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. `Expl` identifiers also start with `CDR`, after and eventual leading `c_`, `l_` or `g_`. l3keys module path's first component is either `CDR` starts with or `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. with coder.sty, LaTeX parses a code snippet, store it in `\l_CDR_snippet_tl`, 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 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_all_files` method is 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. Each 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 lpeg   = require("lpeg")
local P, Cg, Cp, V = lpeg.P, lpeg.Cg, lpeg.Cp, lpeg.V
local lfs    = _ENV.lfs
local tex    = _ENV.tex
local token  = _ENV.token
local rep    = string.rep
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
local dir_p, json_p
local jobname = tex.jobname
dir_p = './'..jobname..'.pygd/'
if make_directory(dir_p) == nil then
  dir_p = './'
  json_p = dir_p..jobname..'.pyg.json'
else
  json_p = dir_p..'input.pyg.json'
end
```

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

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

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

load_exec `CDR.load_exec(`⟨*code chunk*⟩`)`

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

```lua
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 `[`⟨`=...=`⟩`[` nor `]`⟨*ans*⟩`]`.

```lua
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
    j, i = eq_pattern:match(s, i)
    if j == nil then
      return rep('=', max + 1)
    end
    j = i - j
    if j > max then
      max = j
    end
  end
end
```

self:record_start `self:record_start(`⟨*tags variable name*⟩`)`

Instance method. Setup the receiver's `record_line` method to record a line stored in a LaTeX3 token variable.

```lua
local function record_start(self, tags_variable)
  local records = self['.records']
  local tags = assert(token.get_macro(tags_variable))
  tags = tags:gmatch('[^,]*')
  local list = {}
  for _,tag in ipairs(tags) do
    local t = records[tag] or {}
    records[tag] = t
    list[#list+1] = t
  end
  self.record_line = function (this, line_variable)
    local line = assert(token.get_macro(line_variable))
    for _,t in ipairs(list) do
      t[#t+1]=line
    end
  end
end
```

self:record_stop `self:record_stop()`

Instance method. Reset the `record_line` method to a noop.

```lua
local function record_stop(self)
  self.record_line = function (this, line_variable)
  end
end
```

self:record_line `self:record_line(`⟨*line variable*⟩`)`

Instance method. Record a line stored in the ⟨*line variable*⟩ or noop. Sets by `self:record_start`, unset by `self:record_stop`.

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

Instance method to parse the ⟨*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.

```lua
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)
      tex.print([[%
\directlua{self.load_exec([=]]..eqs..[[]..cmd..[[]=]]..eqs..[[])}%
]])
    else
      return
    end
  end
end
```

process_code `CDR:process_code()`

Instance method. This is called by function `\CDRCode`. First, we get the content of the ⟨*cs name*⟩ as code to be colored. Then we build a JSON string, save it in a file at `json_p` location. Next we call the coder-tool.py, parse its output and execute commands with `load_exec_output`.

```
local function process_code(self, code_name)
  if lfs.attributes(json_p,"mode") ~= nil then
    os.remove(json_p)
  end
  local t = {
    ['code']    = token.get_macro(code_name),
    ['jobname'] = jobname,
    ['options'] = self.options or {},
    ['already'] = self.already and 'true' or 'false'
  }
  local s = json.tostring(t,true)
  local fh = assert(io.open(json_p,'w'))
  fh:write(s, '\n')
  fh:close()
  local cmd = "python3 "..CDR_PY_PATH..' "'..self.escape(json_p)..'"'
  fh = assert(io.popen(cmd))
  self.already = true
  s = fh:read('a')
  self:load_exec_output(s)
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*$_n$*ame*⟩`)`

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
```

options_reset `CDR:options_reset()`
option_add `CDR:option_add(`⟨*string key*⟩`,`⟨*json value*⟩`)`

Instance method. The extra options used for formatting are collected, then forwarded to coder-tool.py utility through its JSON input, with key `options`. First we have to clear the option list with `options_reset` before any call to `option_add`.

```
local function options_reset(self)
  self.options = {}
end
local function option_add(self,k,v)
  self.options[k] = v
end
```

5 Exportation

export_file `CDR:export_file(`⟨*file name*⟩`, `⟨*tag clist*⟩`)`

Instance method. This is called by `\CDRExport` function. The ⟨*file name*⟩ is a string. ⟨*tag clist*⟩ is an ordered list of tags, The receiver's `__export_files` is a ⟨*file name*⟩`->`⟨*tags table*⟩ table.

```
local function export_file(self, name, tags)
  local t = {}
  tags:gsub(
    ’([^,]*)’,
    function(tag) t[#t+1] = tag end
  )
  self[’.export_files’][name] = t
end
```

exportNO `CDR:export(⟨file⟩, ⟨tags⟩, ⟨preamble⟩, ⟨postamble⟩)`

Instance method. This is called by the `\CDRExport` command. The receiver's `.exports` is a ⟨*file name*⟩`->`⟨*export record*⟩ table whereas the receiver's `.records` is a ⟨*tag name*⟩`->`⟨*code record*⟩ table.

```
local function export(self, file, tags, preamble, postamble)
  local exports = self['.exports']
  file = assert(token.get_macro(assert(file)))
  local t = {}
  tags = assert(token.get_macro(assert(tags)))
  t.tags = tags:gmatch('[^,]*')
  if #preamble>0 then
    t.preamble = assert(token.get_macro(preamble))
  end
  if #postamble>0 then
    t.postamble = assert(token.get_macro(postamble))
  end
  exports[file] = t
end
```

export_all_files `CDR:export_all_files()`

Instance method. This is called at the very end of the typesetting process. The receiver's `.records` is a ⟨*file name*⟩`->`⟨*code record*⟩ table.

```
local function export_all_files(self)
  local exports = self['.exports']
  local records = self['.records']
  for name, export in pairs(exports) do
    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
  end
end
```

6 Caching

We save some computation time by pygmentizing files only when necessary. The coder-tool.py is expected to create a `*.pyg.sty` file for a style and a `*.pyg.tex` file for 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`
`cache_record`
`cache_clean_unused`

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

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

```
local function cache_clean_all(self)
  local to_remove = {}
  for f in lfs.dir(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
```

6.1 Fields

The fields are properties gathered by domain. A domain is uniquely identified by its name. We can wrap domain changes in groups, pretty much like TeX groups by using lua metatables.

