
```

Known keys include:

- **show tags[=true|false]** to enable/disable the display of the code chunks tags. Initially `true`.
- **tags=⟨*tag name comma list*⟩** to export and display.

```
  tags .code:n = {
    \clist_set:Nn \l_CDR_tags_clist { #1 }
    \clist_remove_duplicates:N \l_CDR_tags_clist
    \clist_remove_all:N \l_CDR_tags_clist {}
    \tl_set:Nx \l_CDR_tags_tl { \clist_use:Nn \l_CDR_tags_clist { , } }
    \exp_args:NV
    \CDR_tag_set:n \l_CDR_tags_tl
  },
```

```
show~tags .choices:nn =
  { false, true, {} } { \CDR_tag_boolean_set: },
```

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 .choices:nn =
  { false, true, {} } { \CDR_tag_boolean_set: },
```

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

```
use~margin .choices:nn =
  { false, true, {} } { \CDR_tag_boolean_set: },
```

tags format=⟨*format*⟩ , where ⟨*format*⟩ is used to display the tag names (mainly font, size and color),

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

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

```
blockskip .code:n = \CDR_tag_set:,
blockskip .required_value:n = true,
```

__initialize the separation with the surrounding text. Initially `\topsep`.

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

}
```

10 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.

THERE IS A BIG PROBLEM WITH THE l3keys `.initial:n` design when I am relying on `\l_keys_path_str` to save values. It is based on the module path used for the definition.

10.1 \CDR_tag:n{__fancyvrb.block} l3keys module

Block specific options.

```
\keys_define:on { \CDR_tag:n { __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:
  },
```

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: },
```

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,
```

labelposition=none|topline|bottomline|all position where to print the label(s) when defined. When options happen to be contradictory, like `frame=topline`, `labelposition=bottomli` 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: },
```

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 { ^(+|-)?\d+$ } { #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 .choices:nn =
  { false, true, {} } { \CDR_tag_boolean_set: },
```

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,
```

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

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

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

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

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

```
xrightmargin .code: = \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 .choices:nn =
  { false, true, {} } { \CDR_tag_boolean_set: },
```

🛑 **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: = \CDR_tag_set:,
hfuzz .value_required:n = true,
```

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

```
samepage .choices:nn =
  { false, true, {} } { \CDR_tag_boolean_set: },
```

✅ **__initialize** Initialization.

```
__initialize .meta:n = {
  commentchar = ,
  gobble = 0,
  frame = none,
  label = ,
  labelposition = none,% auto?
  numbers = left,
  numbersep = \hspace{1ex},
  firstnumber = auto,
  stepnumber = 1,
  numberblanklines = true,
  firstline = ,
  lastline = ,
```

```
    baselinestretch = auto,
    resetmargins = true,
    xleftmargin = 0pt,
    xrightmargin = 0pt,
    hfuzz = 2pt,
    samepage = false,
  },
  __initialize .value_forbidden:n = true,

}
```

10.2 \CDR_tag:n{__fancyvrb} l3keys module

```
\keys_define:on { \CDR_tag:n { __fancyvrb } } {
```

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

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

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

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

```
  showspaces .choices:nn =
    { false, true, {} } { \CDR_tag_boolean_set: },
```

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

```
  showtabs .choices:nn =
    { false, true, {} } { \CDR_tag_boolean_set: },
```

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

```
obeytabs .choices:nn =
  { false, true, {} } { \CDR_tag_boolean_set: },
```

🛑 **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: = \CDR_tag_set:,
defineactive .value_required:n = true,
```

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

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

✅ **__initialize** Initialization.

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

}
```

10.3 `\CDR_tag:n{__fancyvrb.all}` l3keys module

Options available when pygment is not used.

```
\keys_define:on { \CDR_tag:n { __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 pygment mode.

```
commandchars .code: = \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 pygment mode.

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

✅ **__initialize** Initialization.

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

```
}
```

10.4 pygment options

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

🛑 **style=⟨*name*⟩** the pygment 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.

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

🚫 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`.

- **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`.

11 \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.

11.1 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_index_tl = 1 {
      \cs_set_eq:NN \CDR_if_only_description:TF \use_ii:nn
      \cs_set_eq:NN \CDR_if_only_description:F \use:n
      \cs_set_eq:NN \CDR_if_only_description:T \use_none:n
    } {
      \cs_set_eq:NN \CDR_if_only_description:TF \use_i:nn
      \cs_set_eq:NN \CDR_if_only_description:F \use_none:n
      \cs_set_eq:NN \CDR_if_only_description:T \use:n
    }
  },
  only~description .initial:n = false

}
```

11.2 Branching

`\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.

11.3 Implementation

\CDR_check_unknown:N

\CDR_check_unknown:N {⟨*tl variable*⟩}

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

```
\exp_args_generate:n { nnV }
\cs_new:Npn \CDR_check_unknown:N #1 {
  \tl_if_empty:NF #1 {
    \cs_set:Npn \CDR_check_unknown:n ##1 {
      \PackageWarning
        { coder }
        { Unknow~key~`##1' }
    }
    \cs_set:Npn \CDR_check_unknown:nn ##1 ##2 {
      \CDR_check_unknown:n { ##1 }
    }
    \exp_args:NnnV
    \keys_parse:nnn {
      \CDR_check_unknown:n
    } {
      \CDR_check_unknown:nn
    } #1
  }
}
```

```
\cs_new:Npn \CDR_keys_set_known:nnN #1 #2 {
  \keys_set_known:nnnN { #1 } { #2 } { #1 }
}
\clist_map_inline:nn {nV,VV,oV,Vn,Vx} {
  \cs_generate_variant:Nn \CDR_keys_set_known:nnN { #1 }
}
```

```
\NewDocumentCommand \CDRSet { m } {
  \CDR_keys_set:nn { default.block } { __initialize }
  \CDR_keys_set:nn { default.code  } } __initialize }
  \CDR_keys_set:nn { default       } } __initialize }
  \keys_set_known:nnnN { CDR@set } { #1 } { CDR@set } \l_CDR_tl
  \CDR_keys_set_known:nVN { default.block } \l_CDR_tl \l_CDR_tl
  \CDR_keys_set_known:nVN { default.code  } \l_CDR_tl \l_CDR_tl
  \CDR_keys_set_known:nVN { default       } \l_CDR_tl \l_CDR_tl
  \exp_args:NnV
  \keys_set_known:nnnN { CDR@tags } \l_CDR_tl { CDR@tags } \l_CDR_tl
  \CDR_tag_provide_from_keyval:V \l_CDR_tl
  \exp_args:NnV
  \keys_set_known:nnnN { CDR@tags } \l_CDR_tl { CDR@tags } \l_CDR_tl
  \CDR_check_unknown:N \l_CDR_tl
}
```

12 \CDRExport

\CDRExport `\CDRExport {⟨key[=value] controls⟩}`

The ⟨*key*⟩[=⟨*value*⟩] controls are defined by `CDR@Export` l3keys module.

