User manual for the coder package

Jérôme LAURENS*

April 11, 2022

Contents

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

1 Installation

This LuaLaTeX package is part of any standard TeX distribution. To use it in a document, put the next instruction in the preamble:

```
\RequirePackage{coder}
```

and run LuaLaTeX. In order to have syntaxe highlighting like above, you must have pygments installed (see https://pygments.org for that purpose) but this is not a requirement for the other features of the coder package. To test pygments installation, you can run from a terminal

`pygmentize -O full -o` ⟨*file name*⟩`-colorized.tex` ⟨*file name*⟩`.tex`

followed by

`latex` ⟨*file name*⟩`-colorized.tex`.

The colorized code is then found in ⟨*file name*⟩`-colorized.pdf`

To test the installation for an editor, the next document should output a small diagnostic page with informations about paths.

```
\documentclass{article}
\RequirePackage{coder}
\begin{document}
\CDRTest
\end{document}
```

- An example of a bad installation:

  Path to python: `/usr/bin/python`
  Path to pygmentize:
  Pygments is not available

- An example of a good installation:

  Path to python: `/opt/anaconda3/bin/python`
  Path to pygmentize: `/opt/anaconda3/bin/pygmentize`
  Pygments is available

To use pygments, some editors need to be launched from a terminal, like TeXworks on OSX for example. An alternate solution is to add to the preamble of each document

`\CDRSet{python path=`⟨*path to python with pygmentize*⟩`}`

Notice that such a command can be put in a `coder.cfg` file accessible to LuaLaTeX. When available this configuration file is automatically loaded by package coder.

2 Display code

2.1 Example

To see coder in action, we compare in figure 1 lua and python syntax while computing recursively the factorial of an integer. The coloring is made by pygments using its built in style `autumn` on the left and style `trac` on the right. Line numbering may appear on each side of the code, or may be hidden.

2.2 `\CDRCode` command

Inserting code as inline text is possible with the command `\CDRCode` which syntax is

`\CDRCode[`⟨*key*⟩$_1$`=`⟨*value*⟩$_1$`,`⟨*key*⟩$_2$`=`⟨*value*⟩$_2$`,...]?`⟨*inline code chunk*⟩`?`

Here the question mark `?` stands for quite any unicode character, at least one that is not in ⟨*inline code chunk*⟩ and is not a `[` of course. Similarly, we can save a code snippet for later use:

lua:
```lua
function factorial(n)
  --[[Compute n!]]
  if n > 1 then
    return n * factorial(n-1)
  end
  return 1
end
```

: python
```python
def factorial(n):
  '''Compute n!'''
  if n > 1:
    return n * factorial(n-1)
  return 1
```

Figure 1: First example

`\CDRCodeSave{UNIK key}?\textbf{`⟨*text*⟩`}?`

With next instruction,

`\CDRCodeUse[`⟨*key*⟩$_1$`=`⟨*value*⟩$_1$`,`⟨*key*⟩$_2$`=`⟨*value*⟩$_2$`,...]{UNIK key}`

we can display the code snippet with different styles like `\textbf{`⟨*text*⟩`}` or `\textbf{`⟨*text*⟩`}`, and even use it in a footnote[1].

The available options are collected in section 6.5 because they are shared.

2.3 The CDRBlock environment

Code chunks are put in a `CDRBlock` LaTeX environment. Like many verbatim envitonments, the closing command is a full line literally consisting of `\end{CDRBlock}`, with no extra space. Optional key–value options are enclosed with square brackets. The opening bracket must follow the `\begin{CDRBlock}` instruction, on the very same line. For the first examples of figure 1, the input was

```
\begin{CDRBlock}[tags=lua]
function factorial(n)
  --[[Compute n!]]
  if n > 1 then
    return n * factorial(n-1)
  end
  return 1
end
\end{CDRBlock}
```

```
\begin{CDRBlock}[tags=python]
def factorial(n):
  '''Compute n!'''
  if n > 1:
    return n * factorial(n-1)
  return 1
\end{CDRBlock}
```

Figure 2: Source of figure 1

Similarly to `\CDRCodeSave` there is a `CDRBlockSave` environment which unique mandatory argument is an identifier

`\begin{CDRBlockSave}{`⟨*identifier*⟩`}`
...
`\end{CDRBlockSave}`

Next command allows to display that source with any option available

`\CDRBlockUse[`⟨*key*⟩$_1$`=`⟨*value*⟩$_1$`,`⟨*key*⟩$_2$`=`⟨*value*⟩$_2$`,...]{`⟨*identifier*⟩`}`

