

inline — code inlined in a L^AT_EX document^{*}

Jérôme LAURENS[†]

Released ?

Abstract

Usually, documentation is put inside the code, `inline` 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 `inline-manual` gives different examples. Here is the implementation of the package.

This L^AT_EX package requires LuaT_EX and may use syntax coloring based on `pygment`.

1 Package dependencies

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

2 Similar technologies

The `docstrip` utility offers similar features, it is somehow more powerful than `inline` 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. `inline` focuses on code inlining and interfaces well with `pygment` for a smart syntax highlighting.

3 Known bugs and limitations

- `inline` does not play well with `docstrip`.

```
1 <*package>
2 \makeatletter
```

^{*}This file describes version ?, last revised ?.

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

4 Constants

`\c_NLN_comment_prop` One line comment marker per language.

```

3 \prop_const_from_keyval:Nn \c_NLN_comment_prop {
4   tex=\c_percent_str,
5   lua=--,
6   python=\c_hash_str,
7   c=//,
8   c++=//,
9   javascript=//,
10 }
```

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

5 Global properties

`\g/NLN/code/` Tree storage for global generic code properties or named code properties. These are overridden locally in environments using key-value actions. `\l_NLN_code_name_tl` is used as $\langle name \rangle$.

```

11 \prop_new:c {g/NLN/code/}
```

(End definition for `\g/NLN/code/` and `\g/NLN/code/<name>`. These variables are documented on page ??.)

`\l_NLN_code_name_tl` Locally used as $\langle name \rangle$ in `\g/NLN/code/<name>` `\g/NLN/int/<name>` and similar.

```

12 \tl_new:N \l_NLN_code_name_tl
```

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

5.1 Management

`\NLN_item:n` `\NLN_item:n {<key>}`

```

13 \cs_new:Npn \NLN_item:n #1 {
14   \prop_item:cn {g/NLN/code/} { #1 }
15 }
```

`\NLN_if_in:nTF` \star `\NLN_if_in:nTF {<key>} {<true code>} {<false code>}`

Execute $\langle true code \rangle$ when `\g/NLN/code/` prop's contains $\langle key \rangle$, $\langle false code \rangle$ otherwise.

```

16 \prg_new_conditional:Nnn \NLN_if_in:n { T, F, TF } {
17   \prop_if_in:NnTF {g/NLN/code/} { #1 } {
18     \prg_return_true:
19   } {
20     \prg_return_false:
21   }
22 }
```

| | |
|--|--|
| <u>\NLN_get:nnTF</u> ★ | <p>\NLN_get:nnTF {<key>} <tl var> {<true code>} {<false code>}</p> <p>Execute <true code> when \g/NLN/code/ prop's <key> is retrieved in <tl var>, <false code> otherwise.</p> <pre> 23 \prg_new_conditional:Nnn \NLN_get:nn { T, F, TF } { 24 \prop_get:cnNTF {g/NLN/code/} { #1 } #2 { 25 \prg_return_true: 26 } { 27 \prg_return_false: 28 } 29 }</pre> |
| <u>\NLN_item:nn</u> ★ | <p>\NLN_item:nn {<name>} {<key>}</p> <p><name> is a code name.</p> <pre> 30 \cs_new:Npn \NLN_item:nn #1 #2 { 31 \prop_item:cn { g/NLN/code/#1 } { #2 } 32 }</pre> |
| <u>\NLN_get:nnNTF</u> | <p>\NLN_get:nnNTF {<name>} {<key>} <tl var> {<true code>} {<false code>}</p> <p>Execute <true code> when g/NLN/code/{<meta>} prop's <key> is retrieved in <tl var>, <false code> otherwise.</p> <pre> 33 \prg_new_conditional:Nnn \NLN_get:nnN { T, F, TF } { 34 \prop_get:cnNTF { g/NLN/code/#1 } { #2 } #3 { 35 \prg_return_true: 36 } { 37 \prg_return_false: 38 } 39 }</pre> |
| <u>\NLN_put:nn</u> <u>\NLN_put:nV</u> <u>\NLN_gput:nn</u> <u>\NLN_gput:nV</u> | <p>\NLN_put:nn {<key>} {<value>}</p> <p>\NLN_gput:nn {<key>} {<value>}</p> <pre> 40 \cs_new:Npn \NLN_put:nn #1 #2 { 41 \prop_put:cnn {g/NLN/code/} { #1 } { #2 } 42 } 43 \cs_new:Npn \NLN_gput:nn #1 #2 { 44 \prop_gput:cnn {g/NLN/code/} { #1 } { #2 } 45 } 46 \cs_generate_variant:Nn \NLN_put:nn { nV } 47 \cs_generate_variant:Nn \NLN_gput:nn { nV }</pre> |

```

\NLN_put:nnn \NLN_put:nnn {<name>} {<key>} {<value>}
\NLN_put:nnV \NLN_gput:nnn {<name>} {<key>} {<value>}
\NLN_gput:nnn <name> is a code name.
\NLN_gput:nnV

```

```

48 \cs_new:Npn \NLN_put:nnn #1 #2 #3 {
49   \prop_put:cnn { g/NLN/code/#1 } { #2 } { #3 }
50 }
51 \cs_new:Npn \NLN_gput:nnn #1 #2 #3 {
52   \prop_gput:cnn { g/NLN/code/#1 } { #2 } { #3 }
53 }
54 \cs_generate_variant:Nn \NLN_put:nnn { nnV }
55 \cs_generate_variant:Nn \NLN_gput:nnn { nnV }

```

5.2 Known keys and conditionals

```

\NLN_new_conditional:n \NLN_new_conditional:n {<key>}

```

Create new conditionals for the given key. Does nothing out of this package..

```

56 \cs_new:Npn \NLN_new_conditional:n #1 {
57   \exp_last_unbraced:Nx
58   \prg_new_conditional:Nnn { \use:c {NLN_if_#1:} } { T, F, TF } {
59     \group_begin:
60     \NLN_get:nNTF { #1 } \l_tmpa_tl {
61       \exp_args:NnV
62       \regex_match:nnTF { ^\s*[tTyY] } \l_tmpa_tl
63       { \group_end: \prg_return_true: }
64       { \group_end: \prg_return_false: }
65     } { \group_end: \prg_return_false: }
66   }
67 }

```

format/code Font/size/color specifier for inline code.

```

68 \NLN_gput:nn { format/code } {
69   \ttfamily
70 }

```

format/name Font/size/color specifier for chunk name.

```

71 \NLN_gput:nn { format/name } {
72   \sffamily
73   \scriptsize
74   \color{gray}
75 }

```

format/lineno Font/size/color specifier for line numbers.

```

76 \NLN_gput:nn { format/lineno } {
77   \sffamily
78   \tiny
79   \color{gray}
80 }

```

lang the language, defaults to `tex`

```
81 \NLN_gput:nn { lang } { tex }
```

lineno show line numbers, defaults to `true`

```
82 \NLN_gput:nn { show_lineno } { T }
```

`\NLN_if_show_lineno:TF` `\NLN_if_show_lineno:TF {<true code>} {<false code>}`

Execute *<true code>* when code property `show_lineno` is truthy, *<false code>* otherwise.

```
83 \NLN_new_conditional:n { show_lineno }
```

name show chunk names, defaults to `true`

```
84 \NLN_gput:nn { show_name } { T }
```

`\NLN_if_show_name:TF` `\NLN_if_show_name:TF {<true code>} {<false code>}`

Execute *<true code>* when code property `show_name` is truthy, *<false code>* otherwise.

```
85 \NLN_new_conditional:n { show_name }
```

only top show names only on top, defaults to `true`

```
86 \NLN_gput:nn { only_top } { T }
```

`\NLN_if_only_top:TF` `\NLN_if_only_top:TF {<true code>} {<false code>}`

Execute *<true code>* when code property `only_top` is truthy, *<false code>* otherwise.

```
87 \NLN_new_conditional:n { only_top }
```

margin use the margin to display line numbers and chunk names, defaults to `true`

```
88 \NLN_gput:nn { use_margin } { T }
```

`\NLN_if_use_margin:TF` `\NLN_if_use_margin:TF {<true code>} {<false code>}`

Execute *<true code>* when code property `use_margin` is truthy, *<false code>* otherwise.

```
89 \NLN_new_conditional:n { use_margin }
```

ignore ignore that chunk or that export, defaults to `false`

```
90 \NLN_gput:nn { ignore } { F }
```

`\NLN_if_ignore:TF` `\NLN_if_ignore:TF {⟨true code⟩} {⟨false code⟩}`
Execute *⟨true code⟩* when code property **ignore** is truthy, *⟨false code⟩* otherwise.