12.1 Storage

\g_CDR_export_prop Global storage for ⟨*file name*⟩=⟨*file export info*⟩

```
513 \prop_new:N \g_CDR_export_prop
```

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

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

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

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

\l_CDR_tags_clist Used by `CDR@Export` l3keys module to temporarily store tags during the export declaration.

```
515 \clist_new:N \l_CDR_tags_clist
```

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

\l_CDR_tags_tl Used to share the tags with coder-tool.py.

```
516 \tl_new:N \l_CDR_tags_tl
```

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

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

```
517 \prop_new:N \l_CDR_export_prop
```

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

12.2 `CDR@Export` l3keys module

No initial value is given for every key. An `__initialize` action will set the storage with proper initial values.

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

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

```
519   file .tl_set:N = \l_CDR_file_tl,
520   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:Nn \l_CDR_tags_clist { #1 }
  \clist_remove_duplicates:N \l_CDR_tags_clist
  \clist_remove_all:Nn \l_CDR_tags_clist {}
  \tl_set:Nx \l_CDR_tags_tl { \clist_use:Nn \l_CDR_tags_clist { , } }
  \prop_put:NVV \l_CDR_prop \l_keys_key_str \l_CDR_tags_clist
},
tags .value_required:n = true,
```

`lang` one of the languages pygment is aware of. Initially `tex`.

```
lang .code:n = {
  \prop_put:NVn \l_CDR_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_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_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_prop \l_keys_key_str {
    \int_compare:nNnTF
      \l_keys_index_tl = 1 { false } { true }
  }
},
```

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

```
__initialize .meta:n = {
  __initialize_prop,
  file=,
  tags=,
  lang=tex,
  preamble=,
  postamble=,
  raw=false,
},
__initialize .value_forbidden:n = true,
```

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

```
__initialize_prop .code:n = \prop_clear:N \l_CDR_prop,
__initialize_prop .value_forbidden:n = true,
```

12.3 Implementation

```
\NewDocumentCommand \CDRExport { m } {
  \keys_set:nn { CDR@Export } { __initialize }
  \CDR_keys_set_known:nnN { CDR@Export } { #1 } \l_CDR_tl
  \tl_if_empty:NTF \l_CDR_file_tl {
    \PackageWarning
      { coder }
      { Missing~key~‘file’ }
  } {
    \CDR_tag_provide_from_keyval:V \l_CDR_tl
    \CDR_keys_set_known:nnN { CDR@Export } { #1 } \l_CDR_tl
    \CDR_check_unknown:N \l_CDR_tl
    \prop_put:NnV \l_CDR_prop { file } \l_CDR_file_tl
    \prop_gput:NVV \g_CDR_export_prop \l_CDR_file_tl \l_CDR_prop
```

If a `lang` is given, forwards the declaration to all the tagged chunks.

```
    \prop_get:NnNT \l_CDR_prop { tags } \l_CDR_tags_clist {
      \prop_get:NnNT \l_CDR_prop { lang } \l_CDR_tl {
        \clist_map_inline:Nn \l_CDR_tags_clist {
          \CDR_tag_set:ccV { ##1 } { lang } \l_CDR_tl
        }
      }
    }
  }
}
```

`\CDR_if_truthy:nTF`
`\CDR_if_truthy:xTF`

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

Execute ⟨*true code*⟩ when ⟨*token list*⟩ is a truthy value once expanded, ⟨*false code*⟩ otherwise. A truthy value is a text which leading character is one of “tTyY”.

```
\prg_new_conditional:Nnn \CDR_if_truthy:n { T, F, TF } {
  \regex_match:nnTF { ^[tTyY] } { #1 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
\prg_generate_conditional_variant:Nnn \CDR_if_truthy:n { x } { T, F, TF }
```

Files are created at the end of the typesetting process.

```
\AddToHook { enddocument / end } {
  \prop_map_inline:Nn \g_CDR_export_prop {
    \tl_set:Nn \l_CDR_prop { #2 }
    \str_set:Nx \l_CDR_str {
      \prop_item:Nn \l_CDR_prop { file }
    }
    \lua_now:n { CDR:export_file(’l_CDR_str’) }
    \clist_map_inline:nn {
      tags, raw, preamble, postamble
    } {
      \str_set:Nx \l_CDR_str {
```

```
        \prop_item:Nn \l_CDR_prop { ##1 }
      }
      \lua_now:n {
        CDR:export_file_info('##1','l_CDR_str')
      }
    }
    \lua_now:n { CDR:export_file_complete() }
  }
}
```

13 Creating display engines

13.1 Utilities

\CDR_code_engine:n ⋆
\CDR_code_engine:V ⋆
\CDR_block_engine:n ⋆
\CDR_block_engine:V ⋆

\CDR_code_engine:n {⟨*engine name*⟩}
\CDR_block_engine:n {⟨*engine name*⟩}

\CDR_code_engine:n builds a command sequence name based on ⟨*engine name*⟩. \CDR_block_engine:n builds an environment name based on ⟨*engine name*⟩.

```
\cs_new:Npn \CDR_code_engine:n #1 {
  CDR \c_CDR_slash colored \c_CDR_slash code \c_CDR_slash #1:n
}
\cs_new:Npn \CDR_block_engine:n #1 {
  CDR \c_CDR_slash colored \c_CDR_slash block \c_CDR_slash #1
}
\cs_generate_variant:Nn \CDR_code_engine:n  { V }
\cs_generate_variant:Nn \CDR_block_engine:n { V }
```

\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

\CDRNewCodeEngine
\CDRRenewCodeEngine

\CDRNewCodeEngine {⟨*engine name*⟩}{⟨*engine body*⟩}
\CDRRenewCodeEngine{⟨*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.

```
\NewDocumentCommand \CDRNewCodeEngine { mm } {
  \exp_args:Nx
  \tl_if_empty:nTF { #1 } {
    \PackageWarning
      { coder }
      { The~engine~cannot~be~void. }
  } {
    \cs_new:cpn { \CDR_code_engine:n {#1} } ##1 ##2 {
      \cs_set_eq:NN \CDRGetOption \CDR_tag_get:n
      #2
    }
    \ignorespaces
  }
}
```

```
\NewDocumentCommand \CDRRenewCodeEngine { mm } {
  \exp_args:Nx
  \tl_if_empty:nTF { #1 } {
    \PackageWarning
      { coder }
      { The~engine~cannot~be~void. }
      \use_none:n
  } {
    \cs_if_exist:cTF { \CDR_code_engine:n { #1 } } {
      \cs_set:cpn { \CDR_code_engine:n { #1 } } ##1 ##2 {
        \cs_set_eq:NN \CDRGetOption \CDR_tag_get:n
        #2
      }
    } {
      \PackageWarning
        { coder }
        { No~code~engine~#1.}
    }
    \ignorespaces
  }
}
```