[1] Here is the aspect when pygments is not available: `\textbf{`⟨*text*⟩`}`

3 Export code with \CDRExport

3.1 Usage

The syntax for the **\CDRExport** command is the following, where the `file` key must be provided with a non empty value,

```
\CDRExport{
  file={⟨output file path⟩},
  tags=⟨tag⟩1|⟨tag⟩2|...,
}
```

At the end of the typesetting process, all the code snippets with $\langle tag \rangle_1$ like

```
\begin{CDRBlock}[tags=⟨some tag⟩|⟨tag⟩1|⟨other tag⟩|...]
...
\end{CDRBlock}
```

are collected in the order of the source, then all the code chunks with $\langle tag \rangle_2$ are collected in turn,... Finally, the result is written to ⟨*output file path*⟩.

If the `tags` list is void, no exportation takes place. This exportation list can contain macros.

3.2 Options

`lang=`⟨*language*⟩ One of the languages pygments is aware of, the list of officially supported languages is available at https://pygments.org/languages/. Initially `tex`. Every subsequent code chunk which first tag is in the exportation list will have the same `lang` option, unil the next change.

`preamble=`⟨*preamble text*⟩**`, preamble file=`**⟨*preamble file path*⟩ The ⟨*preamble text*⟩ is saved before the collected code chunks unless the `raw` option is set to `true`. When a ⟨*preamble file path*⟩ is given, then the ⟨*preamble text*⟩ is taken from the file instead. Initially empty.

`postamble=`⟨*postamble text*⟩**`, postamble file=`**⟨*postamble file path*⟩ the ⟨*postamble text*⟩ is saved after the code collected chunks unless the `raw` option is set to `true`. When a ⟨*preamble file path*⟩ is given, then the ⟨*preamble text*⟩ is taken from the file instead. Initially empty.

`escapeinside=`⟨*delimiters*⟩ the preamble and the postamble can contain LaTeX instructions enclosed between the delimiters, these will be executed before exportation. Not any command is suitable here, macros containing text, **\date** may be convenient here. When the ⟨*delimiters*⟩ is void, no escaping occurs. When ⟨*delimiters*⟩ contains only one character, it is used both as opening and closing delimiter. When it contains at leat two characers, the first one is used as opening delimiter, the second one as closing delimiter.

`raw[=true|false]` When `true`, a preamble and postamble will be added to all the collected code chunks. Initially `false`.

`once[=true|false]` Consider for example the exportation `tags` list $\langle tag \rangle_1|\langle tag \rangle_2$. If a block code chunk has exactly the same tag list, then it should be exported when $\langle tag \rangle_1$ codes are collected but also when $\langle tag \rangle_2$ codes are collected. If the `once` option is set to `true`, such a code chunk will only be exported the first time, and will be ignored afterwards. Initially `true`: export only once.

4 Decorations

4.1 Tags

For the `CDRBlock` environment, the `show tags` option allows to display the list of tags, see section 6.3 for more details about tags. The possible choices are illustrated hereafter below each source.

```
\begin{CDRBlock}[
  tags = dummy,
  show tags=none,
]
no tags
\end{CDRBlock}
```

no tags

```
\begin{CDRBlock}[
  tags = dummy,
  show tags=left,
]
tags on the left
\end{CDRBlock}
```

dummy: tags on the left

```
\begin{CDRBlock}[
  tags = dummy,
  show tags=right,
]
tags on the right
\end{CDRBlock}
```

tags on the right : dummy

```
\begin{CDRBlock}[
  tags = dummy,
  show tags=same,
]
tags like numbers
\end{CDRBlock}
```

dummy: 1 tags like numbers

```
\begin{CDRBlock}[
  tags = dummy,
  show tags=same,
]
tags like numbers
\end{CDRBlock}
```

tags like numbers 1: dummy

```
\begin{CDRBlock}[
  tags = dummy,
  show tags=mirror,
]
tags in mirror
\end{CDRBlock}
```

1 tags in mirror : dummy

```
\begin{CDRBlock}[
  tags = dummy,
  show tags=mirror,
]
tags in mirror
\end{CDRBlock}
```

dummy: tags in mirror 1

4.2 Line numbering

Line numbering only makes sense for blocks of code. Options are illustrated below, to the right of each source.

```
\begin{CDRBlock}[
  tags=X,
  show tags=none,
  numbers=left,
  firstnumber=10,
  stepnumber=3,
]
...
\end{CDRBlock}
```

```
10 line 1 numbered 10
   line 2
12 line 3 numbered 12
   line 4
   line 5
15 line 6 numbered 15
   line 7
   line 8
18 line 9 numbered 18
```