91 `\NLN_new_conditional:n { ignore }`

reset reset line numbering, defaults to **false**

92 `\NLN_gput:nn { reset } { F }`

`\NLN_if_reset:TF` `\NLN_if_reset:TF {⟨true code⟩} {⟨false code⟩}`
Execute *⟨true code⟩* when code property **reset** is truthy, *⟨false code⟩* otherwise.

93 `\NLN_new_conditional:n { reset }`

export whether the code should be exported, defaults to **true**

94 `\NLN_gput:nn { export } { T }`

`\NLN_if_export:TF` `\NLN_if_export:TF {⟨true code⟩} {⟨false code⟩}`
Execute *⟨true code⟩* when code property **export** is truthy, *⟨false code⟩* otherwise.

95 `\NLN_new_conditional:n { export }`

parskip the parskip used to separate lines of code

96 `\AddToHook { begindocument/end } {`
97 `\prop_if_in:cnF { g/NLN/code } { parskip } {`
98 `\exp_args:Nnx`
99 `\NLN_gput:nn { parskip } { \the\parskip }`
100 `}`
101 `}`

sep the separation between inline code blocks and surrounding text.

102 `\NLN_gput:nn { sep } { 4pt plus 2pt minus 2pt }`

code the cumulated inline code

103 `\NLN_gput:nn { .code } {}`

Clean memory.

104 `\cs_set_eq:NN \NLN_new_conditional:n \prg_do_nothing:`

6 Counters

\NLN_int_new:nn \NLN_int_new:n {\langle name \rangle} {\langle value \rangle}

Create an integer after $\langle name \rangle$ and set it globally to $\langle value \rangle$. $\langle name \rangle$ is a code name.

```

105 \cs_new:Npn \NLN_int_new:nn #1 #2 {
106   \int_new:c { g/NLN/int/#1 }
107   \int_gset:cn { g/NLN/int/#1 } { #2 }
108 }
```

\NLN_int_set:nn \NLN_int_set:n {\langle name \rangle} {\langle value \rangle}

\NLN_int_gset:nn

Set the integer named after $\langle name \rangle$ to the $\langle value \rangle$. `\NLN_int_gset:n` makes a global change. $\langle name \rangle$ is a code name.

```

109 \cs_new:Npn \NLN_int_set:nn #1 #2 {
110   \int_set:cn { g/NLN/int/#1 } { #2 }
111 }
112 \cs_new:Npn \NLN_int_gset:nn #1 #2 {
113   \int_gset:cn { g/NLN/int/#1 } { #2 }
114 }
```

\NLN_int_add:nn \NLN_int_add:n {\langle name \rangle} {\langle value \rangle}

\NLN_int_gadd:nn

Add the $\langle value \rangle$ to the integer named after $\langle name \rangle$. `\NLN_int_gadd:n` makes a global change. $\langle name \rangle$ is a code name.

```

115 \cs_new:Npn \NLN_int_add:nn #1 #2 {
116   \int_add:cn { g/NLN/int/#1 } { #2 }
117 }
118 \cs_new:Npn \NLN_int_gadd:nn #1 #2 {
119   \int_gadd:cn { g/NLN/int/#1 } { #2 }
120 }
```

\NLN_int_sub:nn \NLN_int_sub:n {\langle name \rangle} {\langle value \rangle}

\NLN_int_gsub:nn

Subtract the $\langle value \rangle$ from the integer named after $\langle name \rangle$. `\NLN_int_gsub:n` makes a global change. $\langle name \rangle$ is a code name.

```

121 \cs_new:Npn \NLN_int_sub:nn #1 #2 {
122   \int_sub:cn { g/NLN/int/#1 } { #2 }
123 }
124 \cs_new:Npn \NLN_int_gsub:nn #1 #2 {
125   \int_gsub:cn { g/NLN/int/#1 } { #2 }
126 }
```

\NLN_int_if_exist:nTF \NLN_int_if_exist:nTF {\langle name \rangle} {\langle true code \rangle} {\langle false code \rangle}

Execute $\langle true code \rangle$ when an integer named after $\langle name \rangle$ exist, $\langle false code \rangle$ otherwise.

```

127 \prg_new_conditional:Nnn \NLN_int_if_exist:n { T, F, TF } {
128   \int_if_exist:cTF { g/NLN/int/#1 } {
129     \prg_return_true:
130   } {
131     \prg_return_false:
132   }
133 }
```

`\g/NLN/int/` Generic and named line number counter. `\l_NLN_code_name_t` is used as $\langle name \rangle$.
`\g/NLN/int/<name>` 134 `\NLN_int_new:nn {} { 1 }`
(End definition for `\g/NLN/int/` and `\g/NLN/int/<name>`. These variables are documented on page ??.)

`\NLN_int_use:n *` `\NLN_int_use:n { $\langle name \rangle$ }`
 $\langle name \rangle$ is a code name.
135 `\cs_new:Npn \NLN_int_use:n #1 {`
136 `\int_use:c { g/NLN/int/#1 }`
137 `}`

7 Variables

Line number counter for the code chunks.

`\g_NLN_code_int` Chunk number counter.
138 `\int_new:N \g_NLN_code_int`
(End definition for `\g_NLN_code_int`. This variable is documented on page ??.)

`\g_NLN_code_prop` Global code property list.
139 `\prop_new:N \g_NLN_code_prop`
(End definition for `\g_NLN_code_prop`. This variable is documented on page ??.)

`\g_NLN_export_prop` Global storage for $\langle file name \rangle = \langle comma separated chunk name \rangle$
140 `\prop_new:N \g_NLN_export_prop`
(End definition for `\g_NLN_export_prop`. This variable is documented on page ??.)

`\l_NLN_prop` Local scratch variable.
141 `\prop_new:N \l_NLN_prop`
(End definition for `\l_NLN_prop`. This variable is documented on page ??.)

`\g_NLN_chunks_tl` The comma separated list of current chunks. If the next list of chunks is the same as the
`\l_NLN_chunks_tl` current one, then it might not display.
142 `\tl_new:N \g_NLN_chunks_tl`
143 `\tl_new:N \l_NLN_chunks_tl`
(End definition for `\g_NLN_chunks_tl` and `\l_NLN_chunks_tl`. These variables are documented on page ??.)

`\g_NLN_vars` Tree storage for global variables.
144 `\prop_new:N \g_NLN_vars`

WHAT

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

`\g_NLN_vars` Tree storage for global variables.
145 `\tl_new:N \g_NLN_hook_tl`
(End definition for `\g_NLN_vars`. This variable is documented on page ??.)

`\g/NLN/Chunks/<name>` List of chunk keys for given named code.
(End definition for `\g/NLN/Chunks/<name>`. This variable is documented on page ??.)

7.1 Local variables

| | |
|---------------------------------|--|
| <code>\l_NLN_recorded_tl</code> | Full verbatim body of the <code>Inline</code> environment. ¹⁴⁶ <code>\tl_new:N \l_NLN_recorded_tl</code> <i>(End definition for <code>\l_NLN_recorded_tl</code>. This variable is documented on page ??.)</i> |
| <code>\g_NLN_int</code> | Global integer to store linenos locally in time. ¹⁴⁷ <code>\int_new:N \g_NLN_int</code> <i>(End definition for <code>\g_NLN_int</code>. This variable is documented on page ??.)</i> |
| <code>\l_NLN_line_tl</code> | Token list for one line. ¹⁴⁸ <code>\tl_new:N \l_NLN_line_tl</code> <i>(End definition for <code>\l_NLN_line_tl</code>. This variable is documented on page ??.)</i> |
| <code>\l_NLN_lineno_tl</code> | Token list for lineno display. ¹⁴⁹ <code>\tl_new:N \l_NLN_lineno_tl</code> <i>(End definition for <code>\l_NLN_lineno_tl</code>. This variable is documented on page ??.)</i> |
| <code>\l_NLN_name_tl</code> | Token list for chunk name display. ¹⁵⁰ <code>\tl_new:N \l_NLN_name_tl</code> <i>(End definition for <code>\l_NLN_name_tl</code>. This variable is documented on page ??.)</i> |
| <code>\l_NLN_info_tl</code> | Token list for the info of line. ¹⁵¹ <code>\tl_new:N \l_NLN_info_tl</code> <i>(End definition for <code>\l_NLN_info_tl</code>. This variable is documented on page ??.)</i> |
| <code>\l_NLN_clist</code> | The comma separated list of current chunks. ¹⁵² <code>\clist_new:N \l_NLN_clist</code> <i>(End definition for <code>\l_NLN_clist</code>. This variable is documented on page ??.)</i> |
| <code>\l_NLN_in</code> | Input file identifier ¹⁵³ <code>\ior_new:N \l_NLN_in</code> <i>(End definition for <code>\l_NLN_in</code>. This variable is documented on page ??.)</i> |
| <code>\l_NLN_out</code> | Output file identifier ¹⁵⁴ <code>\iow_new:N \l_NLN_out</code> <i>(End definition for <code>\l_NLN_out</code>. This variable is documented on page ??.)</i> |