`\CDRNewBlockEngine`
`\CDRRenewBlockEngine`

`\CDRNewBlockEngine` {⟨*engine name*⟩} {⟨*begin instructions*⟩} {⟨*end instructions*⟩}
`\CDRRenewBlockEngine` {⟨*engine name*⟩} {⟨*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 list of instructions which may refer to the unique argument as `#1`, which is the value given to `CDRBlock` environment for key ⟨*engine name*⟩ `engine options`. Various options are available with the `\CDRGetOption` function. *Implementation detail*: the third argument is parsed by `\NewDocumentEnvironment`.

```
\NewDocumentCommand \CDRNewBlockEngine { mm } {
  \NewDocumentEnvironment { \CDR_block_engine:n { #1 } } { m } {
    \cs_set_eq:NN \CDRGetOption \CDR_tag_get:n
    #2
  }
}
```

```
\NewDocumentCommand \CDRRenewBlockEngine { mm } {
  \tl_if_empty:nTF { #1 } {
    \PackageWarning
      { coder }
      { The~engine~cannot~be~void. }
      \use_none:n
  } {
    \RenewDocumentEnvironment { \CDR_block_engine:n { #1 } } { m } {
      \cs_set_eq:NN \CDRGetOption \CDR_tag_get:n
      #2
    }
  }
}
```

13.3 Conditionals

`\CDR_has_code_engine:nTF` ⋆ `\CDR_has_code_engine:nTF {⟨engine name⟩} {⟨true code⟩} {⟨false code⟩}`

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

```
\prg_new_conditional:Nnn \CDR_has_code_engine:n { T, F, TF } {
  \cs_if_exist:cTF { \CDR_code_engine:n { #1 } } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

`\CDR_has_block_engine:nTF` ⋆ `\CDR_has_block_engine:n {⟨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 \CDR_has_block_engine:n { T, F, TF } {
  \cs_if_exist:cTF { \CDR_block_engine:n { #1 } } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

13.4 Default code engine

The default code engine does nothing.

```
\CDRNewCodeEngine { default } { } { }
```

13.5 Default block engine

The default block engine does nothing.

```
\CDRNewBlockEngine { default } { } { }
```

14 \CDRCode function

14.1 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* **??**.)

14.2 CDR@Code l3keys module

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

```
\keys_define:nn { CDR@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,

}
```

14.3 Implementation

\CDRCode `\CDRCode{⟨key[=value]⟩}⟨delimiter⟩⟨code⟩⟨same delimiter⟩`

```
\NewDocumentCommand \CDRCode { mm } {
  \group_begin:
    \keys_define:Vx \c_CDR_tag {
      __local .inherit:n = {
        CDR@Code,
        \CDR_tag:n { default.code },
        \CDR_tag:n { default },
        \CDR_tag:n { __fancyvrb },
      },
    }
    \keys_set:nn { CDR@Code } { __initialize }
    \keys_set:xn { \CDR_tag:n { default.code } } { __initialize }
    \keys_set:xn { \CDR_tag:n { default      } } { __initialize }
    \keys_set:xn { \CDR_tag:n { __fancyvrb   } } { __initialize }
    \str_set:No \l_CDR_str { \CDR_tag:n { __local } }
    \CDR_keys_set_known:VnN \l_CDR_str { #1 } \l_CDR_tl
    \CDR_tag_provide_from_keyval:V \l_CDR_tl
    \CDR_keys_set_known:VnN \l_CDR_str { #1 } \l_CDR_tl
    \CDR_check_unknown:N \l_CDR_tl
    \DefineShortVerb { #2 }
    \exp_args:Nnx
    \CDR_tag_inherit:nn { __local } {
      \tl_if_empty:NF \l_CDR_tag_tl { \l_CDR_tag_tl, }
      default.code,
      default,
      __fancyvrb,
```

```
    }
    \CDR_to_lua:
    \exp_args:Nx \label { \CDR_tag_get:n {reflabel} }
    \SaveVerb [
      aftersave = {
        \UndefineShortVerb { #2 }
        \lua_now:n {CDR:process_code('FV@SV@CDR@Code')}
        \group_end:
      }
    ] { CDR@Code }
}
```

\CDR_to_lua: \CDR_to_lua:

Retrieve info from the tree storage and forwards to `lua`.

```
\cs_new:Npn \CDR_to_lua: {
  \lua_now:n { CDR:options_reset() }
  \seq_map_inline:Nn \g_CDR_tag_path_seq {
    \CDR_tag_get:nNT { ##1 } \l_CDR_tl {
      \str_set:Nx \l_CDR_str { \l_CDR_tl }
      \lua_now:n { CDR:option_add('##1','l_CDR_str') }
    }
  }
}
```

15 CDRBlock environment

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

15.1 Storage

\l_CDR_block_prop

```
\prop_new:N \l_CDR_block_prop
```

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

15.2 CDR@Block l3keys module

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

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

ignore[=true|false] to ignore this code chunk.

```
  ignore .choices:nn =
    { false, true, {} } { \CDR_tag_boolean_set: },
  ignore .default:n = true,
```

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

```
  test .choices:nn =
    { false, true, {} } { \CDR_tag_boolean_set: },
  test .default:n = true,
```

engine options=⟨*engine options*⟩ exact options forwarded to the engine. Normally, options are appended to the default ones, assuming a key–value interface.

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

__initialize initialize

```
  __initialize .meta:n = {
    tags = ,
    ignore = false,
    test= false,
  },
  __initialize .value_forbidden:n = true,

}
```

15.3 Context

Inside the CDRBlock environments, some local variables are available:

\l_CDR_tags_clist

15.4 Implementation

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

```
\cs_set_eq:NN \CDR@ListProcessLine@i   \FV@ListProcessLine@i
\cs_set_eq:NN \CDR@ListProcessLine@ii  \FV@ListProcessLine@ii
\cs_set_eq:NN \CDR@ListProcessLine@iii \FV@ListProcessLine@iii
\cs_set_eq:NN \CDR@ListProcessLine@iv  \FV@ListProcessLine@iv
\cs_new:Npn \CDR_record_line:n #1 {
  \tl_set:Nn \l_CDR_tl { #1 }
  \lua_now:n {CDR:record_line('l_CDR_tl', 'l_CDR_tags_tl')}
}

\def\FVB@CDRBlock #1 {
  \@bsphack
  \group_begin:
  \keys_define:Vx \c_CDR_tag {
    __local .inherit:n = {
      CDR@block,
      \CDR_tag:n { default.block },
      \CDR_tag:n { default },
      \CDR_tag:n { __fancyvrb.block },
      \CDR_tag:n { __fancyvrb },
    }
  }
  \keys_set:nn { CDR@block }            { __initialize }
```

```
\CDR_keys_set:nn { default.block    } { __initialize }
\CDR_keys_set:nn { default          } { __initialize }
\CDR_keys_set:nn { __fancyvrb.block } { __initialize }
\CDR_keys_set:nn { __fancyvrb       } { __initialize }
\CDR_keys_set:nn { __fancyvrb.all   } { __initialize }
\CDR_keys_set_unknown:nnN { __local } { #1 } \l_CDR_tl
```

`\l_CDR_bool` is `true` iff one of the tags needs pygment.