Blocks with the same leftmost tag can be numbered continuously with option `firstnumber=last`, the next one starting just after the previous has stopped.

```
\begin{CDRBlock}[
  tags=X,
  show tags=none,
  numbers=left,
  firstnumber=last,
  stepnumber=4,
]
...
\end{CDRBlock}
```

```
19 line 1
20 ..... line 2 numbered 20
21 line 3
22 line 4
23 line 5
24 ..... line 6 numbered 24
25 line 7
26 line 8
27 line 9
```

Above, the intermediate line numbers were displayed by redefining the command `\CDRNumberOther` that defaults to no operation:

```
\RenewDocumentCommand\CDRNumberOther{m}{%
  \color{lightgray}#1%
}
```

Similarly, the command `\CDRNumberMain` to display the main line numbers has been redefined with the help of the mboxfill package:

```
\RenewDocumentCommand\CDRNumberMain{m}{%
  \color{⟨color⟩}\bfseries#1
  \color{.!25!white}\makebox[4mm]{\mboxfill[0.75mm]{.}}%
}
```

4.3 Frames

4.3.1 Inline code

When the efbox package is loaded, inline code can be framed by adding the option `engine=efbox`: `\textbf{Vestibulum porttitor.}`. It is possible to pass any option of the efbox package with the key `efbox engine options`, here is the source of the previous magenta box

```
\CDRCode[
  engine=efbox,
  efbox engine options={
    backgroundcolor=magenta!5!white,
    linecolor=magenta!80!black,
    linewidth=5\fboxrule,
  }
] |\textbf{Vestibulum porttitor.}|
```

This engine named `efbox` was declared by next command

```
\CDRCodeEngineNew {⟨engine name⟩} {
  ⟨engine instructions⟩
}
```

with `efbox` as ⟨*engine name*⟩, and `\efbox[#1]{#2}` as ⟨*engine instructions*⟩. It can be used to define a custom display engine as well.

There is an engine named `default` which is always available and used by default. `\CDRCodeEngineRenew` will be used to redefine engines, with a similar syntax.

4.3.2 Block code

If the tcolorbox package is loaded, then blocks of code can be framed by adding the option `engine=tcbox`

Example

```
\textbf{Lorem ipsum dolor sit amet, consectetuer adipiscing elit.}
```

It is possible to pass any option of the tcolorbox package with the key `engine options`, here is the source of the magenta box above

```
\begin{CDRBlock}[
  label=Example,
  engine=tcbox,
  engine options={
    colback=magenta!5!white,
    colframe=magenta!80!black,
    boxrule=5\fboxrule,
    fontupper=\sffamily\bfseries,
    title=\CDRGetOption{label},
  },
]
\textbf{Vestibulum porttitor.}
\end{CDRBlock}
```

This engine named `tcbox` was declared by next command

```
\CDRBlockEngineNew {⟨engine name⟩} {
  ⟨begin engine instructions⟩
} {
  ⟨end engine instructions⟩
}
```

with `tcbox` as ⟨*engine name*⟩, `\begin{tcolorbox}[#1]` as ⟨*begin engine instructions*⟩, where `#1` will be replaced by whatever is provided for key ⟨`engine name`⟩ `engine options`, and as ⟨*end engine instructions*⟩ `\end{tcolorbox}`. This command is availale to define a custom display engine as well.

There is an engine named `default` which is always available and used by default. The command `\CDRBlockEngineRenew` will be used to redefine engines, with a similar syntax. Notice the `\CDRGetOption` command used to retrieve the value for the key `label`.