8 Utilities

Utilities

`\NLN_clist_map_inline:Nnn`

`\NLN_clist_map_inline:Nnn` \langle *clist var* \rangle
 $\{\langle \rangle\}$ non empty code $\{\langle \rangle\}$ empty code

Call `\clist_map_inline:Nnn` \langle *clist var* \rangle $\{\langle$ *non empty code* $\rangle\}$ when the list is not empty, execute metaempty code otherwise.

```
155 \cs_new:Npn \NLN_clist_map_inline:Nnn #1 #2 #3 {  
156   \clist_if_empty:NTF #1 { #3 } {  
157     \clist_map_inline:Nn #1 { #2 }  
158   }  
159 }
```

`\NLN_process_record:`

Record the current line or not.

```
160 \cs_new:Npn \NLN_process_record: {}
```

9 Shared key-value controls

Each action is meant to store the values in a code property, for the almost eponym key.

```
161 \keys_define:nn { NLN } {
```

Keys are:

lineno $[\text{=true/false}]$ to display the line numbers, or not,

```
162   lineno .code:n = \NLN_put:nn { show_lineno } { #1 },  
163   lineno .default:n = true,
```

name $[\text{=true/false}]$ to display the chunk names

```
164   name .code:n = \NLN_put:nn { show_name } { #1 },  
165   name .default:n = true,
```

only top to avoid chunk names repetitions, if on the same page, two consecutive code chunks have the same chunk names, the second names are not displayed.

```
166   only-top .code:n = \NLN_put:nn { only_top } { #1 },  
167   only-top .default:n = true,
```

ignore to ignore chunks.

```
168   ignore .code:n = \NLN_put:nn { ignore } { #1 },  
169   ignore .default:n = true,
```

margin $[\text{=true/false}]$ to use the margin to display line numbers, or not,

```
170   margin .code:n = \NLN_put:nn { use_margin } { #1 },  
171   margin .default:n = true,
```

lang= \langle language name \rangle , where \langle language name \rangle is recognized by pygment,

```
172      lang .code:n = \NLN_put:nn { lang } { #1 },
```

code format= \langle format \rangle , where \langle format \rangle is used to display the code (mainly font, size and color),

```
173      code~format .code:n = \NLN_put:nn { format/code } { #1 },
```

lineno format= \langle format \rangle , where \langle format \rangle is used to display the line numbers (mainly font, size and color),

```
174      name~format .code:n = \NLN_put:nn { format/name } { #1 },
```

name format= \langle format \rangle , where \langle name format \rangle is used to display the chunk names (mainly font, size and color),

```
175      lineno~format .code:n = \NLN_put:nn { format/lineno } { #1 },
```

post processor the name of the pygment post processor,

```
176      post~processor .code:n = \NLN_put:nn { post_processor } { #1 },
```

post processor args the arguments of the pygment post processor,

```
177      post~processor~args .code:n = \NLN_put:nn { post_processor_args } { #1 },
```

sep the separation with the surrounding text,

```
178      sep .code:n = \NLN_put:nn { sep } { #1 },
```

parskip the value of the \parskip in inline code blocks,

```
179      parskip .code:n = \NLN_put:nn { parskip } { #1 },
```

test whether the chunk is a test,

```
180      test .code:n = \NLN_put:nn { test } { #1 },
```

```
181      unknown .code:n = \PackageWarning
182      { inline }
183      { Unknown~option~'\l_keys_key_str' },
184      }
```

10 \InlineSet

\InlineSet \InlineSet { \langle key[=value] list \rangle }

To set up the package. This is executed at least once at the end of the preamble. The unique mandatory argument of \InlineSet is a list of \langle key \rangle [=value] items defined by

10.1 NLN/set key-value controls.

```
185 \keys_define:nn { } { NLN/set .inherit:n = NLN }
186 \keys_define:nn { NLN/set } {
```

minted to activate syntax coloring with pygment, calls `_NLN_minted_on:` and forwards the argument as `minted` option,

```
187     minted .code:n = {
188         \_NLN_minted_on:
189         \setkeys { minted@opt@g } { #1 }
190     },
```

minted style=<name> to select a predefined minted style, forwarded to `\usemintedstyle`,

```
191     minted~style .code:n = {
192         \RemoveFromHook { begindocument/before } [NLN/Minted]
193         \AddToHook { begindocument/before } [NLN/Minted] {
194             \usemintedstyle { #1 }
195         }
196     },
```

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

```
197     only~description .code:n = \prop_put:Nnn \l_NLN_vars
198     { only_description } { #1 },
```

```
199     unknown .code:n = \PackageWarning
200     { NLN/set }
201     { Unknown~option~'\l_keys_key_str' },
202 }
```

10.2 Implementation

```
203 \NewDocumentCommand \InlineSet { m } {
204     \keys_set:nn { NLN/set } {#1}
205     \NLN_if_use_minted:F {
206         \bool_if:NT \g_NLN_minted_on_bool {
207             \sys_if_shell:TF {
208                 \_NLN_if_pygmentize:TF {
209                     \bool_gset_true:N \g_NLN_use_minted_bool
210                 } {
211                     \msg_warning:nnn
212                     { inline }
213                     { :n }
214                     { No~"pygmentize"~found. }
215                 }
216             } {
217                 \msg_warning:nnn
218                 { inline }
219                 { :n }
220                 { No~unrestricted-shell-escape~for~"pygmentize".}
221             }
222         }
223     }
```

```

222     }
223   }
224 }

```

11 InlineSplit environment

12 Inline environment

`Inline` `\begin{<Inline>}{<key[=value] list>} ... \end{<Inline>}`

The `<key> [= <value>]` items are defined by the

12.1 NLN/code key-value controls

```

225 \keys_define:nn { } { NLN/code .inherit:n = NLN }
226 \keys_define:nn { NLN/code } {

```

`chunks=<comma separated list of chunk names>` When declaring an exported file, this is the list of chunks that will appear in that file. When declaring a code chunk, this the list of chunks where it will be stored. Chunks are collected unordered and ordered for comparison.

```

227     chunks .clist_set:N = \l_NLN_clist,

```

`reset[=<boolean string>` When declaring an exported file, this is the list of chunks that will appear in that file. When declaring a code chunk, this the list of chunks where it will be stored. Chunks are collected unordered and ordered for comparison.

```

228     reset .code:n = \NLN_put:nn { reset } { #1 },
229     reset .default:n = true,

```

```

230     unknown .code:n = \PackageWarning
231     { NLN/code }
232     { Unknown~option~'\l_keys_key_str' },
233 }

```

12.2 Implementation

`\NLN_if_record:TF` `\NLN_if_record:TF {<true code>} {<false code>}`

Execute `<true code>` when code should be recorded, `<false code>` otherwise.

```

234 \prg_new_conditional:Nnn \NLN_if_record: { T, F, TF } {
235   \NLN_if_export:TF {
236     \prg_return_true:
237   } {
238     \NLN_if_use_minted:TF {
239       \prg_return_true:
240     } {
241       \prg_return_false:
242     }
243   }
244 }