```
\clist_if_empty:NT \l_CDR_tags_clist {
  \CDR_tag_get:nnN { default.block } { tags } \l_CDR_tags_clist
  \clist_if_empty:NT \l_CDR_tags_clist {
    \PackageWarning
      { coder }
      { No~(default)~tags~provided. }
  }
  \tl_set:Nx \l_CDR_tags_tl { \clist_use:Nn \l_CDR_tags_clist { , } }
}
\bool_set_false:N \l_CDR_bool
\clist_map_inline:Nn \l_CDR_tags_clist {
  \CDR_if_truthy:xT { \CDR_tag_get:nn { ##1 } { pygment } } {
    \clist_map_break:n { \bool_set_true:N \l_CDR_bool }
  }
}
\bool_if:NF \l_CDR_bool {
  \keys_define:Vx \c_CDR_tag {
    __local .inherit:n = {
      \CDR_tag:n { __fancyvrb.all },
    }
  }
  \CDR_keys_set_unknown:nVN { __local } \l_CDR_tl  \l_CDR_tl
}
\CDR_check_unknown:N \l_CDR_tl
\keys_define:Vx \c_CDR_tag_get {
  __local .inherit:n = {
    \clist_map_inline:Nn \l_CDR_tags_clist {
      \CDR_tag:n { ##1 },
    }
    \CDR_tag:n { default.block },
    \CDR_tag:n { default },
    \CDR_tag:n { __fancyvrb.block },
    \CDR_tag:n { __fancyvrb },
    \bool_if:NF \l_CDR_bool {
      \CDR_tag:n { __fancyvrb.all },
    }
  }
}
\CDR_tag_get:nN {reflabel} \l_CDR_tl
\exp_args:NV \label \l_CDR_tl
\tl_if_empty:NF \l_CDR_tags_tl {
  \lua_now:n { CDR:record_new('l_CDR_tags_tl') }
  \cs_set:Npn \FV@ListProcessLine@i ##1 {
    \CDR_record_line:n { ##1 }
    \CDR@ListProcessLine@i { ##1 }
  }
```

```
    \cs_set:Npn \FV@ListProcessLine@ii ##1 {
      \CDR_record_line:n { ##1 }
      \CDR@ListProcessLine@ii { ##1 }
    }
    \cs_set:Npn \FV@ListProcessLine@iii ##1 {
      \CDR_record_line:n { ##1 }
      \CDR@ListProcessLine@iii { ##1 }
    }
    \cs_set:Npn \FV@ListProcessLine@iv ##1 {
      \CDR_record_line:n { ##1 }
      \CDR@ListProcessLine@iv { ##1 }
    }
  }
  \CDR_tag_get:nNF { engine } \l_CDR_engine_tl {
    \tl_set:Nn \l_CDR_engine_tl { default }
  }
  \tl_put_right:Nx \l_CDR_tl { ,
  \CDR_tag_get:xNF { \l_CDR_engine_tl~engine~options } \l_CDR_tl {
    \tl_clear:N \l_CDR_tl
  }
  \exp_args:NnV
  \begin { \CDR_block_engine:V \l_CDR_engine_tl } \l_CDR_tl
  \FV@VerbatimBegin
  \FV@Scan
}
\def\FVE@CDRBlock{
  \FV@VerbatimEnd
  \end { \CDR_block_engine:V \l_CDR_engine_tl }
  \group_end:
  \@esphack
}
\DefineVerbatimEnvironment{CDRBlock}{CDRBlock}{}

```

16 The `CDR@Pyg@Verbatim` environment

This is the environment wrapping the pygmentized code when in block mode. It is the sole content of the various `*.pyg.tex` files.

```
\def\FVB@CDR@Pyg@Verbatim #1 {
  \group_begin:
  \FV@VerbatimBegin
  \FV@Scan
}
\def\FVE@CDR@Pyg@Verbatim{
  \FV@VerbatimEnd
  \group_end:
}
\DefineVerbatimEnvironment{CDR@Pyg@Verbatim}{CDR@Pyg@Verbatim}{}

```

17 More

\CDR_if_record:*TF* ⋆ \CDR_if_record:TF {⟨*true code*⟩} {⟨*false code*⟩}

Execute ⟨*true code*⟩ when code should be recorded, ⟨*false code*⟩ otherwise. The code should be recorded for the CDRBlock environment when there is a non empty list of tags and pygment is used. *Implementation details*: we assume that if \l_CDR_tags_clist is not empty then we are in a CDRBlock environment.

```
\prg_new_conditional:Nnn \CDR_if_record: { T, F, TF } {
  \clist_if_empty:NTF \l_CDR_tags_clist {
    \prg_return_false:
  } {
    \CDR_if_use_pygment:TF {
      \prg_return_true:
    } {
      \prg_return_false:
    }
  }
}
```

```
\cs_new:Npn \CDR_process_record: {
  \tl_put_right:Nx \l_CDR_recorded_tl { \the\verbatim@line \iow_newline: }
  \group_begin:
  \tl_set:Nx \l_tmpa_tl { \the\verbatim@line }
  \lua_now:e {CDR.records.append([===[\l_tmpa_tl]===])}
  \group_end:
}
```

CDR \begin{⟨*CDR*⟩} ... \end{⟨*CDR*⟩}

Private environment.

```
\newenvironment{CDR}{
  \def \verbatim@processline {
    \group_begin:
    \CDR_processline_code_append:
    \group_end:
  }
%  \CDR_if_show_code:T {
%    \CDR_if_use_minted:TF {
%      \Needspace* { 2\baselineskip }
%    } {
%      \frenchspacing\@vobeyspaces
%    }
%  }
} {
  \CDR:nNTF { lang } \l_tmpa_tl {
    \tl_if_empty:NT \l_tmpa_tl {
      \clist_map_inline:Nn \l_CDR_clist {
        \CDR:nnNT { ##1 } { lang } \l_tmpa_tl {
          \tl_if_empty:NF \l_tmpa_tl {
            \clist_map_break:
          }
```

```
        }
      }
      \tl_if_empty:NT \l_tmpa_tl {
        \tl_set:Nn \l_tmpa_tl { tex }
      }
    }
  } {
    \tl_set:Nn \l_tmpa_tl { tex }
  }
% NO WAY
  \clist_map_inline:Nn \l_CDR_clist {
    \CDR_gput:nnV { ##1 } { lang } \l_tmpa_tl
  }
}
```

CDR.M `\begin{⟨CDR.M⟩} ... \end{⟨CDR.N⟩}`

Private environment when `minted`.

