The codelst Package

A **Typst** package to render source code

v0.0.5 2023-07-19

Jonas Neugebauer

https://github.com/jneug/typst-codelst

codelst is a **Typst** package inspired by LaTeX package like listings. It adds functionality to render source code with line numbers, highlighted lines and more.

Table of contents

Part I.
About

This package was created to render source code on my exercise sheets for my computer science classes. The exercises required source code to be set with line numbers that could be referenced from other parts of the document, to highlight certain lines and to load code from files into my documents.

Since I used LaTeX before, I got inspired by packages like LISTINGS[1] and attempted to replicate some of its functionality. CODELST is the result of this effort.

[1] https://ctan.org/pkg/listings

Part II.
Usage

> Version **0.0.5** introduced a new method for rendering raw text, that deals with some of the issues of previous methods. This did break some other features of the package, mostly related to formatting. To cleanup the codebase some other breaking changes (e.g. renaming of argeuments) where done.

II.1. Use as a package (Typst 0.6.0 and later)

For **Typst** 0.6.0 and later codelst can be imported from the preview repository:

```
#import "@preview/codelst:0.0.4": sourcecode
```

Alternatively, the package can be downloaded and saved into the system dependent local package repository.

Either download the current release from GitHub[2] and unpack the archive into your system dependent local repository folder[3] or clone it directly:

```
git clone https://github.com/jneug/typst-codelst.git codelst-0.0.4
```

In either case, make sure the files are placed in a folder with the correct version number: `codelst-0.0.4`

After installing the package, just import it inside your `typ` file:

```
#import "@local/codelst:0.0.4": sourcecode
```

II.2. Use as a module

To use codelst as a module for one project, get the file `codelst.typ` from the repository and save it in your project folder.

Import the module as usual:

```
#import "codelst.typ": sourcecode
```

II.3. Rendering source code

codelst adds the `#sourcecode()` command with various options to render code blocks. It wraps around any `#raw()` block to add some functionality and formatting options to it:

[2] https://github.com/jneug/typst-codelst/releases/latest
[3] https://github.com/typst/packages#local-packages

```
#sourcecode[```typ
  #show "ArtosFlow": name => box[
    #box(image(
      "logo.svg",
      height: 0.7em,
    ))
    #name
  ]

  This report is embedded in the
  ArtosFlow project. ArtosFlow is a
  project of the Artos Institute.
```]
```

---

```
#show "ArtosFlow": name => box[
  #box(image(
    "logo.svg",
    height: 0.7em,
  ))
  #name
]

This report is embedded in the
ArtosFlow project. ArtosFlow is a
project of the Artos Institute.
```

CODELST add line numbers and some formatting to the code. Line numbers can be configured with a variety of options and `frame` sets a custom wrapper function for the code. Setting `frame: none` disables the code frame.

```
#sourcecode(
    numbers-side: right,
    numbering: "I",
    numbers-start: 10,
    numbers-first: 2,
    numbers-step: 4,
    numbers-style: (i) => align(right, text(fill:blue, emph(i))),
    frame: none
)[```typ
#show "ArtosFlow": name => box[
  #box(image(
    "logo.svg",
    height: 0.7em,
  ))
  #name
]

This report is embedded in the
```
```

```
19 ArtosFlow project. ArtosFlow is a
20 project of the Artos Institute.
21 ```]
```

```
#show "ArtosFlow": name => box[
  #box(image(                                                    XI
    "logo.svg",
    height: 0.7em,
  ))
  #name                                                          XV
]

This report is embedded in the
ArtosFlow project. ArtosFlow is a                                XIX
project of the Artos Institute.
```

Since it is common to highlight code blocks by putting them inside a `#block()` element, CODELST does so with a light gray background and a border.

The frame can be modified by setting `frame` to a function with ine argument, a custom wrapper can be specified. To do this globally you can create your own `#sourcecode()` command:

```
1  #let codelst-sourcecode = sourcecode
2  #let sourcecode = codelst-sourcecode.with(
3    frame: block.with(
4      fill: fuchsia.lighten(96%),
5      stroke: 1pt + fuchsia,
6      radius: 2pt,
7      inset: (x: 10pt, y: 5pt)
8    )
9  )
10
11 #sourcecode[```typ
12 #show "ArtosFlow": name => box[
13   #box(image(
14     "logo.svg",
15     height: 0.7em,
16   ))
17   #name
18 ]
19
20 This report is embedded in the
21 ArtosFlow project. ArtosFlow is a
22 project of the Artos Institute.
23 ```]
```

```
1  #show "ArtosFlow": name => box[
2    #box(image(
```

```
    "logo.svg",
    height: 0.7em,
  ))
  #name
]

This report is embedded in the
ArtosFlow project. ArtosFlow is a
project of the Artos Institute.
```