`self:field_group_begin` `self:field_group_begin(⟨domain name⟩)`
`self:field_group_end` `self:field_group_end(⟨domain name⟩)`

⟨*domain name*⟩ is unique identifier within the receiver's lifetime. `self:field_group_begin` and `self:field_group_begin` must be properly balanced, otherwise an exception is raised.

```
local function field_group_begin(self, domain)
  self['.fields'][domain] = setmetatable(
    {},
    self['.fields'][domain]
  )
end
local function field_group_end(self, domain)
  self['.fields'][domain] = assert(
    getmetatable(self['.fields'][domain])
  )
end
```

`self:put` `self:field_put(⟨domain name⟩,⟨key⟩, ⟨value macro name⟩)`
`self:get` `self:field_get(⟨domain name⟩,⟨key⟩)`
`self:print_value` `self:field_print(⟨domain name⟩,⟨key⟩)`

⟨*key*⟩ is a string, whereas ⟨*value macro name*⟩ is the name of a TEX macro containing the value. This allows values to be token lists instead of pure Lua objects. `self:field_put` is a setter, `self:field_get` is a getter to store and retrieve field values. `self:field_print` retrieves field values and print them to the TEX stream.

```
local function field_put(self, domain, key, value)
  value = token.get_macro(assert(value))
  self['.fields'][domain][key] = value
end
local function field_get(self, domain, key)
  return self['.fields'][domain][key]
end
local function field_print(self, domain, key)
  tex.print(self:field_get(domain, key))
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,
  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']   = {},
  ['.export_files']  = {},
  ['.records']       = {},
  ['.fields']        = {},
  ['.options']       = {},
  field_group_begin  = field_group_begin,
  field_group_end    = field_group_end,
  field_put          = field_put,
  field_get          = field_get,
  field_print        = field_print,
  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_if_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` Paths of L3keys modules.
`\c_CDR_get` Used to get or set values.
`\c_CDR_tag_get` These are root path components used throughout the pakage.

```
\str_const:Nn \c_CDR_tags { CDR@tags }
\str_const:Nn \c_CDR_tag { CDR@tags/tag }
\str_const:Nn \c_CDR_get { CDR@get }
\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`, `\c_CDR_get`, *and* `\c_CDR_tag_get`. *These variables are documented on page* ??.)

4 Implementation details

As far as possible, macro making assignments to variables are protected.

5 Utilities

\CDR_tag:n ⋆ \CDR_tag:n {⟨*tag name*⟩}
\CDR_tag:o ⋆

Build a key path.

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

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

6 Variables

6.1 Internal scratch variables

These local variables are used in a very limited scope.

\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* ??.)

6.2 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* ??.)

6.3 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* ??.)

6.4 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* ??.)

7 Tree data storage

7.1 Store

`\CDR_set:nnn` `\CDR_set:nnn {⟨dir⟩} {⟨relative key path⟩} {⟨value⟩}`

Store ⟨*value*⟩, which is retrieved with the instruction `\CDR_get:nn {⟨dir⟩} {⟨relative key path⟩}`. Only ⟨*dir*⟩ and ⟨*relative key path*⟩ containing no `@` character are supported.

```
\cs_new:Npn \CDR_set:nnn #1 #2 #3 {
  \cs_set:cpn { \c_CDR_get @ #1 @ #2 : } { \exp_not:n { #3 } }
}
```

`\CDR_set:n` `\CDR_set:n {⟨value⟩}`

This must be indirectly called by `\keys_set:nn`. Parse the current value of the l3keys variable `\l_keys_path_str` to guess a ⟨*dir*⟩ and a ⟨*relative key path*⟩ to feed `\CDR_set:nnn`. More precisely, `\l_keys_path_str` is expected to read something like `CDR/`⟨*dir*⟩`/`⟨*relative key path*⟩, an exception is raised on the contrary.

Implementation detail: the required parameter will be read by the last instruction.

```
\cs_set:Npn \CDR_set:n {
  \exp_args:NnV
  \regex_extract_once:nnNTF {
    ^CDR/([^/]*?)/(.*?\s*)$
  } \l_keys_path_str \l_CDR_seq {
    \exp_args:Nxx
    \CDR_set:nnn
      { \seq_item:Nn \l_CDR_seq 2 }
      { \seq_item:Nn \l_CDR_seq 3 }
  } {
    \PackageWarning
      { coder }
      { Unexpected~key~`\l_keys_path_str' }
    \use_none:n
  }
}
```

7.2 Get

`\CDR_get_path:nn` ⋆ `\CDR_get_path:nn {⟨dir⟩} {⟨relative key path⟩}`

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

```
\cs_set:Npn \CDR_get_path:nn #1 #2 {
  \c_CDR_get @ #1 @ #2 :
}
```

`\CDR_if_exist_here:nnTF` ⋆ `\CDR_if_exist_here:nnTF {⟨dir⟩} ⟨relative key path⟩ {⟨true code⟩} {⟨false code⟩}`

If the ⟨*relative key path*⟩ is known within ⟨*dir*⟩, the ⟨*true code*⟩ is executed, otherwise, the ⟨*false code*⟩ is executed. Do not use inheritance.

```
\prg_new_conditional:Nnn \CDR_if_exist_here:nn { T, F, TF } {
  \cs_if_exist:cTF { \CDR_get_path:nn { #1 } { #2 } } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

\CDR_if_exist:nnTF ⋆ \CDR_if_exist:nnTF {⟨dir⟩} ⟨relative key path⟩ {⟨true code⟩} {⟨false code⟩}

If the ⟨relative key path⟩ is known within ⟨dir⟩ or one of its parents, the ⟨true code⟩ is executed, otherwise, the ⟨false code⟩ is executed. Takes care of inheritance.

```
\prg_new_conditional:Nnn \CDR_if_exist:nn { T, F, TF } {
  \cs_if_exist:cTF { \CDR_get_path:nn { #1 } { #2 } } {
    \prg_return_true:
  } {
    \seq_if_exist:cTF { \CDR_parent_seq:n { #1 } } {
      \seq_map_tokens:cn
        { \CDR_parent_seq:n { #1 } }
        { \CDR_if_exist_alt_f:nn { #2 } }
    } {
      \prg_return_false:
    }
  }
}
\cs_set:Npn \CDR_if_exist_f:nn #1 #2 {
  \quark_if_no_value:nTF { #2 } {
    \seq_map_break:n {
      \prg_return_false:
    }
  } {
    \CDR_if_exist:nnT { #2 } { #1 } {
      \seq_map_break:n {
        \prg_return_true:
      }
    }
  }
}
```

\CDR_get_here:nn ⋆ \CDR_get_here:nn {⟨dir⟩} {⟨relative key path⟩}

Leave in the input stream whatever was previously stored with an instruction like \CDR_set:nnn {⟨dir⟩} {⟨relative key path⟩} {⟨value⟩}. Do not use inheritance.

```
\cs_set:Npn \CDR_get_here:nn #1 #2 {
  \CDR_if_exist_here:nnT { #1 } { #2 } {
    \use:c { \CDR_get_path:nn { #1 } { #2 } }
  }
}
```

\CDR@tag@getnn ⋆ \CDR_get:nn {⟨dir⟩} {⟨relative key path⟩}

Leave in the input stream whatever was previously stored with an instruction like \CDR_set:nnn {⟨dir⟩} {⟨relative key path⟩} {⟨value⟩}. Uses inheritance.