```
\newenvironment{CDR_M}{
  \setkeys { FV } { firstnumber=last, }
  \clist_if_empty:NTF \l_CDR_clist {
    \exp_args:Nnx \setkeys { FV } {
      firstnumber=\CDR_int_use:n { },
  } } {
    \clist_map_inline:Nn \l_CDR_clist {
      \exp_args:Nnx \setkeys { FV } {
        firstnumber=\CDR_int_use:n { ##1 },
      }
      \clist_map_break:
  } }
  \iow_open:Nn \minted@code { \jobname.pyg }
  \tl_set:Nn \l_CDR_line_tl {
    \tl_set:Nx \l_tmpa_tl { \the\verbatim@line }
    \exp_args:NNV \iow_now:Nn \minted@code \l_tmpa_tl
  }
} {
  \CDR_if_show_code:T {
    \CDR_if_use_minted:TF {
      \iow_close:N \minted@code
      \vspace* { \dimexpr -\topsep-\parskip }
      \tl_if_empty:NF \l_CDR_info_tl {
        \tl_use:N \l_CDR_info_tl
        \vspace* { \dimexpr -\topsep-\parskip-\baselineskip }
        \par\noindent
      }
      \exp_args:NV \minted@pygmentize \l_tmpa_tl
      \DeleteFile { \jobname.pyg }
      \vspace* { \dimexpr -\topsep -\partopsep }
    } {
      \@esphack
    }
  }
}
```

CDR.P `\begin{⟨CDR.P⟩} ... \end{⟨CDR.P⟩}`

Private pseudo environment. This is just a practical way of declaring balanced actions.

```
\newenvironment{CDR_P}{
  \if_mode_vertical:
    \noindent
  \else
    \vspace*{ \topsep }
    \par\noindent
  \fi
  \CDR_gset_chunks:
  \tl_if_empty:NTF \g_CDR_chunks_tl {
    \CDR_if:nTF {show_lineno} {
      \CDR_if_use_margin:TF {
```

No chunk name, line numbers in the margin

```
        \tl_set:Nn \l_CDR_info_tl {
          \hbox_overlap_left:n {
            \CDR:n { format/code }
            {
              \CDR:n { format/name }
              \CDR:n { format/lineno }
              \clist_if_empty:NTF \l_CDR_clist {
                \CDR_int_use:n { }
              } {
                \clist_map_inline:Nn \l_CDR_clist {
                  \CDR_int_use:n { ##1 }
                  \clist_map_break:
                }
              }
            }
            \hspace*{1ex}
          }
        }
      } {
```

No chunk name, line numbers not in the margin

```
        \tl_set:Nn \l_CDR_info_tl {
          {
            \CDR:n { format/code }
            {
              \CDR:n { format/name }
              \CDR:n { format/lineno }
              \hspace*{3ex}
              \hbox_overlap_left:n {
                \clist_if_empty:NTF \l_CDR_clist {
                  \CDR_int_use:n { }
                } {
                  \clist_map_inline:Nn \l_CDR_clist {
                    \CDR_int_use:n { ##1 }
                    \clist_map_break:
                  }
                }
```

```
              }
              \hspace*{1ex}
            }
          }
        }
      }
    } {
```

No chunk name, no line numbers

```
      \tl_clear:N \l_CDR_info_tl
    }
  } {
    \CDR_if:nTF {show_lineno} {
```

Chunk names, line numbers, in the margin

```
      \tl_set:Nn \l_CDR_info_tl {
        \hbox_overlap_left:n {
          \CDR:n { format/code }
          {
            \CDR:n { format/name }
            \g_CDR_chunks_tl :
            \hspace*{1ex}
            \CDR:n { format/lineno }
            \clist_map_inline:Nn \l_CDR_clist {
              \CDR_int_use:n { ####1 }
              \clist_map_break:
            }
          }
          \hspace*{1ex}
        }
        \tl_set:Nn \l_CDR_info_tl {
          \hbox_overlap_left:n {
            \CDR:n { format/code }
            {
              \CDR:n { format/name }
              \CDR:n { format/lineno }
              \clist_map_inline:Nn \l_CDR_clist {
                \CDR_int_use:n { ####1 }
                \clist_map_break:
              }
            }
            \hspace*{1ex}
          }
        }
      }
    } {
```

Chunk names, no line numbers, in the margin

```
      \tl_set:Nn \l_CDR_info_tl {
        \hbox_overlap_left:n {
          \CDR:n { format/code }
          {
            \CDR:n { format/name }
```

```
            \g_CDR_chunks_tl :
          }
          \hspace*{1ex}
        }
        \tl_clear:N \l_CDR_info_tl
      }
    }
  }
  \CDR_if_use_minted:F {
    \tl_set:Nn \l_CDR_line_tl {
      \noindent
      \hbox_to_wd:nn { \textwidth } {
        \tl_use:N \l_CDR_info_tl
        \CDR:n { format/code }
        \the\verbatim@line
        \hfill
      }
      \par
    }
    \@bsphack
  }
} {
  \vspace*{ \topsep }
  \par
  \@esphack
}
```

18 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: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: { 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* **??**.)

19 minted and pygment

\g_CDR_minted_on_bool Whether minted is available, initially set to false.

```
\bool_new:N \g_CDR_minted_on_bool
```

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

\g_CDR_use_minted_bool Whether minted is used, initially set to false.

```
\bool_new:N \g_CDR_use_minted_bool
```

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

\CDR_if_use_minted:*TF* \CDR_if_use_minted:TF {⟨*true code*⟩} {⟨*false code*⟩}

Execute ⟨*true code*⟩ when using minted, ⟨*false code*⟩ otherwise.

```
\prg_new_conditional:Nnn \CDR_if_use_minted: { T, F, TF } {
  \bool_if:NTF \g_CDR_use_minted_bool
    { \prg_return_true:  }
    { \prg_return_false: }
}
```

_CDR_minted_on: _CDR_minted_on:

Private function. During the preamble, loads minted, sets \g_CDR_minted_on_bool to true and prepares pygment processing.