Line numbers can be formatted a similar way or globally with a #show rule:

````
#show <line-number>: (lno) => text(fill:luma(120), size:10pt, emph(lno)
+ sym.arrow.r)

#sourcecode(gutter:2em)[```typ
#show "ArtosFlow": name => box[
  #box(image(
    "logo.svg",
    height: 0.7em,
  ))
  #name
]

This report is embedded in the
ArtosFlow project. ArtosFlow is a
project of the Artos Institute.
```]
````

```
→    #show "ArtosFlow": name => box[
→      #box(image(
→        "logo.svg",
→        height: 0.7em,
→      ))
→      #name
→    ]
→
→    This report is embedded in the
→    ArtosFlow project. ArtosFlow is a
→    project of the Artos Institute.
```

CODELST handles whitespace in the code to save space and view the code as intended (and indented), even if tabs are used. Unnecessary blank lines at the beginning and end will be removed, alongside superfluous indention:

````
#sourcecode[```java
````
````

```
2
3       class HelloWorld {
4         public static void main( String[] args ) {
5           System.out.println("Hello World!");
6         }
7       }
8
9
10  ```]
```

```
1  class HelloWorld {
2    public static void main( String[] args ) {
3      System.out.println("Hello World!");
4    }
5  }
```

This behavior can be disabled or modified:

```
1  #sourcecode(showlines:true, gobble:1, tab-indent:4)[```java
2
3       class HelloWorld {
4         public static void main( String[] args ) {
5           System.out.println("Hello World!");
6         }
7       }
8
9
10  ```]
```

```
1
2       class HelloWorld {
3           public static void main( String[] args ) {
4               System.out.println("Hello World!");
5           }
6       }
7
8
```

To show code from a file load it with `#read()` and pass the result to `#sourcefile()` alongside the filename:

```
1  #sourcefile(read("typst.toml"), file:"typst.toml")
```

```toml
[package]
name = "codelst"
version = "0.0.5"
entrypoint = "codelst.typ"
authors = ["Jonas Neugebauer"]
license = "MIT"
description = "A typst package to render sourcecode"
repository = "https://github.com/jneug/typst-codelst"
exclude = ["example.typ", "example.pdf", "manual.pdf", "manual.typ",
"tbump.toml"]
```

Its useful to define an alias for `#sourcefile()`:

```
let codelst-sourcefile = sourcefile
let sourcefile( filename, ..args ) = codelst-sourcefile(
  read(filename), file:filename, ..args
)
```

`#sourcefile()` takes the same arguments as `#sourcecode()`. For example, to limit the output to a range of lines:

```
#sourcefile(
  showrange: (2, 4),
  read("typst.toml"),
  file:"typst.toml"
)
```

```toml
name = "codelst"
version = "0.0.5"
entrypoint = "codelst.typ"
```

Specific lines can be highlighted:

```
#sourcefile(
  highlighted: (2, 3, 4),
  read("typst.toml"),
  file:"typst.toml"
)
```

```toml
[package]
name = "codelst"
version = "0.0.5"
entrypoint = "codelst.typ"
authors = ["Jonas Neugebauer"]
```

```
6 license = "MIT"
7 description = "A typst package to render sourcecode"
8 repository = "https://github.com/jneug/typst-codelst"
9 exclude = ["example.typ", "example.pdf", "manual.pdf", "manual.typ",
  "tbump.toml"]
```

To reference a line from other parts of the document, CODELST looks for labels in the source code and makes them available to TYPST. The regex to look for labels can be modified to be compatible to different source syntaxes:

```
1 #sourcefile(
2   label-regex: regex("\"(codelst.typ)\""),
3   highlight-labels: true,
4   highlight-color: lime,
5   read("typst.toml"),
6   file:"typst.toml"
7 )
8
9 See #lineref(<codelst.typ>) for the _entrypoint_.
```

```
1 [package]
2 name = "codelst"
3 version = "0.0.5"
4 entrypoint =
5 authors = ["Jonas Neugebauer"]
6 license = "MIT"
7 description = "A typst package to render sourcecode"
8 repository = "https://github.com/jneug/typst-codelst"
9 exclude = ["example.typ", "example.pdf", "manual.pdf", "manual.typ",
  "tbump.toml"]
```

See line 4 for the *entrypoint*.