```
\cs_set:Npn \CDR_get:nn #1 #2 {
  \CDR_if_exist_here:nnTF { #1 } { #2 } {
    \use:c { \CDR_get_path:nn { #1 } { #2 } }
  } {
    \seq_if_exist:cT { \CDR_parent_seq:n { #1 } } {
      \seq_map_tokens:cn
```

```
        { \CDR_parent_seq:n { #1 } }
        { \CDR_get_f:nn { #2 } }
    }
  }
}
\cs_set:Npn \CDR_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_get_path:nn { #2 } { #1 } }
      }
    }
  }
}
```

\CDR_get_here:nnN*TF* \CDR_get_here:nnNTF {⟨*dir*⟩} {⟨*relative key path*⟩} {⟨*tl var*⟩} {⟨*true code*⟩} {⟨*false code*⟩}

If ⟨*relative key path*⟩ is known within ⟨*dir*⟩, fill ⟨*tl var*⟩ with the corresponding value and execute ⟨*true code*⟩. Execute ⟨*false code*⟩ otherwise. No inheritance.

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

\CDR_get:nnN*TF* \CDR_get:nnNTF {⟨*dir*⟩} {⟨*relative key path*⟩} {⟨*tl var*⟩} {⟨*true code*⟩} {⟨*false code*⟩}

If ⟨*relative key path*⟩ is known within ⟨*dir*⟩, fill ⟨*tl var*⟩ with the corresponding value and execute ⟨*true code*⟩. Execute ⟨*false code*⟩ otherwise. Takes care of inheritance.

```
\prg_new_conditional:Nnn \CDR_get:nnN { T, F, TF } {
  \CDR_if_exist_here:nnTF { #1 } { #2 } {
    \tl_set:Nx #3 { \CDR_get_here:nn { #1 } { #2 } }
    \prg_return_true:
  } {
    \seq_if_exist:cTF { \CDR_parent_seq:n { #1 } } {
      \seq_map_inline:cn { \CDR_parent_seq:n { #1 } } {
        \quark_if_no_value:nF { ##1 } {
          \CDR_if_exist_here:nnT { ##1 } { #2 } {
            \seq_map_break:n {
              \tl_set:Nx #3 { \CDR_get_here:nn { ##1 } { #2 } }
              \use:c { \CDR_get_path:nn { ##1 } { #2 } }
              \prg_return_true:
            }
          }
        }
```

```
      }
    } {
      \prg_return_false:
    }
  }
}
```

7.3 Inherit

\CDR_parent_seq:n ⋆ \CDR_parent_seq:n {⟨*dir*⟩}

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

```
\cs_new:Npn \CDR_parent_seq:n #1 {
  g_CDR:parent @ #1 _seq
}
```

\CDR_inherit:nn \CDR_inherit:nn {⟨*dir*⟩} {⟨*parent comma list*⟩}

Set the parents of ⟨*dir*⟩ to the given list.

```
\cs_new:Npn \CDR_inherit:nn #1 #2 {
  \tl_set:Nx \l_CDR_tl { \CDR_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 }
}
```

8 Tag properties

The tag properties concern the code chunks level. 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 one contain `CDR_tag`.

8.1 Set

\g_CDR_tag_path_seq Global variable to store relative key path.

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

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

\CDR_tag_set:n \CDR_tag_set:n {⟨*value*⟩}

The value is provided but not the ⟨*dir*⟩ nor the ⟨*relative key path*⟩: which a tag name, 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. Record the relative key path (the part after the tag name) of the current full key path in `g_CDR_tag_path_seq`. Useful for automatic management to know what has been defined. This is meant to be call from `\keys_define:nn` argument. Implementation detail: the last argument is parsed by the last command.

```
\cs_set:Npn \CDR_tag_set:n {
  \exp_args:NnV
  \regex_extract_once:nnNTF {
    ^CDR@tag/([^/]*)/(.*)$
  } \l_keys_path_str \l_CDR_seq {
    \seq_gput_left:Nx \g_CDR_tag_path_seq { \seq_item:Nn \l_CDR_seq 3 }
    \CDR_tag_set:nnn
      { \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_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_set:Npn \CDR_tag_get_path:nn #1 #2 {
  \c_CDR_tag_get @ #1 @ #2 :
}
```

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

Store ⟨*value*⟩, which is 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.

```
\cs_new:Npn \CDR_set:nnn #1 #2 #3 {
    \seq_gput_left:Nx \g_CDR_tag_path_seq { #2 }
  \cs_set:cpn { \CDR_tag_get_path:nn { #1 } { #2 } } { \exp_not:n { #3 } }
}
```

\CDR_tag_set:n `\CDR_tag_set:n {⟨value⟩}`

The value is provided but not the ⟨*dir*⟩ nor the ⟨*relative key path*⟩: which a tag name, 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. Record the relative key path (the part after the tag name) of the current full key path in `g_CDR_tag_path_seq`. Useful for automatic management to know what has been defined. This is meant to be call from `\keys_define:nn` argument. Implementation detail: the last argument is parsed by the last command.

```
\cs_set:Npn \CDR_tag_set:n {
  \exp_args:NnV
  \regex_extract_once:nnNTF {
    ^CDR@tag/([^/]*)/(.*)$
  } \l_keys_path_str \l_CDR_seq {
    \CDR_tag_set:nnn
```

```
      { \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:nn

\CDR_tag_set:nn {⟨*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_set:Npn \CDR_tag_set:nn #1 {
  \exp_args:NnV
  \regex_extract_once:nnNTF {
    ^CDR@tag/([^/]*)/.*$
  } \l_keys_path_str \l_CDR_seq {
    \CDR_tag_set:nnn
      { \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_set: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 {
      \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_set: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_set: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 }
  }
}
```

8.2 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:nnTF` ⋆

`\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_set: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_set: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_set: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:
  }
}
```

8.3 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 }
}
```

8.4 Handling unknown tags

\keys_define:on \keys_define:(ox|oo) \keys_set:xn \keys_set_known:onoN \keys_set_known:oooN	Various variants

```
\clist_map_inline:nn { o, ox, oo } {
  \cs_generate_variant:Nn \keys_define:nn { #1 }
}
\cs_generate_variant:Nn \keys_set:nn { x }
\clist_map_inline:nn { ono, ooo } {
  \cs_generate_variant:Nn \keys_set_known:nnnN { #1 }
}
```

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_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_keys_tag_set:nn #1 {
  \keys_set:xn { \CDR_tag:n { #1 } }
}
\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_tag_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
  }
}
```

9 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()}
}
```

10 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_if_has_pygment:TF` ⋆ `\CDR_if_has_pygment:TF {⟨true code⟩} {⟨false code⟩}`

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

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

Utilities

`\CDR_clist_map_inline:Nnn` `\CDR_clist_map_inline:Nnn ⟨clist var⟩ {⟨non empty code⟩} {⟨empty code⟩}`

Call `\clist_map_inline:Nnn` ⟨*clist var*⟩ `then` ⟨*non empty code*⟩ when the list is not empty, execute ⟨*empty code*⟩ otherwise.