The coder package also supports fancyvrb options to display frames, here is an example taken from the fancyvrb documentation:

```
Verbatim line.
```

```
\begin{CDRBlock}[
  frame=single,
  framerule=1mm,
  framesep=3mm,
  rulecolor=\color{red},
  fillcolor=\color{yellow}
]
Verbatim line.
\end{CDRBlock}
```

5 Miscellanées

5.1 Filtering

One can display only a selected range of lines.

```
\begin{CDRBlock}[
  firstline=2,
  lastline=4,
]
Line 1
Line 2 *
Line 3 *
Line 4 *
Line 5
\end{CDRBlock}
```

The output reads

```
Line 2 *
Line 3 *
Line 4 *
```

```
\begin{CDRBlock}[
  firstline=-3,
  lastline=-1,
]
Line 1
Line 2 *
Line 3 *
Line 4 *
Line 5
\end{CDRBlock}
```

The output reads

```
Line 2 *
Line 3 *
Line 4 *
```

```
\begin{CDRBlock}[
  firstline=L.*2,
  lastline=L.*4,
]
Line 1
Line 2 *
Line 3 *
Line 4 *
Line 5
\end{CDRBlock}
```

The output reads

```
Line 2 *
Line 3 *
Line 4 *
```

In the middle column, non positive integers count lines from the end. Notice that line numbering is 1 based such that the last line corresponds to index 0.

In the last column, the option `firstline=L.*2` means: from the first line that matches the regular expression pattern `"L.*2"`, according to LaTeX3 `interface3.pdf`. Similarly, the option `lastline=L.*4` means: up to the first line, from the line found above, that matches the regular expression pattern `"L.*2"`.

5.2 Spacing

For blocks, the size of the left and right margins can be adjusted. On the second line below were used the options `xleftmargin=3cm` and `xrightmargin=2cm`

normal margins: `One line.....`

adjusted margins: `One line.....`

adjusted margins: `One line.....`

The vertical alignment of the first and last tags on their right side is obtained with option `numbersep=\dimexpr1ex+\CDRGetOption{xleftmargin}\relax` for the third block. Notice the usage of `\CDRGetOption` to retrieve the value for the `xleftmargin` key.

The vertical space before and after the blocks is governed by `\topsep`, `\partopsep` and `\parskip` like standard list are. In addition, the `vspace` options, which initial value is exactly `\topsep`, allows a supplemental adjustment:

Nulla malesuada porttitor diam. `Default vspace` Quisque ullamcorper placerat ipsum.	Nulla malesuada porttitor diam. `vspace=5mm` Quisque ullamcorper placerat ipsum.	Nulla malesuada porttitor diam. `vspace=\dimexpr0.5\topsep-\partopsep\relax` Quisque ullamcorper placerat ipsum.

The line height is inherited from the surrounding environment unless the baselinestrech factor has been provided. Columns 2 and 3 are enclosed in a setspace environment with one and a half line spacing. The last column also adds the `baselinstretch=2` option.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
`Line 1`
`Line 2`
Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi.
	Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
	`Line 1`
	`Line 2`
	Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi.
		Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
		`Line 1`
		`Line 2`
		Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi.

When used in itemize and enumerate lists or similar environments, the `CDRBlock` environment ignores the indentation. With option `resetmargins=false` it adapts its width to the indentation as occurs on the right:

```
Line of code.....................
```

- One

```
Line of code.....................
```

- Two
  1. Three

```
Line of code.....................
```

```
Line of code.....................
```

- One

  ```
  Line of code.................
  ```

- Two
  1. Three

     ```
     Line of code............
     ```

5.3 Escaping to LaTeX

When some part of the code does not belong to the programming language, it is possible to break temporarily the syntax coloring process with the `escapeinside={`⟨*delimiters*⟩`}` where ⟨*delimiters*⟩ is a placeholder for a string composed of two different characters, possibly without surrounding braces. The delimiters are removed and what is between them is typeset by LaTeX.

The source code for the previous instruction was entered with

```
\CDRCode[escapeinside={()},...]|escapeinside={(\MyMeta{delimiters})}|
```

Where `\MyMeta` is a private command. The delimiters must not be part of the code, the unicode characters ⟨ and ⟩ are suitable amongst many others.

5.4 Cross references

We can always refer to the page of a specific code extract with the `reflabel=`⟨*label name*⟩ option. This is how we know that `\textbf{`⟨*text*⟩`}` was also used on page 3. For block code, this is similar:

```
\begin{CDRBlock}[reflabel=My]
Curabitur dictum gravida mauris.
\end{CDRBlock}
```

On the second typeset run, we have `\pageref{My}` = 10.

Line references are also available: `escapeinside` is used to define 2 characters that will delimit some LaTeX instructions to be executed, here `\label{line.421}`.

```
\begin{CDRBlock}[
  numbers=left,
  firstnumber=421,
  escapeinside=||,
]
line 421|\label{line.421}|
\end{CDRBlock}
```

which output simply reads

```
421 line 421
```

`\ref*{line.421}` = 421. However this does not play well with hyperref hence the starred command.