II.4. Formatting

If `#sourcecode()` can be used inside `#figure()` and will show the correct supplement. It is recommended to allow page breaks for `raw` figures:

```
1 #show figure.where(kind: raw): set block(breakable: true)
```

Instead of the build in styles, custom functions can be used:

```
1 #show <line-number>: (i) => i.counter.display(
2     (n, ..args) => text(
3         fill:rgb(220, 65, 241),
```

```
        font:("Comic Sans MS"),
        str(n)
    )
)

#sourcecode(frame: (code) => block(
    width:100%,
    inset:(x:10%, y:0pt),
    block(fill: green, width:100%, code)
), raw("*some*
_source_
= code", lang:"typc"))
```

```
*some*
_source_
= code
```

Note that the style function for line numbers receives the result of a call to `#counter.display()`. The `counter` can be accessed via the `counter` attribute.

Using other packages like SHOWYBOX is easy:

```
#import "@preview/showybox:0.2.0": showybox

#let showycode = sourcecode.with(
  frame: (code) => showybox(
    frame: (
      upper-color: red.darken(40%),
      lower-color: red.lighten(90%),
      border-color: black,
      width: 2pt
    ),
    title: "Source code",
    code
  )
)

#showycode[```typ
*some*
_source_
= code
```]
```

**Source code**

```
some
source
= code
```
```

This is nice in combination with figures:

```
#import "@preview/showybox:0.2.0": showybox

#show figure.where(kind: raw): (fig) => showybox(
  frame: (
    upper-color: red.darken(40%),
    lower-color: red.lighten(90%),
    border-color: black,
    width: 2pt
  ),
  title: [#fig.caption #h(1fr) #fig.supplement #fig.counter.display()],
  fig.body
)

#figure(
  sourcecode(frame: none)[```typ
    *some*
    _source_
    = code
  ```],
  caption: "Some code"
)
```

---

**Some code** **Listing 1**

```
*some*
_source_
= code
```

Using a `#show` rule to set all `#raw()` blocks inside `#sourcecode()` is not possible, since the command internally creates a new `#raw()` block and would cause Tyost to crash with an overflow error. Using a custom `lang` can work around this, though:

```
#show raw.where(lang: "clst-typ"): (code) => sourcecode(lang:"typ",
code)

```clst-typ
*some*
_source_
= code
```
```

---

```
some
source
= code
```
```

CODELST provides two ways to get around this issue, however. One is to setup a custom language that is directly followed by a collon and the true language tag:

```
:typ
*some*
_source_
= code
```

This is a robust way to send anything to CODELST. But since this might prevent proper syntax highlighting in IDEs, a reversed syntax is possible:

```
:codelst
*some*
_source_
= code
```

This will look at the first line of every `raw` text and if it matches `:codelst`, it will remove the activation tag and send the code to `#sourcecode()`.

Setting up one of these catchall methods is easily done by using the `#codelst()` function in a `#show` rule. Any arguments will be passed on to `#sourcecode()`:

```
#show: codelst( ..sourcecode-args )

// or

#show: codelst( reversed: true, ..sourcecode-args )
```

II.5. Command overview

```
#sourcecode(lang: auto,
  numbering: "1",
  numbers-start: auto,
  numbers-side: left,
  numbers-width: auto,
  numbers-style: "function",
  numbers-first: 1,
  numbers-step: 1,
  continue-numbering: false,
  gutter: 10pt,
  tab-indent: 2,
  gobble: auto,
  highlighted: (),
  highlight-color: rgb("#eaeabd"),
  label-regex: regex("// <([a-z-]{3,})>$"),
  highlight-labels: false,
  showrange: none,
```

```
showlines: false,
frame: "code-frame")[code]
```

`numbering: "1"` A `numbering pattern` to use for line numbers. Set to `none` to disable line numbers. `string` | `function` | `none`

`numbers-start: auto` The number of the first code line. If set to `auto`, the first line will be set to the start of `showrange` or `1` otherwise. `integer` | `auto`

`numbers-side: left|right` On which side of the code the line numbers should appear. `alignment`

`numbers-width: auto` The width of the line numbers column. Setting this to `auto` will measure the maximum size of the line numbers and size the column accordingly. Giving a negative length will move the numbers into the margin. `auto` | `length`

`numbers-style: (i) => i` A function of one argument to format the line numbers. Should return `content`. `function`

`continue-numbering: false` If set to `true`, the line numbers will continue from the last call of `#sourcecode()`. `boolean`

```
#sourcecode[```
one
two
```]
#lorem(10)
#sourcecode(continue-numbering:
true)[```
three
four
```]
```

```
one
two
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

```
three
four
```

`gutter: 10pt` Gutter between line numbers and code lines. `length`

`tab-indent: 4` Number of spaces to replace tabs at the start of each line with. `integer`

`gobble: auto` How many whitespace characters to remove from each line. By default, the number is automatically determined by finding the maximum number of whitespace all lines have in common. If `gobble: false`, no whitespace is removed. — `auto` | `integer` | `boolean`

`highlighted: ()` Line numbers to highlight. — `array`

Note that the numbers will respect `numbers-start`. To highlight the second line with `numbers-start: 15`, pass `highlighted: (17,)`

`highlight-color: rgb("#eaeabd")` Color for highlighting lines. — `color`

`label-regex` A `regular expression` for matching labels in the source code. The default value will match labels with at least three characters at the end of lines, separated with a line comment (`//`). For example: — `regular expression`