```

```

245 \cs_set:Npn \NLN_process_record: {
246   \tl_put_right:Nx \l_NLN_recorded_tl { \the\verbatim@line \iow_newline: }
247   \group_begin:
248   \tl_set:Nx \l_tmpa_tl { \the\verbatim@line }
249   \exp_args:Nx \directlua {NLN.records.append([===[\l_tmpa_tl]===])}
250   \group_end:
251 }

252 \DeclareDocumentEnvironment { Inline } { m } {
253   \directlua{NLN:start_recording()}
254   \clist_clear:N \l_NLN_clist
255   \keys_set:nn { NLN/code } { #1 }
256   \clist_map_inline:Nn \l_NLN_clist {
257     \NLN_int_if_exist:nF { ##1 } {
258       \NLN_int_new:nn { ##1 } { 1 }
259       \seq_new:c { g/NLN/chunks/##1 }
260     }
261   }
262   \NLN_if_reset:T {
263     \NLN_clist_map_inline:Nnn \l_NLN_clist {
264       \NLN_int_gset:nn { ##1 } 1
265     } {
266       \NLN_int_gset:nn { } 1
267     }
268   }
269   \tl_clear:N \l_NLN_code_name_tl
270   \clist_map_inline:Nn \l_NLN_clist {
271     \prop_concat:ccc
272     {g/NLN/code/}
273     { g/NLN/code/##1 }
274     {g/NLN/code/}
275     \tl_set:Nn \l_NLN_code_name_tl { ##1 }
276     \clist_map_break:
277   }
278   \int_gset:Nn \g_NLN_int
279   { \NLN_int_use:n { \l_NLN_code_name_tl } }
280   \tl_clear:N \l_NLN_info_tl
281   \tl_clear:N \l_NLN_name_tl
282   \tl_clear:N \l_NLN_recorded_tl
283   \tl_clear:N \l_NLN_chunks_tl
284   \cs_set:Npn \verbatim@processline {
285     \NLN_process_record:
286   }
287   \NLN_if_show_code:TF {
288     \exp_args:NNx
289     \skip_set:Nn \parskip { \NLN_item:n { parskip } }
290     \clist_if_empty:NTF \l_NLN_clist {
291       \tl_gclear:N \g_NLN_chunks_tl
292     } {
293       \clist_set_eq:NN \l_tmpa_clist \l_NLN_clist
294       \clist_sort:Nn \l_tmpa_clist {
295         \str_compare:nNnTF { ##1 } > { ##2 } {
296           \sort_return_swapped:
297         } {
298           \sort_return_same:

```

```

299     }
300   }
301   \tl_set:Nx \l_tmpa_tl { \clist_use:Nn \l_tmpa_clist , }
302   \NLN_if_show_name:T {
303     \NLN_if_use_margin:T {
304       \NLN_if_only_top:T {
305         \tl_if_eq:NNT \l_tmpa_tl \g_NLN_chunks_tl {
306           \tl_gset_eq:NN \g_NLN_chunks_tl \l_tmpa_tl
307           \tl_clear:N \l_tmpa_tl
308         }
309       }
310       \tl_if_empty:NF \l_tmpa_tl {
311         \tl_set:Nx \l_NLN_chunks_tl {
312           \clist_use:Nn \l_NLN_clist ,
313         }
314         \tl_set:Nn \l_NLN_name_tl {
315           {
316             \NLN_item:n { format/name }
317             \l_NLN_chunks_tl :
318             \hspace*{1ex}
319           }
320         }
321       }
322     }
323     \tl_if_empty:NF \l_tmpa_tl {
324       \tl_gset_eq:NN \g_NLN_chunks_tl \l_tmpa_tl
325     }
326   }
327 }
328 \if_mode_vertical:
329 \else:
330 \par
331 \fi:
332 \vspace{ \NLN_item:n { sep } }
333 \noindent
334 \frenchspacing
335 \@vobeyspaces
336 \normalfont\ttfamily
337 \NLN_item:n { format/code }
338 \hyphenchar\font\m@ne
339 \@noligs
340 \NLN_if_record:F {
341   \cs_set_eq:NN \NLN_process_record: \prg_do_nothing:
342 }
343 \NLN_if_use_minted:F {
344   \NLN_if_show_lineno:T {
345     \NLN_if_use_margin:TF {
346       \tl_set:Nn \l_NLN_info_tl {
347         \hbox_overlap_left:n {
348           {
349             \l_NLN_name_tl
350             \NLN_item:n { format/name }
351             \NLN_item:n { format/lineno }
352             \int_use:N \g_NLN_int

```

```

353         \int_gincr:N \g_NLN_int
354     }
355     \hspace*{1ex}
356 }
357 }
358 } {
359     \tl_set:Nn \l_NLN_info_tl {
360     {
361         \NLN_item:n { format/name }
362         \NLN_item:n { format/lineno }
363         \hspace*{3ex}
364         \hbox_overlap_left:n {
365             \int_use:N \g_NLN_int
366             \int_gincr:N \g_NLN_int
367         }
368     }
369     \hspace*{1ex}
370 }
371 }
372 }
373 \cs_set:Npn \verbatim@processline {
374     \NLN_process_record:
375     \hspace*{\dimexpr \linewidth-\columnwidth}%
376     \hbox_to_wd:nn { \columnwidth } {
377         \l_NLN_info_tl
378         \the\verbatim@line
379         \color{lightgray}\dotfill
380     }
381     \tl_clear:N \l_NLN_name_tl
382     \par\noindent
383 }
384 }
385 } {
386     \@bsphack
387 }
388 \group_begin:
389 \g_NLN_hook_tl
390 \let \do \@makeother
391 \dospecials \catcode '\^M \active
392 \verbatim@start
393 } {
394     \int_gsub:Nn \g_NLN_int {
395         \NLN_int_use:n { \l_NLN_code_name_tl }
396     }
397     \int_compare:nNnT { \g_NLN_int } > { 0 } {
398         \NLN_clist_map_inline:Nnn \l_NLN_clist {
399             \NLN_int_gadd:nn { ##1 } { \g_NLN_int }
400         } {
401             \NLN_int_gadd:nn { } { \g_NLN_int }
402         }
403         \int_gincr:N \g_NLN_code_int
404         \tl_set:Nx \l_tmpb_tl { \int_use:N \g_NLN_code_int }
405         \clist_map_inline:Nn \l_NLN_clist {
406             \seq_gput_right:cV { g/NLN/chunks/##1 } \l_tmpb_tl

```



```

407 }
408 \prop_gput:NVV \g_NLN_code_prop \l_tmpb_tl \l_NLN_recorded_tl
409 }
410 \group_end:
411 \NLN_if_show_code:T {
412 }
413 \NLN_if_show_code:TF {
414 \NLN_if_use_minted:TF {
415 \tl_if_empty:NF \l_NLN_recorded_tl {
416 \exp_args:Nnx \setkeys { FV } {
417 firstnumber=\NLN_int_use:n { \l_NLN_code_name_tl },
418 }
419 \iow_open:Nn \minted@code { \jobname.pyg }
420 \exp_args:NNV \iow_now:Nn \minted@code \l_NLN_recorded_tl
421 \iow_close:N \minted@code
422 \vspace* { \dimexpr -\topsep-\parskip }
423 \tl_if_empty:NF \l_NLN_info_tl {
424 \tl_use:N \l_NLN_info_tl
425 \skip_vertical:n { \dimexpr -\topsep-\parskip-\baselineskip }
426 \par\noindent
427 }
428 \exp_args:Nnx \minted@pygmentize { \jobname.pyg } { \NLN_item:n { lang } }
429 %\DeleteFile { \jobname.pyg }
430 \skip_vertical:n { -\topsep-\partopsep }
431 }
432 } {
433 \exp_args:Nx \skip_vertical:n { \NLN_item:n { sep } }
434 \noindent
435 }
436 } {
437 \@esphack
438 }
439 }

```

NLN \begin{<NLN>} ... \end{<NLN>}
Private environment.

```

440 \newenvironment{NLN}{
441 \def \verbatim@processline {
442 \group_begin:
443 \NLN_processline_code_append:
444 \group_end:
445 }
446 % \NLN_if_show_code:T {
447 % \NLN_if_use_minted:TF {
448 % \Needspace* { 2\baselineskip }
449 % } {
450 % \frenchspacing\@vobeyspaces
451 % }
452 % }
453 } {
454 \NLN_get:nNTF { lang } \l_tmpa_tl {
455 \tl_if_empty:NT \l_tmpa_tl {
456 \clist_map_inline:Nn \l_NLN_clist {

```

```

457     \NLN_get:nnNT { ##1 } { lang } \l_tmpa_tl {
458         \tl_if_empty:NF \l_tmpa_tl {
459             \clist_map_break:
460         }
461     }
462 }
463 \tl_if_empty:NT \l_tmpa_tl {
464     \tl_set:Nn \l_tmpa_tl { tex }
465 }
466 }
467 } {
468     \tl_set:Nn \l_tmpa_tl { tex }
469 }
470 \clist_map_inline:Nn \l_NLN_clist {
471     \NLN_gput:nnV { ##1 } { lang } \l_tmpa_tl
472 }
473 }