```
\cs_new:Npn \CDR_clist_map_inline:Nnn #1 #2 #3 {
  \clist_if_empty:NTF #1 { #3 } {
    \clist_map_inline:Nn #1 { #2 }
  }
}
```

`\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: {}
```

11 l3keys modules

11.1 \c_CDR_tag l3keys module

Each action is meant to store the values in the tree storage.

11.2 \c_CDR_tag/default

```
\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:n { #1 },
  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:n { #1 },
  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:n { #1 },
```

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

```
  parskip .code:n = \CDR_tag_set:n { #1 },
  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:n { #1 },
  engine .value_required:n = true,
```

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

```
  default~engine~options .code:n = \CDR_tag_set:n { #1 },
  default~engine~options .value_required:n = true,
```

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

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

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

11.3 \c_CDR_tag/__pygment

These are pygment's `LatexFormatter` options.

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`.

11.4 \c_CDR_tag/default.block

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

Known keys include:

- **show tags[=true|false]** to enable/disable the display of the code chunks tags. Initially `true`.

```
  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:n { #1 },
  tags~format .required_value:n = true,
```

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

```
  blockskip .code:n = \CDR_tag_set:n { #1 },
  blockskip .required_value:n = true,
```

- **__initialize_block** the separation with the surrounding text. Initially `\topsep`.

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

```
}
```

12 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.

12.1 `\c_CDR_tag/__fancyvrb.block`

Block specific options.

```
\keys_define:on { \c_CDR_tag / __fancyvrb.block } {
```

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

```
  commentchar .code:n = \CDR_tag_set:n { #1 },
  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. Ignored in `code` mode.

```
  label .code:n = \CDR_tag_set:n { #1 },
  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. Ignored in `code` mode.

```
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. Ignored in `code` mode.

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

numbersep=⟨*dimension*⟩ gap between numbers and verbatim lines. Initially 12pt. Ignored in `code` mode.

```
numbersep .code:n = \CDR_tag_set:n { #1 },
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. Ignored in `code` mode.

```
firstnumber .code:n = {
  \regex_match:nnTF { ^(+|-)?\d+$ } { #1 } {
    \CDR_tag_set:n { #1 }
  } {
    \str_case:nnF { #1 } {
      { auto } { \CDR_tag_set:n { #1 } }
      { last } { \CDR_tag_set:n { #1 } }
    } {
      \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. Ignored in `code` mode.

```
stepnumber .code:n = \CDR_tag_set:n { #1 },
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. Ignored in `code` mode.

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

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

```
firstline .code:n = \CDR_tag_set:n { #1 },
firstline .value_required:n = true,
```

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

```
lastline .code:n = \CDR_tag_set:n { #1 },
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:n { #1 },
baselinestretch .value_required:n = true,
```

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

```
xleftmargin .code: = \CDR_tag_set:n { #1 },
xleftmargin .value_required:n = true,
```

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

```
xrightmargin .code: = \CDR_tag_set:n { #1 },
xrightmargin .value_required:n = true,
```

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

```
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. Ignored in `code` mode.

```
hfuzz .code: = \CDR_tag_set:n { #1 },
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`. Ignored in `code` mode.

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

✅ **__initialize_fancyvrb.block** Initialization.

```
  __initialize_fancyvrb .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_fancyvrb.block .value_required:n = true,

}
```

12.2 \c_CDR_tag / __fancyvrb

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

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

```
  formatcom .code:n = \CDR_tag_set:n { #1 },
  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:n { #1 },
  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:n { #1 },
  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:n { #1 },
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:n { #1 },
tabsize .value_required:n = true,
```

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:n { #1 },
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:n { #1 },
codes .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:n { #1 },
defineactive .value_required:n = true,
```

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

```
reflabel .code: = \CDR_tag_set:n { #1 },
reflabel .value_required:n = true,
```

✅ **__initialize_fancyvrb** Initialization.

```
    formatcom = ,
    fontfamily = tt,
    fontsize = auto,
    fontshape = auto,
    showspaces = false,
    showtabs = false,
    obeytabs = false,
    tabsize = 2,
    commandchars = ,
    codes = ,
    defineactive = ,
    reflabel = ,
  },
  __initialize_fancyvrb .value_required:n = true,

}
```

13 \CDRSet

\CDRSet `\CDRSet {⟨key[=value] list⟩}`

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.

13.1 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.

13.2 `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
```

```
}
```

13.3 Implementation

\CDR_check_unknown:V

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

Check for unknown keys.

```
\cs_new:Npn \CDR_check_unknown:V #1 {
  \tl_if_empty:NF #1 {
    \cs_set:Npn \CDRSet_unknown:n ##1 {
      \PackageWarning
        { coder }
        { Unknow~key~`##1' }
    }
    \cs_set:Npn \CDRSet_unknown:nn ##1 ##2 {
      \CDRSet_unknown:n { ##1 }
    }
    \exp_args:Nnno
    \keys_parse:nnn {
      \CDRSet_unknown:n
    } {
      \CDRSet_unknown:nn
    } #1
  }
}
```

```
\NewDocumentCommand \CDRSet { m } {
  \keys_set:nn { CDR:set } { __initialize_set }
  \keys_set_known:nnnN { CDR:set } { #1 } { CDR:set } \l_CDR_tl
  \keys_set_known:oooN
    \c_CDR_tag \l_CDR_tl \c_CDR_tag \l_CDR_tl
  \exp_args:NV
  \CDR_tag_provide_from_keyval:n \l_CDR_tl
  \keys_set_known:oooN
    { \c_CDR_tag / default.block } { \l_CDR_tl }
    { \c_CDR_tag / default.block } \l_CDR_tl
  \keys_set_known:oooN
    { \c_CDR_tag / default.code } { \l_CDR_tl }
    { \c_CDR_tag / default.code } \l_CDR_tl
  \keys_set_known:oooN
    { \c_CDR_tag / default } { \l_CDR_tl }
    { \c_CDR_tag / default } \l_CDR_tl
  \CDR_check_unknown:V \l_CDR_tl
}
```

14 \CDRExport

\CDRExport

\CDRExport {⟨*key[=value] controls*⟩}

The ⟨*key*⟩[=⟨*value*⟩] controls are defined by CDR:export l3keys module.

14.1 Storage

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

```
\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.

```
\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.

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

(*End definition for* `\l_CDR_tags_clist`. *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.

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

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

14.2 `CDR:export` l3keys module

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

```
\keys_define:nn { CDR:export } {
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

✅ **__initialize_prop** properly initialize the local property storage.