It can be used as well when pygments is not available.

```
\begin{CDRBlock}[
  numbers=left,
  firstnumber=123,
  escapeinside=||,
]
line 123|\label{line.123}|
\end{CDRBlock}
```

which output simply reads

```
123 line 123
```

`\ref*{line.123}` = 123. This does not play well with hyperref hence the starred command.

Cross references can be used with saved block, with some extra precaution. Here for example, the label is dynamically based on the macro `\My`.

```
\begin{CDRBlockSave}{MyBlock}
line 421 or 123|\label{line.\My}|
\end{CDRBlock}
```

First use:

```
\newcommand\My{A}
\CDRBlockUse[
  escapeinside=||,
  numbers=left,
  firstnumber=421,
]{MyBlock}
```

which reads

```
421 line 421 or 123
```

Second use:

```
\newcommand\My{B}
\CDRBlockUse[
  escapeinside=||,
  numbers=left,
  firstnumber=123,
]{MyBlock}
```

which reads

```
123 line 421 or 123
```

We get for the first use, `\ref*{line.A}` = 421 and for the second use `\ref*{line.B}` = 123.

6 The \CDRSet command

So far we have provided the various coder commands and environments with key–value options. The `\CDRSet` command allows to apply some setting once for all code snippets, moreover it can collect setting in style, already used above with tag names.

6.1 General settings

Next command will turn the color of any forthcoming code snippet in dark magenta.

```
\CDRSet{format=\color{magenta!10!black}}
```

The changes are local to the LaTeX environment where they are performed.

6.2 Styling

If some options may not be set globally and should only apply on demand, because for example different programming languages are used, then we specify a tag name like `lua` or `python`:

```
\CDRSet{
  tags/lua/lang=lua,
  tags/python/lang=python,
}
```

We can also specify at once many tag names separated by a `|`, each setting applying separately to each named tag.

```
\CDRSet{tags/lua|python/pygments=true}
```

Finally, changing many options at once for the same tag is also possible:

```
\CDRSet{tags/lua={
  showspaces=true,
  no export=true,
} }
```

6.3 Tags

Tags are used not only for styling but also for continuous line numbering (section 4.2) and exportation (section 3). If only one tag is used and there is no exportation, it can be omitted.

Tag names must not contain commas nor pipe characters, moreover, names with exactly two leading underscores are reserved by the coder package. More names are reserved by the coder package, but they are available to the user. They need not appear in a `tags=...` setting. Options set for the tag name `default` automatically apply to any forthcoming code snippet. Any option set for the tag name `default.code` applies to any forthcoming code snippet displayed inline, eventually overriding the `default` setting. Finally, options set for tag name `default.block` apply to any forthcoming code snippet displayed in block, taking precedence over the `default` setting.

6.4 Engines options

Engine options are provided to the `CDRBlock` environment and the `\CDRCode` , `\CDRCodeUse` and `\CDRBlockUse` commands with key `engine options`. This does not refer to the engine name and is not suitable for the `\CDRSet` command argument. Instead, one will use the key ⟨`engine name`⟩ `engine options` to distinguish between the engines.

6.5 Options

6.5.1 `default tag`

`tags=`⟨*tag*⟩$_1$`|`⟨*tag*⟩$_2$`|...` used for exportation (section 3), for line numbering (section 4.2) and also for styling, see section 6.2. Initially a void list. ⟨*tag*⟩ is subject to minor restrictions (section 6.3).

`engine=`⟨*engine name*⟩ to specify the engine used to display inline code or blocks, see section 4.3. Initially `default`.

⟨`engine name`⟩ **`engine options=`**⟨*engine options*⟩ to specify the options that should apply when the engine named ⟨*engine name*⟩ is selected, see section 4.3.

`default engine options=`⟨*engine options*⟩ to specify the options for the default engine which is named `default`, see section 4.3. Initially empty, depends on the engine used.

`engine options=`⟨*engine options*⟩ options forwarded to the engine, see section 4.3. They are appended to the options given with key ⟨`engine name`⟩ `engine options`. Mainly a convenient user interface shortcut. Suitable for `\CDRCode` and `CDRBlock` but not for `\CDRSet`.

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

`debug[=true|false]` Set to `true` if various debugging messages should be printed to the console. Initially `false`.