```

NLN.M \begin{<NLN.M> ... \end{<NLN.N>}
Private environment when minted.

```

474 \newenvironment{NLN_M}{
475     \setkeys { FV } { firstnumber=last, }
476     \clist_if_empty:NTF \l_NLN_clist {
477         \exp_args:Nnx \setkeys { FV } {
478             firstnumber=\NLN_int_use:n { },
479         } } {
480         \clist_map_inline:Nn \l_NLN_clist {
481             \exp_args:Nnx \setkeys { FV } {
482                 firstnumber=\NLN_int_use:n { ##1 },
483             }
484         \clist_map_break:
485     } }
486     \iow_open:Nn \minted@code { \jobname.pyg }
487     \tl_set:Nn \l_NLN_line_tl {
488         \tl_set:Nx \l_tmpa_tl { \the\verbatim@line }
489         \exp_args:NNV \iow_now:Nn \minted@code \l_tmpa_tl
490     }
491 } {
492     \NLN_if_show_code:T {
493         \NLN_if_use_minted:TF {
494             \iow_close:N \minted@code
495             \vspace* { \dimexpr -\topsep-\parskip }
496             \tl_if_empty:NF \l_NLN_info_tl {
497                 \tl_use:N \l_NLN_info_tl
498                 \vspace* { \dimexpr -\topsep-\parskip-\baselineskip }
499                 \par\noindent
500             }
501             \exp_args:NV \minted@pygmentize \l_tmpa_tl
502             \DeleteFile { \jobname.pyg }
503             \vspace* { \dimexpr -\topsep -\partopsep }
504         } {
505             \@esphack
506         }

```

```

507 }
508 }

```

NLN.P `\begin{<NLN.P>} ... \end{<NLN.P>}`
Private pseudo environment. This is just a practical way of declaring balanced actions.

```

509 \newenvironment{NLN_P}{
510   \if_mode_vertical:
511     \noindent
512   \else
513     \vspace*{ \topsep }
514     \par\noindent
515   \fi
516   \NLN_gset_chunks:
517   \tl_if_empty:NTF \g_NLN_chunks_tl {
518     \NLN_if_show_lineno:TF {
519       \NLN_if_use_margin:TF {

```

No chunk name, line numbers in the margin

```

520       \tl_set:Nn \l_NLN_info_tl {
521         \hbox_overlap_left:n {
522           \NLN_item:n { format/code }
523           {
524             \NLN_item:n { format/name }
525             \NLN_item:n { format/lineno }
526             \clist_if_empty:NTF \l_NLN_clist {
527               \NLN_int_use:n { }
528             } {
529               \clist_map_inline:Nn \l_NLN_clist {
530                 \NLN_int_use:n { ##1 }
531               }
532             }
533           }
534         }
535         \hspace*{1ex}
536       }
537     }
538   } {

```

No chunk name, line numbers not in the margin

```

539       \tl_set:Nn \l_NLN_info_tl {
540         {
541           \NLN_item:n { format/code }
542           {
543             \NLN_item:n { format/name }
544             \NLN_item:n { format/lineno }
545             \hspace*{3ex}
546             \hbox_overlap_left:n {
547               \clist_if_empty:NTF \l_NLN_clist {
548                 \NLN_int_use:n { }
549               } {
550                 \clist_map_inline:Nn \l_NLN_clist {
551                   \NLN_int_use:n { ##1 }
552                 }

```

```

553         }
554     }
555 }
556 \hspace*{1ex}
557 }
558 }
559 }
560 }
561 } {

```

No chunk name, no line numbers

```

562 \tl_clear:N \l_NLN_info_tl
563 }
564 } {
565 \NLN_if_show_lineno:TF {

```

Chunk names, line numbers, in the margin

```

566 \tl_set:Nn \l_NLN_info_tl {
567   \hbox_overlap_left:n {
568     \NLN_item:n { format/code }
569     {
570       \NLN_item:n { format/name }
571       \g_NLN_chunks_tl :
572       \hspace*{1ex}
573       \NLN_item:n { format/lineno }
574       \clist_map_inline:Nn \l_NLN_clist {
575         \NLN_int_use:n { ###1 }
576         \clist_map_break:
577       }
578     }
579     \hspace*{1ex}
580   }
581   \tl_set:Nn \l_NLN_info_tl {
582     \hbox_overlap_left:n {
583       \NLN_item:n { format/code }
584       {
585         \NLN_item:n { format/name }
586         \NLN_item:n { format/lineno }
587         \clist_map_inline:Nn \l_NLN_clist {
588           \NLN_int_use:n { ###1 }
589           \clist_map_break:
590         }
591       }
592       \hspace*{1ex}
593     }
594   }
595 }
596 } {

```

Chunk names, no line numbers, in the margin

```

597 \tl_set:Nn \l_NLN_info_tl {
598   \hbox_overlap_left:n {
599     \NLN_item:n { format/code }
600     {
601       \NLN_item:n { format/name }

```

```

602         \g_NLN_chunks_tl :
603     }
604     \hspace*{1ex}
605 }
606 \tl_clear:N \l_NLN_info_tl
607 }
608 }
609 }
610 \NLN_if_use_minted:F {
611     \tl_set:Nn \l_NLN_line_tl {
612         \noindent
613         \hbox_to_wd:nn { \textwidth } {
614             \tl_use:N \l_NLN_info_tl
615             \NLN_item:n { format/code }
616             \the\verbatim@line
617             \hfill
618         }
619         \par
620     }
621     \@bsphack
622 }
623 } {
624     \vspace*{ \topsep }
625     \par
626     \@esphack
627 }

```

13 \InlineExport

\InlineExport `\InlineExport{<key[=value] list}`

The `<key>[=<value>]` items are defined by

13.1 NLN/export key-value controls

```

628 \keys_define:nn { } { NLN/export .inherit:n = NLN/code }
629 \keys_define:nn { NLN/export } {

```

file the output file name

```

630     file .tl_set:N = \l_NLN_tl,
631     file .value_required:n = true,

```

preamble the added preamble.

```

632     preamble .code:n = \prop_put:Nnn \l_NLN_vars { preamble } { #1 },

```

raw to remove any additional material,

```

633     raw .code:n = \prop_put:Nnn \l_NLN_vars { raw } { #1 },

```

```

634     unknown .code:n = \PackageWarning
635         { NLN/export }
636         { Unknown~option~'\l_keys_key_str' },
637 }

```

13.2 Implementation

```

638 \DeclareDocumentCommand \InlineExport { m } {
639   \group_begin:
640   \clist_clear:N \l_NLN_clist
641   \prop_clear:c {g/NLN/code/}
642   \prop_put:cnn {g/NLN/code/} { lang } { tex }
643   \keys_set:nn { NLN/export } { #1 }
644   \prop_gput:NVV \g_NLN_export_prop \l_NLN_tl \l_NLN_clist
645   \prop_gput:cnV { g/NLN/export/\l_NLN_tl } { chunks } \l_NLN_clist
646   \prop_gput:cnx { g/NLN/export/\l_NLN_tl } { preamble }
647     { \prop_item:Nn \l_NLN_vars { preamble } }
648   \bool_set:Nx \l_tmpa_bool { \prop_item:Nn \l_NLN_vars { raw } }
649   \prop_gput:cnV { g/NLN/export/\l_NLN_tl } { preamble } \l_tmpa_bool
650   \NLN_get:nNT { lang } \l_tmpa_tl {
651     \clist_map_inline:Nn \l_NLN_clist {
652       \prop_gconcat:ccc
653         { g/NLN/code/##1 }
654         { g/NLN/code/##1 }
655         {g/NLN/code/}
656     }
657   }
658   \group_end:
659 }