```
__initialize_prop .code:n = \prop_clear:N \l_CDR_export_prop,
```

14.3 Implementation

```
\DeclareDocumentCommand \CDRExport { m } {
  \keys_set:nn { CDR:export } { __initialize_export }
  \keys_set_known:nnnN
    { CDR:export } { #1 } { CDR:export } \l_CDR_tl
  \tl_if_empty:NTF \l_CDR_file_tl {
    \PackageWarning
      { coder }
      { Missing~key~`file' }
  } {
    \CDR_check_unknown:V \l_CDR_tl
    \prop_put:Nno \l_CDR_prop { file } \l_CDR_file_tl
    \prop_gput:Noo \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_clist {
      \prop_get:NnNT \l_CDR_prop { lang } \l_CDR_tl {
        \clist_map_inline:Nn \l_CDR_clist {
          \exp_args:Nnno
          \CDR_tag_set:nnn { ##1 } { lang } \l_CDR_tl
        }
      }
    }
  }
}
```

`\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:x { T, F, TF } {
  \exp_args:Nnx
  \regex_match:nnTF { ^[tTyY] } { #1 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

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() }
  }
}
```

15 Creating display engines

\CDRNewCodeEngine
\CDRRenewCodeEngine

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

⟨*engine name*⟩ is a non void string, once expanded. The code methods create a command with a unique argument which is the colored code.

```
\cs_new:Npn \CDRNewCodeEngine #1 #2 {
  \exp_args:Nx
  \str_if_empty:nTF { #1 } {
    \PackageWarning
      { coder }
      { The~engine~cannot~be~void. }
  } {
    \cs_new:cpn { \CDR_code_engine:n {#1} } ##1 {
      #2
    }
    \ignorespaces
  }
}
```

```
\cs_new:Npn \CDRRenewCodeEngine #1 #2 {
  \exp_args:Nx
  \str_if_empty:nTF { #1 } {
    \PackageWarning
      { coder }
      { The~engine~cannot~be~void. }
  } {
    \cs_if_exist:cTF { \CDR_code_engine:n { #1 } } {
      \cs_set:cpn { \CDR_code_engine:n { #1 } ##1 {
        #2
      }
    } {
      \PackageWarning
        { coder }
        { No~code~method~#1.}
    }
    \ignorespaces
  }
}
```

\CDRNewBlockEngine
\CDRRenewBlockEngine

\CDRNewBlockEngine{⟨*engine name*⟩}{⟨*begin instructions*⟩}{⟨*end instructions*⟩}
\CDRRenewBlockEngine{⟨*engine name*⟩}{⟨*begin instructions*⟩}{⟨*end instructions*⟩}

⟨*engine name*⟩ is a non void string. The block methods create an environment. Various options are available with the `\CDRGetOption` function.

```
\cs_new:Npn \CDRNewBlockEngine #1 #2 {
  \NewDocumentEnvironment { \CDR_block_engine:n { #1 } } {} {
    \cs_set_eq:NN \CDRGetOption \CDR_tag_get:n
    #2
  }
}
```

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

\CDR_has_code_engine:n*TF* ⋆ \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:n*TF* ⋆ \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:
  }
}
```

15.1 Code mode default engine

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

15.2 Block mode default engine

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

16 \CDRCode function

16.1 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* ??.)

16.2 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,

}
```

16.3 Implementation

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

```
\NewDocumentCommand \CDRCode { mm } {
  \group_begin:
    \keys_define:ox { \c_CDR_tag } {
      __local .inherit:n = {
        CDR/code,
        \c_CDR_tag/default.code,
        \c_CDR_tag/default,
      },
    }
    \str_set:No \l_CDR_str { \CDR_tag:n { __local } }
    \keys_set_known:onoN
      \l_CDR_str { #1 } \l_CDR_str \l_CDR_tl
    \CDR_check_unknown:V \l_CDR_tl
    \exp_aegs:Nono
    \keys_set_known:onoN
      \l_CDR_str { #1 } \l_CDR_str \l_CDtl
    \CDR_check_unknown:V \l_CDtl
    \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,
    }
    \CDR_to_lua:
    \exp_args:Nx \label { \CDR_tag_get:n {reflabel} }
```

```
    \SaveVerb
      [aftersave={
       \UndefineShortVerb { #2 }
       \lua_now:n {CDR:process_code('FV@SV@CDRCode')}
       \group_end:
      }]
      {CDRCode}
}
```

\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() }
  \prop_clear:N \l_CDR_prop
  \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') }
    }
  }
}
```

17 CDRBlock environment

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

17.1 Storage

\l_CDR_tags_tl

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

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

\l_CDR_block_prop

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

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

17.2 CDR/block l3keys module

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

tag related keys See \c_CDR_tag l3keys module,

```
\keys_define:nn { CDR } {
  block .inherit:n = {CDR/default.block, CDR/default},
}
\keys_define:nn { CDR / block } {
```

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

```
  tags .code:n = {
    \clist_set:Nn \l_CDR_clist { #1 }
    \clist_remove_duplicates:N \l_CDR_clist
    \clist_remove_all:N \l_CDR_clist {}
    \exp_args:Nx \CDR_tag_set:n { \clist_use:Nn \l_CDR_clist , }
  },
  tags .required:n = true,
```

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,
```

__initialize_block initialize

```
  __initialize_block .meta:n = {
    __initialize_default,
    __initialize_default.block,
    tags = ,
    ignore = false,
    test= false,
  },

}
```

17.3 Context

Inside the `CDRBlock` environments, some local variables are available:

`\l_CDR_tags_clist`

17.4 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:ox { \c_CDR_tag } {
    __local .inherit:n = {
      CDR/block,
      \CDR_tag:n { default.block },
      \CDR_tag:n { default },
    }
  }
  \keys_set:xn { \CDR_tag: { __local } } { #1 }
  \keys_define:ox { \c_CDR_tag_get } {
    __local .inherit:n = {
      \clist_if_empty:NF \l_CDR_tags_clist {
        \c_CDR_tag_get/\clist_use:Nn \l_CDR_tags_clist {,\c_CDR_tag_get/}
      },
      \c_CDR_tag_get/default.block,
      \c_CDR_tag_get/default,
    }
  }
  \CDR_feed_local_prop:
  \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 }
    }
  }
  \FV@VerbatimBegin
  \FV@Scan
}
\def\FVE@CDRBlock{
  \FV@VerbatimEnd
  \group_end:
  \@esphack
}
\DefineVerbatimEnvironment{CDRBlock}{CDRBlock}{}


\NewDocumentEnvironment{pygmented}{+O{}m}{%
```

```
  \lua_now:n {CDR:record_start()}
  \CDR@process@options{#1}%
  \immediate\write\CDR@outfile{<@@CDR@display@\the\CDR@counter}%
  \immediate\write\CDR@outfile{
    \exp_args:NV\detokenize\CDR@global@options,\detokenize{#1}
  }%
  \VerbatimEnvironment
  \begin{VerbatimOutAppend}{\CDR@outfile}%
}{%
  \end{VerbatimOutAppend}%
  \immediate\write\CDR@outfile{>@@CDR@display@\the\CDR@counter}%
  \csname CDR@snippet@\the\CDR@counter\endcsname
  \global\advance\CDR@counter by 1\relax
}

```

18 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}{}

```

19 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_set: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
}
```

20 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* **??**.)

21 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')} }
}
```

22 Section separators

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

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

23 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:
```

24 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_eq:NN #3 #3
}
\ExplSyntaxOff