```
#strong[Some text] // <my-line-label>
```

If this line matches on a line, the full match will be removed from the output and the content of the first capture group will be used as the label's name (`my-line-label` in the example above).

Note that to be valid, the expression needs to have at least one capture group.

To reference a line, `#lineref()` should be used.

`highlight-labels: false` If set to `true`, lines matching `label-regex` will be highlighted. — `boolean`

`showrange: none` If set to an array with exactly two `integer`s, the code-lines will be sliced to show only the lines within that range. — `none` | `array`

For example, `showrange: (5, 10)` will only show the lines 5 to 10.

If settings this and `numbers-start: auto`, the line numbers will start at the number indicated by the first number in `showrange`. Otherwise, the numbering will start as specified with `numbers-start`.

`showlines: false` If set to `true`, no blank lines will be stripped from the start and end of the code. Otherwise, those lines will be removed from the output. — `boolean`

Line numbering will not be adjusted to the removed lines (other than with `showrange`).

#sourcefile(code, filename: none, lang: auto, ..args)

Takes a text string `code` loaded via the `#read()` function and passes it to `#sourcecode()` for display. If `filename` is given, the code language is guessed by the file's extension. Otherwise, `lang` can be provided explicitly.

Any other `args` will be passed to `#sourcecode()`.

```
#sourcefile(read("typst.toml"), lang:"toml")
```

```
[package]
name = "codelst"
version = "0.0.5"
entrypoint = "codelst.typ"
authors = ["Jonas Neugebauer"]
license = "MIT"
description = "A typst package to render sourcecode"
repository = "https://github.com/jneug/typst-codelst"
exclude = ["example.typ", "example.pdf", "manual.pdf",
"manual.typ", "tbump.toml"]
```

The original intend for `#sourcefile()` was, to raed the provided filename, without the need for the user to call `#read()`. Due to the security measure, that packages can only read files from their own directory, the call to `#read()` needs to happen outside of `#sourcefile()` in the document.

For this reason, the command differs from `#sourcecode()` only insofar as it accepts a `string` instead of raw `content`.

Future releases might use the `filename` for other purposes, though.

To deal with this, simply add the following code to the top of your document:

```
#let srcfile( filename, ..args ) = sourcefile(read(filename),
file:filename, ..args)
```

#lineref(label, supplement: "line")

Creates a reference to a labeled line in the source code. `label` is the label to reference.

```
#sourcecode[```java
class HelloWorld {
  public static void main( String[] args ) {
    System.out.println("Hello World!");
  }
}
```]

See #lineref(<main-method>) for a main method in Java.
```

---
```

```java
class HelloWorld {
  public static void main( String[] args ) {
    System.out.println("Hello World!");
  }
}
```

See line 3 for a main method in Java.

How to set labels for lines, refer to the documentation of `label-regex` at command `#sourcecode()` on page 12.

`#code-frame(fill: luma(250), stroke: 1pt + luma(200), inset: (x: 5pt, y: 10pt), radius: 4pt)[code]`

Applies the CODELST default styles to the document. Source code will be wrapped in `#code-frame()` and numbers styled with `#numbers-style()`.

```
#show <codelst>: code-frame.with(
  fill: gray,
  stroke: 2pt + lime,
  radius: 8pt
)
#sourcecode[```
some code
```]
```

---

```
some code
```

`#codelst-styles()[body]`

Applies the CODELST default styles to the document. Source code will be wrapped in `#code-frame()` and numbers styled with `#numbers-style()`.

```
#show: codelst-styles
```
```

Part III.
Limiations and alternatvies

III.1. Limitations

To render code with correct syntax highlighting and line numbers CODELST renders content line by line in a table. Since the complete code is rendered *once per line* (!), it has a lot of overhead. This also mostly prevents the selection of code in a PDF.

III.2. Alternatives

There are some alternatives to CODELST that fill similar purposes, but have more or other functionality. If CODELST does not suit your needs, one of those might do the trick.

platformer/typst-algorithms[4] *TYPST module for writing algorithms. Use the algo function for writing pseudocode and the code function for writing code blocks with line numbers.*

hugo-s29/typst-algo[5] *This package helps you typeset [pseudo] algorithms in TYPST.*

[4] https://github.com/platformer/typst-algorithms
[5] https://github.com/hugo-s29/typst-algo

Part IV.
Index