```

Files are created at the end of the typesetting process.

```

660 \AddToHook { enddocument / end } {
661   \group_begin:
662   \prop_map_inline:Nn \g_NLN_export_prop {
663     \iow_open:Nn \l_NLN_out { #1 }
664     \iow_term:x { Exporting~chunks~#2~to~#1 }
665     \prop_get:cnNF { g/NLN/export/#1 } { raw } \l_tmpa_bool {
666       \bool_set_false:N \l_tmpa_bool
667     }
668     \bool_if:NF \l_tmpa_bool {
669       \prop_get:cnNT { g/NLN/export/#1 } { preamble } \l_tmpa_tl {
670         \prop_get:cnNF { g/NLN/export/#1 } { lang } \l_tmpa_str {
671           \str_set:Nn \l_tmpa_str { tex }
672         }
673         \prop_get:NVNTF \c_NLN_comment_prop \l_tmpa_str \l_tmpa_str {
674           \tl_set:Nn \l_tmpb_tl {
675             \l_tmpa_str\l_tmpa_str\space\space
676           }
677         } {
678           \tl_clear:N \l_tmpb_tl
679         }
680         \tl_put_right:Nx \l_tmpb_tl {
681           This~is~file~‘#1’~
682           generated~from~‘\c_sys_jobname_str.tex’~on~\DTMnow.
683         }
684         \iow_now:Nx \l_NLN_out { \l_tmpb_tl }
685         \iow_now:Nx \l_NLN_out { \l_tmpa_tl }
686       }
687     }
688     \clist_map_inline:nn { #2 } {

```

```

689     \NLN_get:nnNT { ##1 } { .code } \l_tmpa_tl {
690         \exp_args:NNV \iow_now:Nn \l_NLN_out \l_tmpa_tl
691     }
692 }
693 \iow_close:N \l_NLN_out
694 }
695 \group_end:
696 }

```

14 Management

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

```

697 \bool_new:N \g_NLN_in_impl_bool

```

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

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

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

```

698 \prg_new_conditional:Nnn \NLN_if_show_code: { T, F, TF } {
699     \bool_if:nTF {
700         \g_NLN_in_impl_bool && !\g_NLN_with_impl_bool
701     } {
702         \prg_return_false:
703     } {
704         \prg_return_true:
705     }
706 }

```

`\g_NLN_with_impl_bool`

```

707 \bool_new:N \g_NLN_with_impl_bool

```

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

15 All purpose messaging

16 minted and pygment

`\g_NLN_minted_on_bool` Whether minted is available, initially set to `false`.

```

708 \bool_new:N \g_NLN_minted_on_bool

```

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

`\g_NLN_use_minted_bool` Whether minted is used, initially set to `false`.

```

709 \bool_new:N \g_NLN_use_minted_bool

```

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

`\NLN_if_use_minted:TF` `\NLN_if_use_minted:TF` `{\true code}` `{\false code}`

Execute `\true code` when using minted, `\false code` otherwise.

```

710 \prg_new_conditional:Nnn \NLN_if_use_minted: { T, F, TF } {
711   \bool_if:NTF \g_NLN_use_minted_bool
712     { \prg_return_true: }
713     { \prg_return_false: }
714 }
```

`_NLN_if_pygmentize:TF` `\NLN_if_pygmentize:TF` `{\true code}` `{\false code}`

Execute `\true code` when pygmentize is available, `\false code` otherwise.

```

715 \prg_new_conditional:Nnn\_NLN_if_pygmentize: { T, F, TF } {
716   \group_begin:
717   \sys_get_shell:nnN {which-pygmentize} {} \l_tmpa_tl
718   \tl_if_empty:NTF \l_tmpa_tl {
719     \tl_set:Nn \l_tmpa_tl { \prg_return_false: }
720   } {
721     \tl_set:Nn \l_tmpa_tl { \prg_return_true: }
722   }
723   \exp_last_unbraced:NV
724   \group_end: \l_tmpa_tl
725 }
```

`_NLN_minted_on:` `_NLN_minted_on:`

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

```

726 \cs_set:Npn \_NLN_minted_on: {
727   \directlua{NLN.make_directory("_pygmented")}
728   \bool_gset_true:N \g_NLN_minted_on_bool
729   \RequirePackage{minted}
730   \setkeys{ minted@opt@g } { linenos=false }
731   \minted@def@opt{post~processor}
732   \minted@def@opt{post~processor~args}
733   \pretocmd\minted@inputpyg{
734     \NLN@postprocesspyg {\minted@outputdir\minted@infile}
735   }{\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

```

736 \newcommand{\NLN@postprocesspyg}[1]{%
737   \group_begin:
738   \tl_set:Nx \l_tmpa_tl {\NLN_item:n { post_processor } }
739   \tl_if_empty:NF \l_tmpa_tl {
```



```

Execute ‘python3 <script.py> <file.pygtex> <more_args>’
740     \tl_set:Nx \l_tmpb_tl {\NLN_item:n { post_processor_args } }
741     \exp_args:Nx
742     \sys_shell_now:n {
743         python3\space
744         \l_tmpa_tl\space
745         ##1\space
746         \l_tmpb_tl
747     }
748 }
749 \group_end:
750 }
751 }

752 %\AddToHook { begindocument / end } {
753 % \cs_set_eq:NN \_NLN_minted_on: \prg_do_nothing:
754 %}

```

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

```

755 \ProvideDocumentCommand{\InlineEmph}{m}{\textcolor{red}{#1}}

```

| | |
|-----------------------------------|--|
| <code>\InlineStorePreamble</code> | <code>\InlineStorePreamble {<variable>} {<file name>}</code> |
|-----------------------------------|--|

Store the content of `<file name>` into the variable `<variable>`.

17 Separators

| | |
|------------------------------------|------------------------------------|
| <code>\InlineImplementation</code> | <code>\InlineImplementation</code> |
|------------------------------------|------------------------------------|

Start an implementation part where all the sectioning commands do nothing.

| | |
|----------------------------|----------------------------|
| <code>\InlineFinale</code> | <code>\InlineFinale</code> |
|----------------------------|----------------------------|

Stop an implementation part.

18 Finale

```

756 \DeclareDocumentCommand \InlineStorePreamble { m m } {
757     \group_begin:
758     \msg_info:nnn
759     { inline }
760     { :n }
761     { Reading~preamble~from~file~"#2". }
762     \tl_clear:N \g_tmpa_tl
763     \tl_clear:N \g_tmpb_tl
764     \ior_open:Nn \l_NLN_in { #2 }
765     \bool_until_do:nn { \ior_if_eof_p:N \l_NLN_in } {
766         \ior_str_get:NN \l_NLN_in \l_tmpa_tl
767         \tl_if_empty:NTF \l_tmpa_tl {
768             \tl_put_right:Nn \g_tmpb_tl { \iow_newline: }
769         } {

```

```

770     \tl_put_right:Nx \g_tmpa_tl { \g_tmpb_tl }
771     \tl_set:Nn \g_tmpb_tl { \iow_newline: }
772     \tl_put_right:NV \g_tmpa_tl \l_tmpa_tl
773   }
774 }
775 \ior_close:N \l_NLN_in
776 \exp_args:NNNx
777 \group_end:
778 \tl_set:Nn #1 { \tl_to_str:N \g_tmpa_tl }
779 }
780 \newcounter{NLN@impl@page}
781 \DeclareDocumentCommand \InlineImplementation {} {
782   \bool_if:NF \g_NLN_with_impl_bool {
783     \clearpage
784     \bool_gset_true:N \g_NLN_in_impl_bool
785     \let\NLN@old@part\part
786     \DeclareDocumentCommand\part{som}{ }
787     \let\NLN@old@section\section
788     \DeclareDocumentCommand\section{som}{ }
789     \let\NLN@old@subsection\subsection
790     \DeclareDocumentCommand\subsection{som}{ }
791     \let\NLN@old@subsubsection\subsubsection
792     \DeclareDocumentCommand\subsubsection{som}{ }
793     \let\NLN@old@paragraph\paragraph
794     \DeclareDocumentCommand\paragraph{som}{ }
795     \let\NLN@old@subparagraph\subparagraph
796     \DeclareDocumentCommand\subparagraph{som}{ }
797     \cs_if_exist:NT \refsection{ \refsection }
798     \setcounter{ NLN@impl@page }{ \value{page} }
799   }
800 }
801 \DeclareDocumentCommand \InlineFinale {} {
802   \bool_if:NF \g_NLN_with_impl_bool {
803     \clearpage
804     \bool_gset_false:N \g_NLN_in_impl_bool
805     \let\part\NLN@old@part
806     \let\section\NLN@old@section
807     \let\subsection\NLN@old@subsection
808     \let\subsubsection\NLN@old@subsubsection
809     \let\paragraph\NLN@old@paragraph
810     \let\subparagraph\NLN@old@subparagraph
811     \setcounter { page } { \value{ NLN@impl@page } }
812   }
813 }
814 \cs_set_eq:NN \NLN_line_number: \prg_do_nothing:

```

19 Finale

```

815 \AddToHook { cmd/FancyVerbFormatLine/before } {
816   \NLN_line_number:
817 }
818 \AddToHook { shipout/before } {
819   \tl_gclear:N \g_NLN_chunks_tl
820 }

```