```

25 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 } {
  \directlua{CDR:record_start()}
  \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 } 1
    } {
      \CDR_int_gset:nn { } 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 { ##1 } { \g_CDR_int }
    } {
      \CDR_int_gadd:nn { } { \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


```

26 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.

26.1 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}%
}
```

26.2 Block mode default engine

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

26.3 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 hould 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:

27 Something else

some settings used by fancyvrb: * for line numbering: numbers, numbersep, firstnumber, stepnumber, numberblanklines * for selection of lines to print: firstline, lastline,

```
\pgfkeys{%
  /CDR.cd,
  %
  %
  %
  label/.code          = \CDR_set:nn {label} { #1 },
  caption/.code        = \CDR_set:nn {caption} { #1 },
  %
  %
  linenos/.code        = \CDR_set:nn {linenos} { #1 },% boolean
  linenostart/.code    = \CDR_set:nn {linenostart} { #1 },
  linenostep/.code     = \CDR_set:nn {linenostep} { #1 },
  linenosep/.code      = \CDR_set:nn {linenosep} { #1 },
  %
  colback/.code        = \CDR_set:nn {colback} { #1 },
  font/.code           = \CDR_set:nn {font} { #1 },
```

```
  %
  linenos/.default     = true,
}

\pgfqkeys{/CDR}{
  block~method = mdframed,
  run~method = efbox,
  sty           = default,
  linenos       = false,
  linenosep     = 2pt,
  font          = \ttfamily,
  tabsize       = 0,
}

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

\newwrite\CDR@outfile

\newcount\CDR@counter

  WHERE ? \fvset{gobble=0,tabsize=0}%


\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)}
\prg_new_conditional:Nnn \CDR_yorn:n { T, F, TF } {
  \group_begin:
  \prop_get:cnNT {g/CDR/Code/} { #1 } \l_tmpa_tl {
    \exp_args:NnV
    \regex_match:nnT {^[tTyY]} \l_tmpa_tl {
      \group_end:
      \prg_return_true:
    }
  }
  \group_end:
  \prg_return_false:
}
\newenvironment{CDR@snippet@framed}{%
  \group_begin:
  \CDR@leftmargin\z@
  \CDR_yorn:nT {linenos} {
    \expandafter \CDRWidest\CDR@alllinenos{\FormatLineNumber}{\CDR@leftmargin}%
    \exp_args:NNx
    \advance\CDR@leftmargin { \CDR:n {linenosep} }
  }
  %
  \tl_clear:N \l_CDR_tl
  \CDR:nNTF {label} \l_tmpa_tl {
    \tl_set:N \l_CDR_tl {%
      \captionof{pygcode}{\label{\CDR:n {label}} \CDR:n {caption}}%
      % \nopagebreak
      \vskip -0.7\baselineskip
    }%
  } {
    \CDR:nNT {caption} \l_tmpa_tl {
      \tl_set:N \l_CDR_tl {%
        \captionof {pygcode} {\l_tmpa_tl}%
        % \nopagebreak
        \vskip -0.7\baselineskip
      }%
    }
  }
  \l_CDR_tl
  %
  \exp_args:Nx \tl_if_empty:nF { \CDR:n {block_engine} } {
    \exp_args:Nx
    \CDR@process@more@options { \CDR:n {block_engine} }%
    \exp_last_unbraced:NxNo
    \begin { \CDR:n {block_engine} } [ \CDRRemainingOptions ]
  }
  \csname PYstyle\CDR@opt@style\endcsname
  \CDR@opt@font
  \noindent
} {
```

```
  \exp_args:Nx \tl_if_empty:nF { \CDR:n {block_engine} } {
    \exp_args:Nx
    \end { \CDR:n {block_engine} }
  }
  \group_end:
}

\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}%
}

```

28 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:n*TF*

\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}
}
```

29 Global properties

This package is using a key–value design to store and retrieve properties with the aid of getters and setters. We only use 3 different types of variables: `tl`s, `bool`s and `clist`s. Nevertheless, they are all stored as `tl`s to allow coder-util.lua access them directly through method `token.get_macro`. Some normalization takes place for `bool` and `clist` data types.

coder-util.lua can read macros defined on the LaTeX side, except what concerns category codes. Moreover, it cannot expand them. This is why before giving coder-util.lua a chance to read a macro, it must be exhaustively expanded.

The internals of properties storage are private and should not be relied upon.

29.1 ⟨*domain*⟩

`\g_CDR_domain_tl` It defaults to `var` but can be overriden locally within TeX groups. It will also take the value `file`.

```
\tl_new:N \g_CDR_domain_tl
\tl_set:Nn \g_CDR_domain_tl { var }
```

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

29.2 ⟨*name*⟩

`\g_CDR_name_tl` This is the value of ⟨*name*⟩ in next functions, when not explicitely provided. It defaults to `default` but can be overriden locally within TeX groups.

```
\tl_new:N \g_CDR_name_tl
\tl_set:Nn \g_CDR_name_tl { default }
```

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

29.3 Modifying properties

\CDR_set:nnn
\CDR_set:nnV
\CDR_gput:nnn
\CDR_gput:nnV
\CDR_set:nn
\CDR_set:nV
\CDR_gput:nn
\CDR_gput:nV

\CDR_put:nnn {⟨*name*⟩} {⟨*key*⟩} {⟨*value*⟩}
\CDR_put:nn {⟨*key*⟩} {⟨*value*⟩}

The value is stored in a variable uniquely named after ⟨*name*⟩ and ⟨*key*⟩.

```
\cs_new:Npn \CDR_set:nnnn #1 #2 #3 {
  \tl_set:cn {CDR.#1/#2/#3}
}
\cs_new:Npn \CDR_gput:nnn #1 #2 {
  \tl_gset:cn {CDR.#1/#2/#3}
}
\cs_new:Npn \CDR_set:nnn #1 #2 {
  \tl_set:cn {CDR.\g_CDR_domain_tl/#1/#2}
}
\cs_new:Npn \CDR_gput:nnn #1 #2 {
  \tl_gset:cn {CDR.\g_CDR_domain_tl/#1/#2}
}
\cs_new:Npn \CDR_set:nn {
  \exp_args:NVV \CDR_set:nnnn \g_CDR_domain_tl \g_CDR_name_tl
}
\cs_new:Npn \CDR_gput:nn {
  \exp_args:NVV \CDR_gput:nnnn \g_CDR_domain_tl \g_CDR_name_tl
}
```