```
\cs_set:Npn \_CDR_minted_on: {
  \bool_gset_true:N \g_CDR_minted_on_bool
  \RequirePackage{minted}
  \setkeys{ minted@opt@g } { linenos=false }
  \minted@def@opt{post~processor}
  \minted@def@opt{post~processor~args}
  \pretocmd\minted@inputpyg{
    \CDR@postprocesspyg {\minted@outputdir\minted@infile}
  }{}{\fail}
```

In the execution context of \minted@inputpyg,

#1 is the name of the python script, e.g., "process.py"

#2 is the input ".pygtex" file "\minted@outputdir\minted@infile"

#3 are more args passed to the python script, possibly empty

```
  \newcommand{\CDR@postprocesspyg}[1]{%
    \group_begin:
    \tl_set:Nx \l_tmpa_tl {\CDR:n { post_processor } }
    \tl_if_empty:NF \l_tmpa_tl {
```

Execute ‘python3 <script.py> <file.pygtex> <more_args>‘

```
      \tl_set:Nx \l_tmpb_tl {\CDR:n { post_processor_args } }
      \exp_args:Nx
      \sys_shell_now:n {
        python3\space
        \l_tmpa_tl\space
        ##1\space
        \l_tmpb_tl
      }
    }
    \group_end:
  }
}
```

```
%\AddToHook { begindocument / end } {
%  \cs_set_eq:NN \_CDR_minted_on: \prg_do_nothing:
%}
```

Utilities to setup `pygment` post processing. The `pygment` post processor marks some code with `\CDREmph`.

```
\ProvideDocumentCommand{\CDREmph}{m}{\textcolor{red}{#1}}
```

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

Store the content of ⟨*file name*⟩ into the variable ⟨*variable*⟩.

```
\DeclareDocumentCommand \CDRPreamble { m m } {
  \msg_info:nnn
    { coder }
    { :n }
    { Reading~preamble~from~file~"#2". }
  \group_begin:
  \tl_set:Nn \l_tmpa_tl { #2 }
  \exp_args:NNNx
  \group_end:
  \tl_set:Nx #1 { \directlua{CDR.print_file_content('l_tmpa_tl')} }
}
```

20 Section separators

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

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

21 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:
```

22 Finale

```
\AddToHook { cmd/FancyVerbFormatLine/before } {
  \CDR_line_number:
}
\AddToHook { shipout/before } {
  \tl_gclear:N \g_CDR_chunks_tl
}
\CDRSet {}

% ==========================================================
% Auxiliary:
%   finding the widest string in a comma
%   separated list of strings delimited by parenthesis
% ==========================================================

% arguments:
% #1) text: a comma separeted list of strings
% #2) formatter: a macro to format each string
% #3) dimension: will hold the result

```

```
\cs_new:Npn \CDRWidest (#1) #2 #3 {
  \group_begin:
  \dim_set:Nn #3 { 0pt }
  \clist_map_inline:nn { #1 } {
    \hbox_set:Nn \l_tmpa_box { #2{##1} }
    \dim_set:Nn \l_tmpa_dim { \dim_eval:n { \box_wd:N \l_tmpa_box } }
    \dim_compare:nNnT { #3 } < { \l_tmpa_dim } {
      \dim_set_eq:NN #3 \l_tmpa_dim
    }
  }
  \exp_args:NNNV
  \group_end:
  \dim_set:Nn #3 #3
}
\ExplSyntaxOff