```

821 \InlineSet {}
822 % =====
823 % Auxiliary:
824 %   finding the widest string in a comma
825 %   separated list of strings delimited by parenthesis
826 % =====
827
828 % arguments:
829 % #1) text: a comma separated list of strings
830 % #2) formatter: a macro to format each string
831 % #3) dimension: will hold the result
832
833 \cs_new:Npn \NLNWidest (#1) #2 #3 {
834   \group_begin:
835   \dim_set:Nn #3 { 0pt }
836   \clist_map_inline:nn { #1 } {
837     \hbox_set:Nn \l_tmpa_box { #2{##1} }
838     \dim_set:Nn \l_tmpa_dim { \dim_eval:n { \box_wd:N \l_tmpa_box } }
839     \dim_compare:nNnT { #3 } < { \l_tmpa_dim } {
840       \dim_set_eq:NN #3 \l_tmpa_dim
841     }
842   }
843   \exp_args:NNNV
844   \group_end:
845   \dim_set_eq:NN #3 #3
846 }
847
848 \ExplSyntaxOff
849

```

20 pygmentex implementation

```

850 % =====
851 % fancyvrb new commands to append to a file
852 % =====
853
854 % See http://tex.stackexchange.com/questions/47462/inputenc-error-with-unicode-chars-and-verbatim
855
856 \ExplSyntaxOn
857
858 \seq_new:N \l_NLN_records_seq
859
860 \long\def\unexpanded@write#1#2{\write#1{\unexpanded{#2}}}
861
862 \def\VerbatimOutAppend{\FV@Environment{}{\VerbatimOutAppend}}
863
864 \def\FVB@VerbatimOutAppend#1{%
865   \@bsphack
866   \begingroup
867     \seq_clear:N \l_NLN_records_seq
868     \FV@UseKeyValues
869     \FV@DefineWhiteSpace
870     \def\FV@Space{\space}%

```

```

871 \FV@DefineTabOut
872 \def\FV@ProcessLine{##1
873 % \seq_put_right:Nn \l_NLN_records_seq { ##1 }%
874 \immediate\unexpanded@write#1{##1}
875 }%
876 \let\FV@FontScanPrep\relax
877 \let\@noligs\relax
878 \FV@Scan
879 }
880
881 \def\FVE@VerbatimOutAppend{
882 \seq_use:Nn \l_NLN_records_seq /
883 \endgroup
884 \@esphack
885 }
886
887 \DefineVerbatimEnvironment{VerbatimOutAppend}{VerbatimOutAppend}{}
888
889 % =====
890 % Main options
891 % =====
892
893 \newif\ifNLN@left
894 \newif\ifNLN@right
895
896 % some settings used by fancyvrb:
897 % * for line numbering:
898 % numbers, numbersep, firstnumber, stepnumber, numberblanklines
899 % * for selection of lines to print:
900 % firstline, lastline,
901
902 \pgfkeys{%
903 /NLN/.cd,
904 %
905 boxing-method/.code = \NLN_put:nn {boxin_method} { #1 },
906 inline-method/.code = \NLN_put:nn {inline_method} { #1 },
907 %
908 lang/.code = \NLN_put:nn {lang} { #1 },
909 sty/.code = \NLN_put:nn {sty} { #1 },
910 escapeinside/.code = \NLN_put:nn {escapeinside} { #1 },
911 texcomments/.code = \NLN_put:nn {texcomments} { #1 },% boolean
912 mathescape/.code = \NLN_put:nn {mathescape} { #1 },% boolean
913 %
914 label/.code = \NLN_put:nn {label} { #1 },
915 caption/.code = \NLN_put:nn {caption} { #1 },
916 %
917 gobble/.code = \NLN_put:nn {gobble} { #1 },
918 tabsize/.code = \NLN_put:nn {tabsize} { #1 },
919 %
920 linenos/.code = \NLN_put:nn {linenos} { #1 },% boolean
921 linenostart/.code = \NLN_put:nn {linenostart} { #1 },
922 linenostep/.code = \NLN_put:nn {linenostep} { #1 },
923 linenosep/.code = \NLN_put:nn {linenosep} { #1 },
924 %

```

```

925 colback/.code          = \NLN_put:nn {colback} { #1 },
926 font/.code             = \NLN_put:nn {font} { #1 },
927 %
928 texcomments/.default = true,
929 mathescape/.default  = true,
930 linenos/.default     = true,
931 }
932
933 \pgfqkeys{/NLN}{
934   boxing-method = mdframed,
935   inline-method = efbox,
936   sty           = default,
937   linenos       = false,
938   linenossep    = 2pt,
939   font          = \ttfamily,
940   tabsize       = 0,
941 }
942
943 % =====
944 % pygmented commands and environments
945 % =====
946
947 \newwrite\NLN@outfile
948
949 \newcount\NLN@counter
950
951 \newcommand\NLN@process@options[1]{%
952   \pgfkeys{%
953     /pgf/key~filters/defined/.install~key~filter,%
954     /pgf/key~filter~handlers/append~filtered~to/.install~key~filter~handler=\NLNRemainingGlo
955   }%
956   \def\NLNRemainingGlobalOptions{}%
957   \pgfkeysalsofilteredfrom{\NLN@global@options}%
958   \pgfkeysalso{%
959     /pgf/key~filter~handlers/append~filtered~to/.install~key~filter~handler=\NLNRemainingUse
960   }%
961   \def\NLNRemainingUserOptions{}%
962   \pgfqkeysfiltered{/NLN}{#1}%
963   % %%%% DEBUGING
964   % \typeout{}%
965   % \typeout{\string\NLN@global@options:}\typeout{\meaning\NLN@global@options}%
966   % \typeout{\string\NLNRemainingGlobalOptions:}\typeout{\meaning\NLNRemainingGlobalOptions}%
967   % \typeout{\string\NLNRemainingUserOptions:}\typeout{\meaning\NLNRemainingUserOptions}%
968   %
969   \fvset{gobble=0,tabsize=0}%
970 }
971
972 \newcommand\NLN@process@more@options[1]{%
973   \pgfkeysalso{%
974     /pgf/key~filters/false/.install~key~filter,%
975     /pgf/key~filter~handlers/append~filtered~to/.install~key~filter~handler=\NLNRemainingOpt
976   }%
977   \def\NLNRemainingOptions{}%
978   \pgfkeysalsofilteredfrom{\NLNRemainingGlobalOptions}%

```

```

979 \cs_if_exist:cT {NLN@#1@more@options} {
980   \exp_args:Nx
981   \pgfkeysalsofilteredfrom { \use:c{NLN@#1@more@options}, }
982 }
983 \pgfkeysalsofilteredfrom{\NLNRemainingUserOptions}%
984 % %%% DEBUGING
985 % \typeout{}%
986 % \typeout{\string\NLNRemainingOptions:}%
987 % \typeout{meaning\NLNRemainingOptions}%
988 }
989
990 \newcommand\inputpygmented[2] [] {%
991   \begingroup
992     \NLN@process@options{#1}%
993     \immediate\write\NLN@outfile{<@@NLN@input@the\NLN@counter}%
994     \immediate\write\NLN@outfile{\exp_args:NV\detokenize\NLN@global@options,\detokenize{#1}}%
995     \immediate\write\NLN@outfile{#2}%
996     \immediate\write\NLN@outfile{>@@NLN@input@the\NLN@counter}%
997     %
998     \csname NLN@snippet@the\NLN@counter\endcsname
999     \global\advance\NLN@counter by 1\relax
1000   \endgroup
1001 }
1002
1003 \NewDocumentEnvironment{pygmented}{+0{m}}{%
1004   \directlua{NLN:start_recording()}
1005   \NLN@process@options{#1}%
1006   \immediate\write\NLN@outfile{<@@NLN@display@the\NLN@counter}%
1007   \immediate\write\NLN@outfile{
1008     \exp_args:NV\detokenize\NLN@global@options,\detokenize{#1}
1009   }%
1010   \VerbatimEnvironment
1011   \begin{VerbatimOutAppend}{\NLN@outfile}%
1012 }{%
1013   \end{VerbatimOutAppend}%
1014   \immediate\write\NLN@outfile{>@@NLN@display@the\NLN@counter}%
1015   \csname NLN@snippet@the\NLN@counter\endcsname
1016   \global\advance\NLN@counter by 1\relax
1017 }
1018
1019 \newcommand\pyginline[2] [] {%
1020   \begingroup
1021     \typeout{DEBUG1}
1022     \prop_set_eq:Nc \l_NLN_prop {g/NLN/prop}
1023     \cs_set:Npn \NLN_put:nn #1 #2 {
1024       \prop_put:Nnn \l_NLN_prop { #1 } { #2 }
1025     }
1026     \typeout{DEBUG2}
1027     \NLN@process@options{#1}%
1028     \typeout{DEBUG3}
1029     \directlua{NLN:clear_options()}
1030     \typeout{DEBUG4}
1031     \prop_map_inline:Nn \l_NLN_prop {
1032       \typeout{DEBUG5/#1/#2/}

```