```
\cs_generate_variant:Nn \CDR_set:nnnn { nnnV }
\cs_generate_variant:Nn \CDR_gput:nnnn { nnnV }
\cs_generate_variant:Nn \CDR_set:nnn { nnV }
\cs_generate_variant:Nn \CDR_gput:nnn { nnV }
\cs_generate_variant:Nn \CDR_set:nn { nV }
\cs_generate_variant:Nn \CDR_gput:nn { nV }
```

\CDR_put_bool:nnnn
\CDR_put_not_bool:nnnn
\CDR_gput_bool:nnnn
\CDR_gput_not_bool:nnnn
\CDR_put_bool:nnn
\CDR_put_not_bool:nnn
\CDR_gput_bool:nnn
\CDR_gput_not_bool:nnn
\CDR_put_bool:nn
\CDR_put_not_bool:nn
\CDR_gput_bool:nn
\CDR_gput_not_bool:nn

\CDR_put_bool:nnnn {⟨*domain*⟩} {⟨*name*⟩} {⟨*key*⟩} {⟨*bool literal*⟩}
\CDR_put_not_bool:nnnn {⟨*domain*⟩} {⟨*name*⟩} {⟨*key*⟩} {⟨*bool literal*⟩}
\CDR_put_bool:nnn {⟨*name*⟩} {⟨*key*⟩} {⟨*bool literal*⟩}
\CDR_put_not_bool:nnn {⟨*name*⟩} {⟨*key*⟩} {⟨*bool literal*⟩}
\CDR_put_bool:nn {⟨*key*⟩} {⟨*bool literal*⟩}
\CDR_put_not_bool:nn {⟨*key*⟩} {⟨*bool literal*⟩}

The value is stored in a variable uniquely named after ⟨*name*⟩ and ⟨*key*⟩. `bool`s are normalized before storage, only `true` and `false` literals are used afterwards. In the `_not_` variants, the inverse of the value is stored instead.

```
\cs_new:Npn \CDR_put_bool:nnn #1 #2 #3 #4 {
  \group_begin:
  \tl_set:Nn \l_tmpa_tl { \CDR_set:nnnn { #1 } { #2 } { #3 } }
  \tl_put_right:Nx \l_tmpa_tl {
    \regex_match:nnTF { ^\s*[tTyY] } { #4 } { true } { false }
  }
```

```
  \exp_last_unbraced:NV \group_end: \l_tmpa_tl
}
\cs_new:Npn \CDR_put_not_bool:nnnn #1 #2 #3 #4 {
  \group_begin:
  \tl_set:Nn \l_tmpa_tl { \CDR_set:nnnn { #1 } { #2 } { #3 } }
  \tl_put_right:Nx \l_tmpa_tl {
    \regex_match:nnTF { ^\s*[tTyY] } { #4 } { false } { true }
  }
  \exp_last_unbraced:NV \group_end: \l_tmpa_tl
}
\cs_new:Npn \CDR_put_bool:nnn #1 #2 #3 {
  \exp_args:Nnx
  \CDR_set:nnn { #1 } { #2 } {
    \regex_match:nnTF { ^\s*[tTyY] } { #3 } { true } { false }
  }
}
\cs_new:Npn \CDR_put_not_bool:nnn #1 #2 #3 {
  \exp_args:Nnx
  \CDR_set:nnn { #1 } { #2 } {
    \regex_match:nnTF { ^\s*[tTyY] } { #3 } { false } { true }
  }
}
\cs_new:Npn \CDR_put_bool:nn #1 #2 {
  \exp_args:Nnx
  \CDR_set:nn { #1 } {
    \regex_match:nnTF { ^\s*[tTyY] } { #2 } { true } { false }
  }
}
\cs_new:Npn \CDR_put_not_bool:nn {
  \exp_args:Nnx
  \CDR_set:nn { #1 } {
    \regex_match:nnTF { ^\s*[tTyY] } { #2 } { false } { true }
  }
}
\cs_new:Npn \CDR_put_bool:nnn #1 #2 #3 #4 {
  \group_begin:
  \tl_set:Nn \l_tmpa_tl { \CDR_gput:nnnn { #1 } { #2 } { #3 } }
  \tl_put_right:Nx \l_tmpa_tl {
    \regex_match:nnTF { ^\s*[tTyY] } { #4 } { true } { false }
  }
  \exp_last_unbraced:NV \group_end: \l_tmpa_tl
}
\cs_new:Npn \CDR_put_not_bool:nnnn #1 #2 #3 #4 {
  \group_begin:
  \tl_set:Nn \l_tmpa_tl { \CDR_gput:nnnn { #1 } { #2 } { #3 } }
  \tl_put_right:Nx \l_tmpa_tl {
    \regex_match:nnTF { ^\s*[tTyY] } { #4 } { false } { true }
  }
  \exp_last_unbraced:NV \group_end: \l_tmpa_tl
}
\cs_new:Npn \CDR_gput_bool:nnn #1 #2 #3 {
  \exp_args:Nnnx
  \CDR_gput:nnn { #1 } { #2 } {
    \regex_match:nnTF { ^\s*[tTyY] } { #3 } { true } { false }
```

```
  }
}
\cs_new:Npn \CDR_gput_not_bool:nnn #1 #2 #3 {
  \exp_args:Nnnx
  \CDR_gput:nn { #1 } { #2 } {
    \regex_match:nnTF { ^\s*[tTyY] } { #3 } { false } { true }
  }
}
\cs_new:Npn \CDR_gput_bool:nn #1 #2 {
  \exp_args:Nnx
  \CDR_gput:nn { #1 } {
    \regex_match:nnTF { ^\s*[tTyY] } { #2 } { true } { false }
  }
}
\cs_new:Npn \CDR_gput_not_bool:nn {
  \exp_args:Nnx
  \CDR_gput:nn { #1 } {
    \regex_match:nnTF { ^\s*[tTyY] } { #2 } { false } { true }
  }
}
```

\CDR_put_clist:nnnn
\CDR_gput_clist:nnnn
\CDR_put_clist:nnn
\CDR_gput_clist:nnn
\CDR_put_clist:nn
\CDR_gput_clist:nn

\CDR_put_clist:nnn {⟨*name*⟩} {⟨*key*⟩} {⟨*clist literal*⟩}
\CDR_put_clist:nn {⟨*key*⟩} {⟨*clist literal*⟩}

The value is stored in a variable uniquely named after ⟨*name*⟩ and ⟨*key*⟩. clists are normalized before storage, spaces around values are discarded. Not strictly necessary yet.