```

23 pygmentex implementation

```
% ==========================================================
% fancyvrb new commands to append to a file
% ==========================================================

% See http://tex.stackexchange.com/questions/47462/inputenc-error-with-unicode-chars-and-verbati

\ExplSyntaxOn

\seq_new:N \l_CDR_records_seq

\long\def\unexpanded@write#1#2{\write#1{\unexpanded{#2}}}

\def\CDRAppend{\FV@Environment{}{CDRAppend}}

\def\FVB@CDRAppend#1{%
  \@bsphack
  \begingroup
    \seq_clear:N \l_CDR_records_seq
    \FV@UseKeyValues
    \FV@DefineWhiteSpace
    \def\FV@Space{\space}%
    \FV@DefineTabOut
    \def\FV@ProcessLine{%##1
      \seq_put_right:Nn \l_CDR_records_seq { ##1 }%
      \immediate\unexpanded@write#1%{##1}
    }%
    \let\FV@FontScanPrep\relax
    \let\@noligs\relax
    \FV@Scan
}
\def\FVE@CDRAppend{
  \seq_use:Nn \l_CDR_records_seq /
  \endgroup
  \@esphack
```

```
}
\DefineVerbatimEnvironment{CDRAppend}{CDRAppend}{}

\DeclareDocumentEnvironment { Inline } { m } {
  \clist_clear:N \l_CDR_clist
  \keys_set:nn { CDR_code } { #1 }
  \clist_map_inline:Nn \l_CDR_clist {
    \CDR_int_if_exist:nF { ##1 } {
      \CDR_int_new:nn { ##1 } { 1 }
      \seq_new:c { g/CDR/chunks/##1 }
    }
  }
  \CDR_if:nT {reset} {
    \CDR_clist_map_inline:Nnn \l_CDR_clist {
      \CDR_int_gset:nn { } 1
    } {
      \CDR_int_gset:nn { ##1 } 1
    }
  }
  \tl_clear:N \l_CDR_code_name_tl
  \clist_map_inline:Nn \l_CDR_clist {
    \prop_concat:ccc
      {g/CDR/Code/}
      {g/CDR/Code/##1/}
      {g/CDR/Code/}
    \tl_set:Nn \l_CDR_code_name_tl { ##1 }
    \clist_map_break:
  }
  \int_gset:Nn \g_CDR_int
    { \CDR_int_use:n { \l_CDR_code_name_tl } }
  \tl_clear:N \l_CDR_info_tl
  \tl_clear:N \l_CDR_name_tl
  \tl_clear:N \l_CDR_recorded_tl
  \tl_clear:N \l_CDR_chunks_tl
  \cs_set:Npn \verbatim@processline {
    \CDR_process_record:
  }
  \CDR_if_show_code:TF {
    \exp_args:NNx
    \skip_set:Nn \parskip { \CDR:n { parskip } }
    \clist_if_empty:NTF \l_CDR_clist {
      \tl_gclear:N \g_CDR_chunks_tl
    } {
      \clist_set_eq:NN \l_tmpa_clist \l_CDR_clist
      \clist_sort:Nn \l_tmpa_clist {
        \str_compare:nNnTF { ##1 } > { ##2 } {
          \sort_return_swapped:
        } {
          \sort_return_same:
        }
      }
      \tl_set:Nx \l_tmpa_tl { \clist_use:Nn \l_tmpa_clist , }
      \CDR_if:nT {show_name} {
        \CDR_if:nT {use_margin} {
```

```
          \CDR_if:nT {only_top} {
            \tl_if_eq:NNT \l_tmpa_tl \g_CDR_chunks_tl {
              \tl_gset_eq:NN \g_CDR_chunks_tl \l_tmpa_tl
              \tl_clear:N \l_tmpa_tl
            }
          }
          \tl_if_empty:NF \l_tmpa_tl {
            \tl_set:Nx \l_CDR_chunks_tl {
              \clist_use:Nn \l_CDR_clist ,
            }
            \tl_set:Nn \l_CDR_name_tl {
              {
                \CDR:n { format/name }
                \l_CDR_chunks_tl :
                \hspace*{1ex}
              }
            }
          }
        }
        \tl_if_empty:NF \l_tmpa_tl {
          \tl_gset_eq:NN \g_CDR_chunks_tl \l_tmpa_tl
        }
      }
    }
    \if_mode_vertical:
    \else:
    \par
    \fi:
    \vspace{ \CDR:n { sep } }
    \noindent
    \frenchspacing
    \@vobeyspaces
    \normalfont\ttfamily
    \CDR:n { format/code }
    \hyphenchar\font\m@ne
    \@noligs
    \CDR_if_record:F {
      \cs_set_eq:NN \CDR_process_record: \prg_do_nothing:
    }
    \CDR_if_use_minted:F {
      \CDR_if:nT {show_lineno} {
        \CDR_if:nTF {use_margin} {
          \tl_set:Nn \l_CDR_info_tl {
            \hbox_overlap_left:n {
              {
                \l_CDR_name_tl
                \CDR:n { format/name }
                \CDR:n { format/lineno }
                \int_use:N \g_CDR_int
                \int_gincr:N \g_CDR_int
              }
              \hspace*{1ex}
            }
          }
```

```
        } {
          \tl_set:Nn \l_CDR_info_tl {
            {
              \CDR:n { format/name }
              \CDR:n { format/lineno }
              \hspace*{3ex}
              \hbox_overlap_left:n {
                \int_use:N \g_CDR_int
                \int_gincr:N \g_CDR_int
              }
            }
            \hspace*{1ex}
          }
        }
      }
      \cs_set:Npn \verbatim@processline {
        \CDR_process_record:
        \hspace*{\dimexpr \linewidth-\columnwidth}%
        \hbox_to_wd:nn { \columnwidth } {
          \l_CDR_info_tl
          \the\verbatim@line
          \color{lightgray}\dotfill
        }
        \tl_clear:N \l_CDR_name_tl
        \par\noindent
      }
    }
  } {
    \@bsphack
  }
  \group_begin:
  \g_CDR_hook_tl
  \let \do \@makeother
  \dospecials \catcode `\^^M \active
  \verbatim@start
} {
  \int_gsub:Nn \g_CDR_int {
    \CDR_int_use:n { \l_CDR_code_name_tl }
  }
  \int_compare:nNnT { \g_CDR_int } > { 0 } {
    \CDR_clist_map_inline:Nnn \l_CDR_clist {
      \CDR_int_gadd:nn { } { \g_CDR_int }
    } {
      \CDR_int_gadd:nn { ##1 } { \g_CDR_int }
    }
    \int_gincr:N \g_CDR_code_int
    \tl_set:Nx \l_tmpb_tl { \int_use:N \g_CDR_code_int }
    \clist_map_inline:Nn \l_CDR_clist {
      \seq_gput_right:cV { g/CDR/chunks/##1 } \l_tmpb_tl
    }
    \prop_gput:NVV \g_CDR_code_prop \l_tmpb_tl \l_CDR_recorded_tl
  }
  \group_end:
  \CDR_if_show_code:T {
```

```
  }
  \CDR_if_show_code:TF {
    \CDR_if_use_minted:TF {
      \tl_if_empty:NF \l_CDR_recorded_tl {
        \exp_args:Nnx \setkeys { FV } {
          firstnumber=\CDR_int_use:n { \l_CDR_code_name_tl },
        }
        \iow_open:Nn \minted@code { \jobname.pyg }
        \exp_args:NNV \iow_now:Nn \minted@code \l_CDR_recorded_tl
        \iow_close:N \minted@code
        \vspace* { \dimexpr -\topsep-\parskip }
        \tl_if_empty:NF \l_CDR_info_tl {
          \tl_use:N \l_CDR_info_tl
          \skip_vertical:n { \dimexpr -\topsep-\parskip-\baselineskip }
          \par\noindent
        }
        \exp_args:Nnx \minted@pygmentize { \jobname.pyg } { \CDR:n { lang } }
        %\DeleteFile { \jobname.pyg }
        \skip_vertical:n { -\topsep-\partopsep }
      }
    } {
      \exp_args:Nx \skip_vertical:n { \CDR:n { sep } }
      \noindent
    }
  } {
    \@esphack
  }
}
% ==========================================================
% Main options
% ==========================================================

\newif\ifCDR@left
\newif\ifCDR@right


```

24 Display engines

Inserting code snippets follows one of two modes: run or block. The former is displayed as running text and used by the `\CDRCode` command whereas the latter is displayed as a separate block and used by the `CDRBlock` environment. Both have one single required argument, which is a ⟨*key-value*⟩ configuration list conforming to `CDR_code` l3keys module. The contents is then colorized with the aid of coder-tool.py which will return some code enclosed within an environment created by one of `\CDRNewCodeEngine`, `\CDRRenewCodeEngine`, `\CDRNewBlockEngine`, `\CDRRenewBlockEngine` functions.

24.1 Run mode efbox engine

CDRCallWithOptions ⋆ \CDRCallWithOptions⟨*cs*⟩

Call ⟨*cs*⟩, assuming it has a first optional argument. It will receive the arguments passed to \CDRCode with the options key.

```
\cs_new:Npn \CDRCallWithOptions #1 {
  \exp_last_unbraced:NNx
  #1[\CDR:n { options }]
}
\CDRNewCodeEngine {efbox} {
  \CDRCallWithOptions\efbox{#1}%
}
```

24.2 Block mode default engine

```
\CDRNewBlockEngine {} {
} {
}
```

24.3 options key-value controls

We accept any value because we do not know in advance the real target. Everything is collected in \l_CDR_options_clist.

\l_CDR_options_clist All the ⟨*key[=value] items*⟩ passed as options are collected here. This should be cleared before arguments are parsed.