```

1033     \directlua{NLN:add_option([===[#1]===], [===[#2]===])}
1034   }
1035   \DefineShortVerb{#2}%
1036   \SaveVerb
1037     [aftersave={%
1038       \UndefineShortVerb{#2}%
1039       \directlua{NLN:process_inline([===[\FV@SV@NLN]===])}
1040       \endgroup
1041     }]%
1042   {NLN}#2%
1043 }
1044
1045 \cs_generate_variant:Nn \exp_last_unbraced:NnNo { NxNo }
1046
1047 \newcommand\NLN@snippet@inlined[1]{%
1048   \group_begin:
1049   \typeout{DEBUG~PY~STYLE:<\NLN@opt@style>}
1050   \use_c:n { PYstyledefault }
1051   \tl_if_empty:NF \NLN@opt@style {
1052     \use_c:n { PYstyle\NLN@opt@style }
1053   }
1054   \cs_if_exist:cTF {PY} {PYOK} {PYKO}
1055   \NLN@opt@font
1056   \NLN@process@more@options{ \NLN_item:n { inline_method } }%
1057   \exp_last_unbraced:NxNo
1058   \use:c { \NLN_item:n { inline_method } } [ \NLNRemainingOptions ]{#1}%
1059   \group_end:
1060 }
1061
1062 % ERROR: JL undefined \NLN@alllinenos
1063
1064 \ProvideDocumentCommand\captionof{mm}{-}{
1065   \def\NLN@alllinenos{(0)}
1066   \prg_new_conditional:Nnn \NLN_yorn:n { T, F, TF } {
1067     \group_begin:
1068     \prop_get:cnNT {g/NLN/code/} { #1 } \l_tmpa_tl {
1069       \exp_args:NnV
1070       \regex_match:nnT {^[tTyY]} \l_tmpa_tl {
1071         \group_end:
1072         \prg_return_true:
1073       }
1074     } {
1075       \group_end:
1076       \prg_return_false:
1077     }
1078   \newenvironment{NLN@snippet@framed}{%
1079     \group_begin:
1080     \NLN@leftmargin\z@
1081     \NLN_yorn:nT {linenos} {
1082       \expandafter \NLNwidest\NLN@alllinenos{\FormatLineNumber}{\NLN@leftmargin}%
1083       \exp_args:NNx
1084       \advance\NLN@leftmargin { \NLN_item:n {linenosep} }
1085     }
1086     %

```

```

1087 \tl_clear:N \l_NLN_tl
1088 \NLN_get:nNTF {label} \l_tmpa_tl {
1089   \tl_set:N \l_NLN_tl {%
1090     \captionof{pygcode}{\label{\NLN_item:n {label}} \NLN_item:n {caption}}}%
1091     % \nopagebreak
1092     \vskip -0.7\baselineskip
1093   }%
1094 } {
1095   \NLN_get:nNT {caption} \l_tmpa_tl {
1096     \tl_set:N \l_NLN_tl {%
1097       \captionof {pygcode} {\l_tmpa_tl}%
1098       % \nopagebreak
1099       \vskip -0.7\baselineskip
1100     }%
1101   \fi
1102 }
1103 \l_NLN_tl
1104 %
1105 \exp_args:Nx \tl_if_empty:nF { \NLN_item:n {boxing_method} } {
1106   \exp_args:Nx
1107   \NLN@process@more@options { \NLN_item:n {boxing_method} }%
1108   \exp_last_unbraced:NxNo
1109   \begin { \NLN_item:n {boxing_method} } [ \NLNRemainingOptions ]
1110 }
1111 \csname PYstyle\NLN@opt@style\endcsname
1112 \NLN@opt@font
1113 \noindent
1114 } {
1115   \exp_args:Nx \tl_if_empty:nF { \NLN_item:n {boxing_method} } {
1116     \exp_args:Nx
1117     \end { \NLN_item:n {boxing_method} }
1118   }
1119   \group_end:
1120 }
1121
1122
1123 \newcommand\NLN@inlined[1]{%
1124   \exp_last_unbraced:NNV
1125   \efbox[\NLNRemainingOptions]{#1}%
1126 }
1127
1128 \def\FormatLineNumber#1{{\rmfamily\tiny#1}}
1129
1130
1131 \newdimen\NLN@leftmargin
1132 \newdimen\NLN@linenosep
1133
1134 \def\NLN@lineno@do#1{%
1135   \NLN@linenosep Opt%
1136   \use:c { \NLN@ \NLN_item:n {boxing_method} @margin }
1137   \exp_args:NNx
1138   \advance \NLN@linenosep { \NLN_item:n {linenosep} }
1139   \hbox_overlap_left:n {%
1140     \FormatLineNumber{#1}%

```



```

1141     \hspace*{\NLN@linenosep}}}%
1142 }
1143
1144 \newcommand\NLN@tcbox@more@options{%
1145     nobeforeafter,%
1146     tcbox~raise~base,%
1147     left=0mm,%
1148     right=0mm,%
1149     top=0mm,%
1150     bottom=0mm,%
1151     boxsep=2pt,%
1152     arc=1pt,%
1153     boxrule=0pt,%
1154     \NLN_if_in:nNT {colback} {
1155         colback=\NLN_item:n {colback}
1156     }
1157 }
1158
1159 \newcommand\NLN@efbox@more@options{%
1160     \NLN_if_in:nNT {colback} {
1161         backgroundcolor=\NLN_item:n {colback}
1162     }
1163 }
1164
1165 \newcommand\NLN@mdframed@more@options{%
1166     leftmargin=\NLN@leftmargin,%
1167     frametitlerule=true,%
1168     \NLN_if_in:nNT {colback} {
1169         backgroundcolor=\NLN_item:n {colback}
1170     }
1171 }
1172
1173 \newcommand\NLN@tcolorbox@more@options{%
1174     grow~to~left~by=-\NLN@leftmargin,%
1175     \NLN_if_in:nNT {colback} {
1176         colback=\NLN_item:n {colback}
1177     }
1178 }
1179
1180 \newcommand\NLN@boite@more@options{%
1181     leftmargin=\NLN@leftmargin,%
1182     \ifcsname NLN@opt@colback\endcsname
1183         colback=\NLN@opt@colback,%
1184     \fi
1185 }
1186
1187 \newcommand\NLN@mdframed@margin{%
1188     \advance \NLN@linenosep \mdflength{outerlinewidth}%
1189     \advance \NLN@linenosep \mdflength{middlelinewidth}%
1190     \advance \NLN@linenosep \mdflength{innerlinewidth}%
1191     \advance \NLN@linenosep \mdflength{innerleftmargin}%
1192 }
1193
1194 \newcommand\NLN@tcolorbox@margin{%

```

```

1195 \advance \NLN@linenosep \kvtcb@left@rule
1196 \advance \NLN@linenosep \kvtcb@leftupper
1197 \advance \NLN@linenosep \kvtcb@boxsep
1198 }
1199
1200 \newcommand\NLN@boite@margin{%
1201 \advance \NLN@linenosep \boite@leftrule
1202 \advance \NLN@linenosep \boite@boxsep
1203 }
1204
1205 \def\NLN@global@options{
1206
1207 \newcommand\setpygmented[1]{%
1208 \def\NLN@global@options{/NLN/.cd,#1}%
1209 }
1210
1211
1212 % =====
1213 % final actions
1214 % =====
1215
1216 \AtEndOfPackage{%
1217 \IfFileExists{\jobname.pygmented}{%
1218 \input{\jobname.pygmented}%
1219 }{%
1220 \PackageWarning{inline}{File ‘\jobname.pygmented’ not found.}%
1221 }%
1222 \immediate\openout\NLN@outfile\jobname.snippets%
1223 }
1224
1225 \AtEndDocument{%
1226 \closeout\NLN@outfile%
1227 }
1228 \ExplSyntaxOff
1229 </package>

```