```
\cs_new:Npn \CDR_put_clist:nnnn #1 #2 #3 #4 {
  \group_begin:
  \tl_set:Nn \l_tmpa_tl { \CDR_set:nnnn { #1 } { #2 } { #3 } }
  \clist_set:Nn \l_tmpa_clist { #4 }
  \tl_put_right:Nx \l_tmpa_tl { \clist_use:nn { #4 } , }
  \exp_args:NNnx
  \group_end:
  \CDR_set:nn { #2 } { \clist_use:Nn \l_tmpa_clist , }
}
\cs_new:Npn \CDR_gput_clist:nnnn #1 #2 #3 #4 {
  \group_begin:
  \tl_set:Nx \g_CDR_name_tl { #1 }
  \clist_set:Nn \l_tmpa_clist { #3 }
  \exp_args:NNnx
  \group_end:
  \CDR_gput:nn { #2 } { \clist_use:Nn \l_tmpa_clist , }
}
\cs_new:Npn \CDR_put_clist:nnn #1 #2 #3 {
  \group_begin:
  \tl_set:Nx \g_CDR_name_tl { #1 }
  \clist_set:Nn \l_tmpa_clist { #3 }
  \exp_args:NNnx
  \group_end:
  \CDR_set:nn { #2 } { \clist_use:Nn \l_tmpa_clist , }
}
\cs_new:Npn \CDR_gput_clist:nnn #1 #2 #3 {
  \group_begin:
  \tl_set:Nx \g_CDR_name_tl { #1 }
  \clist_set:Nn \l_tmpa_clist { #3 }
  \exp_args:NNnx
  \group_end:
  \CDR_gput:nn { #2 } { \clist_use:Nn \l_tmpa_clist , }
}
\cs_new:Npn \CDR_put_clist:nnn #1 #2  {
  \group_begin:
  \clist_set:Nn \l_tmpa_clist { #2 }
  \exp_args:NNnx
  \group_end:
  \CDR_set:nn { #1 } { \clist_use:Nn \l_tmpa_clist , }
}
\cs_new:Npn \CDR_gput_clist:nnn #1 #2  {
  \group_begin:
  \clist_set:Nn \l_tmpa_clist { #2 }
  \exp_args:NNnx
  \group_end:
  \CDR_gput:nn { #1 } { \clist_use:Nn \l_tmpa_clist , }
}
```

29.4 Retrieving properties

\CDR:nn ⋆ \CDR:nn {⟨*name*⟩} {⟨*key*⟩}
\CDR:n ⋆ \CDR:n {⟨*key*⟩}

The value previously stored for the given ⟨*name*⟩ and ⟨*key*⟩ is left in the input stream. If nothing was previously store, the property for `default` instead of ⟨*name*⟩, and ⟨*key*⟩ is left in the input stream if any.

```
\cs_new:Npn \CDR:nn #1 #2 {
  \cs_if_exist:cTF { CDR.var/#1/#2 } {
    \tl_use:c { CDR.var/#1/#2 }
  } {
    \cs_if_exist:cT { CDR.var/default/#2 } {
      \tl_use:c { CDR.var/default/#2 }
    }
  }
}
\cs_new:Npn \CDR:n {
  \exp_args:NV \CDR:nn \g_CDR_name_tl
}
```

29.5 Property conditionals

\CDR_when:nn*TF* ⋆ \CDR_when:nTF {⟨*key*⟩} {⟨*true code*⟩} {⟨*false code*⟩}
\CDR_when:n*TF* ⋆

Execute ⟨*true code*⟩ when there is a property for ⟨*key*⟩, ⟨*false code*⟩ otherwise.

```
\prg_new_conditional:Nnn \CDR_when:nn { T, F, TF } {
  \cs_if_exist:cTF { CDR.var/#1/#2 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
\prg_new_conditional:Nnn \CDR_when:n { T, F, TF } {
  \cs_if_exist:cTF { CDR.var/\g_CDR_name_tl/#1 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

\CDR_when:nnNTF ⋆ \CDR_when:nNTF ⋆

\CDR_when:nNTF {⟨name⟩} {⟨key⟩} ⟨tl var⟩ {⟨true code⟩} {⟨false code⟩}
\CDR_when:nNTF {⟨key⟩} ⟨tl var⟩ {⟨true code⟩} {⟨false code⟩}

Execute ⟨true code⟩ when the property for given ⟨name⟩ and ⟨key⟩ is retrieved into ⟨tl var⟩, ⟨false code⟩ otherwise.

```
\prg_new_conditional:Nnn \CDR_when:nnN { T, F, TF } {
  \cs_if_exist:cTF { CDR.var/#1/#2 } {
    \tl_set_eq:Nc #3 { CDR.var/#1/#2 }
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
\prg_new_conditional:Nnn \CDR_when:nN { T, F, TF } {
  \cs_if_exist:cTF { CDR.var/\g_CDR_name_tl/#1 } {
    \tl_set_eq:Nc #2 { CDR.var/\g_CDR_name_tl/#1 }
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

\CDR_if:nTF ⋆

\CDR_if:nTF {⟨key⟩} {⟨true code⟩} {⟨false code⟩}

Execute ⟨true code⟩ if the property for ⟨key⟩ is truthy, ⟨false code⟩ otherwise. Mainly for boolean properties.

```
\prg_new_conditional:Nnn \CDR_if:n { T, F, TF } {
  \group_begin:
  \CDR_when:nNTF { #1 } \l_tmpa_tl {
    \exp_args:NnV
    \regex_match:nnTF { ^\s*[tTyY] } \l_tmpa_tl
      { \group_end: \prg_return_true:  }
      { \group_end: \prg_return_false: }
  } { \group_end: \prg_return_false: }
}
```

29.6 Properties and TEX groups

Removing a property at a group level is not straightforward. Once a property has been modified inside a TEX group, the value outside the group is definitely overriden and is no longer available until the end of the group.

30 Constants

\c_CDR_comment_prop

One line comment marker per language.

```
\prop_const_from_keyval:Nn \c_CDR_comment_prop {
  tex=\c_percent_str,
  lua=--,
  python=\c_hash_str,
  c=//,
```

```
  c++=//,
  javascript=//,
}
```

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

31 REMAINING

`\l_CDR_code_name_tl` Locally used as ⟨*name*⟩ in `\g/CDR/Code/<name>/` `\g/CDR/int/<name>` and similar.

```
\tl_new:N \l_CDR_code_name_tl
```

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

32 l3keys modules

The various l3keys modules define the LaTeX user interface of commands and environments.

32.1 Utilities

Values are stored in some property list, the key gives a hint on the type.

`\CDR_put_boolean:nn`
`\CDR_put_clist:nn`

```
\CDR_put_bool:nn {⟨key⟩} {⟨value⟩}
\CDR_put_inverse_bool:nn {⟨key⟩} {⟨value⟩}
\CDR_put_clist:nn {⟨key⟩} {⟨value⟩}
```

The ⟨*key*⟩ is appended with the data type. The value is obtained by `\key_set:nn`. They rely on `\CDR_put:nn`.

```
\cs_new:Npn \CDR_put_bool:nn #1 #2 {
  \regex_match:nnTF { ^[tTyY] } { #1 } {
    \CDR_set:nn {#1_bool} { \c_true_bool }
  } {
    \CDR_set:nn {#1_bool} { \c_false_bool }
  }
}
\cs_new:Npn \CDR_put_inverse_bool:nn #1 #2 {
  \regex_match:nnTF { ^[tTyY] } { #1 } {
    \CDR_set:nn {#1_bool} { \c_false_bool }
  } {
    \CDR_set:nn {#1_bool} { \c_true_bool }
  }
}
\cs_new:Npn \CDR_put_clist:nn #1 #2 {
  \group_begin:
  \clist_set:Nn \l_tmpa_clist { #2 }
  \exp_args:NNnC
  \group_end:
  \CDR_set:nn {#1_clist} \l_tmpa_clist
}
```

```
% ==========================================================
% 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>
```