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

There are 2 ways to collect options:

25 Something else

```

% ==========================================================
% pygmented commands and environments
% ==========================================================


\newcommand\inputpygmented[2][]{%
  \begingroup
    \CDR@process@options{#1}%
    \immediate\write\CDR@outfile{<@@CDR@input@\the\CDR@counter}%
    \immediate\write\CDR@outfile{\exp_args:NV\detokenize\CDR@global@options,\detokenize{#1}}%
    \immediate\write\CDR@outfile{#2}%
    \immediate\write\CDR@outfile{>@@CDR@input@\the\CDR@counter}%
    %
    \csname CDR@snippet@\the\CDR@counter\endcsname
    \global\advance\CDR@counter by 1\relax
  \endgroup
}

```

```
\cs_generate_variant:Nn \exp_last_unbraced:NnNo { NxNo }

\newcommand\CDR@snippet@run[1]{%
  \group_begin:
  \typeout{DEBUG~PY~STYLE:< \CDR:n { style } > }
  \use_c:n { PYstyle }
  \CDR_when:nT { style } {
    \use_c:n { PYstyle \CDR:n { style } }
  }
  \cs_if_exist:cTF {PY} {PYOK} {PYKO}
  \CDR:n {font}
  \CDR@process@more@options{ \CDR:n {engine} }%
  \exp_last_unbraced:NxNo
  \use:c { \CDR:n {engine} } [ \CDRRemainingOptions ]{#1}%
  \group_end:
}

% ERROR: JL undefined \CDR@alllinenos

\ProvideDocumentCommand\captionof{mm}{}
\def\CDR@alllinenos{(0)}

\def\FormatLineNumber#1{{\rmfamily\tiny#1}}

\newdimen\CDR@leftmargin
\newdimen\CDR@linenosep

\def\CDR@lineno@do#1{%
  \CDR@linenosep 0pt%
  \use:c { CDR@ \CDR:n {block_engine} @margin }
  \exp_args:NNx
  \advance \CDR@linenosep { \CDR:n {linenosep} }
  \hbox_overlap_left:n {%
    \FormatLineNumber{#1}%
    \hspace*{\CDR@linenosep}%
  }%
}

\newcommand\CDR@tcbox@more@options{%
  nobeforeafter,%
  tcbox~raise~base,%
  left=0mm,%
  right=0mm,%
  top=0mm,%
  bottom=0mm,%
  boxsep=2pt,%
  arc=1pt,%
  boxrule=0pt,%
  \CDR_options_if_in:nT {colback} {
    colback=\CDR:n {colback}
  }
}

\newcommand\CDR@mdframed@more@options{%
```

```
  leftmargin=\CDR@leftmargin,%
  frametitlerule=true,%
  \CDR_if_in:nT {colback} {
    backgroundcolor=\CDR:n {colback}
  }
}

\newcommand\CDR@tcolorbox@more@options{%
  grow~to~left~by=-\CDR@leftmargin,%
  \CDR_if_in:nNT {colback} {
    colback=\CDR:n {colback}
  }
}

\newcommand\CDR@boite@more@options{%
  leftmargin=\CDR@leftmargin,%
  \ifcsname CDR@opt@colback\endcsname
    colback=\CDR@opt@colback,%
  \fi
}

\newcommand\CDR@mdframed@margin{%
  \advance \CDR@linenosep \mdflength{outerlinewidth}%
  \advance \CDR@linenosep \mdflength{middlelinewidth}%
  \advance \CDR@linenosep \mdflength{innerlinewidth}%
  \advance \CDR@linenosep \mdflength{innerleftmargin}%
}

\newcommand\CDR@tcolorbox@margin{%
  \advance \CDR@linenosep \kvtcb@left@rule
  \advance \CDR@linenosep \kvtcb@leftupper
  \advance \CDR@linenosep \kvtcb@boxsep
}

\newcommand\CDR@boite@margin{%
  \advance \CDR@linenosep \boite@leftrule
  \advance \CDR@linenosep \boite@boxsep
}

\def\CDR@global@options{}

\newcommand\setpygmented[1]{%
  \def\CDR@global@options{/CDR.cd,#1}%
}

```

26 Counters

\CDR_int_new:nn

\CDR_int_new:n {⟨name⟩} {⟨value⟩}

Create an integer after ⟨name⟩ and set it globally to ⟨value⟩. ⟨name⟩ is a code name.

```
\cs_new:Npn \CDR_int_new:nn #1 #2 {
  \int_new:c {g/CDR/int/#1}
  \int_gset:cn {g/CDR/int/#1} { #2 }
}
```

\CDR_int_set:nn
\CDR_int_gset:nn

\CDR_int_set:n {⟨name⟩} {⟨value⟩}

Set the integer named after ⟨name⟩ to the ⟨value⟩. \CDR_int_gset:n makes a global change. ⟨name⟩ is a code name.

```
\cs_new:Npn \CDR_int_set:nn #1 #2 {
  \int_set:cn {g/CDR/int/#1} { #2 }
}
\cs_new:Npn \CDR_int_gset:nn #1 #2 {
  \int_gset:cn {g/CDR/int/#1} { #2 }
}
```

\CDR_int_add:nn
\CDR_int_gadd:nn

\CDR_int_add:n {⟨name⟩} {⟨value⟩}

Add the ⟨value⟩ to the integer named after ⟨name⟩. \CDR_int_gadd:n makes a global change. ⟨name⟩ is a code name.

```
\cs_new:Npn \CDR_int_add:nn #1 #2 {
  \int_add:cn {g/CDR/int/#1} { #2 }
}
\cs_new:Npn \CDR_int_gadd:nn #1 #2 {
  \int_gadd:cn {g/CDR/int/#1} { #2 }
}
```

\CDR_int_sub:nn
\CDR_int_gsub:nn

\CDR_int_sub:n {⟨name⟩} {⟨value⟩}

Substract the ⟨value⟩ from the integer named after ⟨name⟩. \CDR_int_gsub:n makes a global change. ⟨name⟩ is a code name.

```
\cs_new:Npn \CDR_int_sub:nn #1 #2 {
  \int_sub:cn {g/CDR/int/#1} { #2 }
}
\cs_new:Npn \CDR_int_gsub:nn #1 #2 {
  \int_gsub:cn {g/CDR/int/#1} { #2 }
}
```

\CDR_int_if_exist:nTF

\CDR_int_if_exist:nTF {⟨name⟩} {⟨true code⟩} {⟨false code⟩}

Execute ⟨*true code*⟩ when an integer named after ⟨*name*⟩ exist, ⟨*false code*⟩ otherwise.

```
\prg_new_conditional:Nnn \CDR_int_if_exist:n { T, F, TF } {
  \int_if_exist:cTF {g/CDR/int/#1} {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

\g/CDR/int/
\g/CDR/int/<name>

Generic and named line number counter. \l_CDR_code_name_t is used as ⟨*name*⟩.

```
\CDR_int_new:nn {} { 1 }
```

(*End definition for* \g/CDR/int/ *and* \g/CDR/int/<name>. *These variables are documented on page* **??**.)

\CDR_int_use:n ⋆

\CDR_int_use:n {⟨name⟩}

⟨*name*⟩ is a code name.

```
\cs_new:Npn \CDR_int_use:n #1 {
  \int_use:c {g/CDR/int/#1}
}
```

```
% ==========================================================
% final actions
% ==========================================================

\AtEndOfPackage{%
  \IfFileExists{\jobname.pygmented}{%
    \input{\jobname.pygmented}%
  }{%
    \PackageWarning{coder}{File ‘\jobname.pygmented’ not found.}%
  }%
  \immediate\openout\CDR@outfile\jobname.snippets%
}

\AtEndDocument{%
  \closeout\CDR@outfile%
}
\ExplSyntaxOff
```

```
%</sty>
```