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

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

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

`cache[=true|false]` Set to `true` if the coder package should use already existing files instead of creating new ones. When using pygments for syntax coloring the file ⟨`file name`⟩`.tex`, the companion script `coder-tool.py` creates a folder named ⟨`file name`⟩`.pygd`. This is where intermediate files are stored and kept from one typesetting process to the next, which happens to save processing time. This folder can safely be removed, besides it will automatically be cleaned if there is no ⟨`file name`⟩`.aux` file yet. The option `cache=false` will disable this caching feature.

Initially `true`.

`escapeinside=`⟨*delimiters*⟩ 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 when pygments is used. Initially empty.

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

```
\begin{CDRBlock}[
  fontfamily=courier,
]
This is courier
\end{CDRBlock}
```

The output reads

This is NOT courier

```
\begin{CDRBlock}[
  fontfamily=helvetica,
]
This is helvetica
\end{CDRBlock}
```

The output reads

This is helvetica

```
\begin{CDRBlock}[
  fontfamily=TGC,
]
This is TeX Gyre Cursor
\end{CDRBlock}
```

The output reads

This is TeX Gyre Cursor

The `TGC` font family was created with package fontspec and instruction

```
\newfontfamily\TGCFont{TeX Gyre Cursor}[NFSSFamily=TGC]
```

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 below, the option used is `fontsize=\relsize{-2}` on the right, compared to `fontsize=\small` on the left). Initially `auto`: the same as the current font.

```
Nulla malesuada porttitor diam.
```

```
Nulla malesuada porttitor diam.
```

fontshape=auto|up|it|sl|sc font shape to use. Initially `auto`: the same as the current font. Here is the difference between slanted with `fontshape=sl` and italic with `fontshape=it`.

Nulla malesuada porttitor diam.

Nulla malesuada porttitor diam.

fontseries=auto|bf|md|lf LaTeX font 'series' to use. Initially `auto`: the same as the current font.

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

```
\textbf{In␣eu␣orci␣massa}
```

```
\textbf{In␣eu␣orci␣massa}
```

reflabel=⟨*label*⟩ define a label to be used with `\pageref`. Initially empty. See section 5.4.

6.5.2 `default.code` tag

mbox[=true|false] When set to `true`, put the argument inside a LaTeX `\mbox` to prevent the code snippet to spread over different lines. Use option `mbox=false` to allow line breaking like this `Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl.` Initially `true`: no line breaking.

6.5.3 `default.block` tag

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

`no export format=`⟨*format commands*⟩ a list of formatting instructions appended to `format`, `tags format` and `numbers format` when `no export` is `true`. Initially empty.

`no top space[=true|false]` When `true`, there is no separation top space before the block. Initially `false`. See section 5.2.

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

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

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

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

`gobble=`⟨*integer*⟩ number of characters to suppress at the beginning of each line (from 0 to 9), mainly useful when environments are indented. Sole option of the `CDRBlockSave` environment.

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

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

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

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

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

`vspace=`⟨*dimension*⟩ the amount of vertical space added to `\parskip` before and after blocks. Initially `\topsep`. See section 5.2.

`samepage[=true|false]` in very special circumstances, we may want to make sure that a block of code is not spread over multiple pages. To avoid a page break as far as possible, set the `samepage` option to `true`. Initially `false`.

`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.

Next options are illustrated in section **??**.

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

numbersep=⟨*dimension*⟩ gap between numbers and verbatim lines. Initially `1ex`.

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

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

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

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

dry numbers[=true|false] When `true`, line numbers are not collected for further continuous line numbering. Initially `false`.

```
\begin{CDRBlock}[
  firstnumber=421,
]
LINE 421
LINE 422
LINE 423
\end{CDRBlock}
```

it reads

```
421 LINE 421
422 LINE 422
423 LINE 423
```

```
\begin{CDRBlock}[
  firstnumber=last,
  dry numbers,
]
LINE 424
LINE 425
LINE 426
\end{CDRBlock}
```

it reads

```
424 LINE 424
425 LINE 425
426 LINE 426
```

```
\begin{CDRBlock}[
  firstnumber=last,
]
LINE 424
LINE 425
LINE 426
\end{CDRBlock}
```

it also reads

```
424 LINE 424
425 LINE 425
426 LINE 426
```

The line numbering of the third column follows the first one and ignore the block with the `dry numbers`.

Next options have no effect when pygments is used. Moreover, the tcolorbox package should be preferred instead.

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.

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

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

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

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

labelposition=none|topline|bottomline|all position where to print the label(s) when defined. When options happen to be contradictory, like `labelposition=bottomline` and `frame=topline`, nothing is displayed. Initially `none` when no labels are defined, `topline` for one label and `all` otherwise.