beamer named overlay specification with beanoves

Jérôme Laurens

v1.0 2022/10/28

Abstract

This package allows the management of multiple slide lists in `beamer` documents. Slide lists are very handy both during edition and to manage complex and variable beamer overlay specifications.

Contents

1 Minimal example

The document below is a contrived example to show how the `beamer` overlay specifications have been extended.

```
1  \documentclass {beamer}
2  \RequirePackage {beanoves}
3  \begin{document}
4  \begin{frame} [
5    beanoves = {
6      A = 1:2,
7      B = A.next:3,
8      C = B.next,
9    }
10 ]
11 {\Large Frame \insertframenumber}
12 {\Large Slide \insertslidenumber}
13 \visible<?(A.1)> {Only on slide 1}\\
14 \visible<?(B.1)-?(B.last)> {Only on slide 3 to 5}\\
15 \visible<?(C.1)> {Only on slide 6}\\
16 \visible<?(A.2)> {Only on slide 2}\\
17 \visible<?(B.2)-?(B.last)> {Only on slide 4 to 5}\\
18 \visible<?(C.2)> {Only on slide 7}\\
19 \visible<?(A.3)-> {From slide 3}\\
20 \visible<?(B.3)-?(B.last)> {Only on slide 5}\\
21 \visible<?(C.3)> {Only on slide 8}\\
22 \end{frame}
23 \end{document}
```

On line 5, we use the dedicated `beanoves` key to declare named slide ranges. On line 6, we declare a slide range named 'A', starting at slide 1 and with length 2. On line 13, the extended *named overlay specification* `?(A.1)` stands for 1, on line 16, `?(A.2)` stands for 2 whereas on line 19, `?(A.3)` stands for 3. On line 7, we declare a second slide range named 'B', starting after the 2 slides of 'A' namely 3. Its length is 3 meaning that its last slide number is 5, thus each `?(B.last)` is replaced by 5. The next slide number after slide range 'B' is 6 which is also the start of the third slide range due to line 8.

2 Named slide lists

2.1 Presentation

Within a beamer frame, there are different slides that appear in turn. The main slide list is a range on integers covering all the slide numbers, from one to the total amount of slides. In general, a slide list is a range of positive integers identified by a unique name. The main practical interest is that such lists may be defined relative to one another, we can even have lists of slide ranges. Finally, we can use these lists to organize beamer overlay specifications logically.

2.2 Defining named slide lists

In order to define named slide lists, we can either use the `\Beanoves` command below inside a `beamer` frame environment, or use the `beanoves` option of this environment. The value of the `beanoves` option is exactly the argument of the `\Beanoves` command. When used, the `\Beanoves` command is executed for each frame, whereas the option is executed only once but is a bit more verbose.

`beanoves`

```
beanoves={
  ⟨name_1⟩=⟨spec_1⟩,
  ⟨name_2⟩=⟨spec_2⟩,
  ...,
  ⟨name_n⟩=⟨spec_n⟩,
}
```

`\Beanoves`

```
\Beanoves{
  ⟨name_1⟩=⟨spec_1⟩,
  ⟨name_2⟩=⟨spec_2⟩,
  ...,
  ⟨name_n⟩=⟨spec_n⟩,
}
```

The keys $\langle name_i \rangle$ are the slide lists names, they are case sensitive and must contain no spaces nor '`/`' character. In order to avoid name conflicts with floating point functions, it is suggested to let them contain an uppercase letter ot an underscore. When the same key is used multiple times, only the last one is taken into account. Possible values for $\langle spec_i \rangle$ are the *slide range specifiers* $\langle first \rangle$, $\langle first \rangle$:$\langle length \rangle$, $\langle first \rangle$::$\langle last \rangle$, :$\langle length \rangle$::$\langle last \rangle$ where $\langle first \rangle$, $\langle length \rangle$ and $\langle last \rangle$ are algebraic expression involving any integer valued named overlay specifications defined below.

Also possible values are *slide list specifiers* which are comma separated list of *slide range specifiers* and *slide list specifier* between square brackets. The definition

$\langle name \rangle$=[$\langle spec_1 \rangle$,$\langle spec_2 \rangle$,...,$\langle spec_n \rangle$],

is a convenient shortcut for

$\langle name \rangle$.1=$\langle spec_1 \rangle$,
$\langle name \rangle$.2=$\langle spec_2 \rangle$,
...,
$\langle name \rangle$.n=$\langle spec_n \rangle$.

The rules above can apply individually to each

$\langle name \rangle$.i=$\langle spec_i \rangle$.

Moreover we can go deeper: the definition

$\langle name \rangle$=[[$\langle spec_{1.1} \rangle$, $\langle spec_{1.2} \rangle$],[[$\langle spec_{2.1} \rangle$, $\langle spec_{2.2} \rangle$]]

is a convenient shortcut for

$\langle name \rangle$.1.1=$\langle spec_{1.1} \rangle$,
$\langle name \rangle$.1.2=$\langle spec_{1.2} \rangle$,
$\langle name \rangle$.2.1=$\langle spec_{2.1} \rangle$,
$\langle name \rangle$.2.2=$\langle spec_{2.2} \rangle$

and so on.

The `\Beanoves` command is used at the very beginning of the `frame` environment body and thus only apply to this frame. It can be used there mutliple times. The `\Beanoves` command does not override what is set by the `beanoves` frame option, which

allows to input the very same source code into different frames and have different combinations of slides.

3 Named overlay specifications

3.1 Named slide ranges

When *slide range specifications* are used, the named overlay specifications are detailled in the tables below together with their replacement meaning value as beamer standard overlay specification.

⟨*name*⟩ == [i, $i+1$, $i+2$, ...]	
syntax	**meaning**
⟨*name*⟩.1	i
⟨*name*⟩.2	$i+1$
⟨*name*⟩.⟨*integer*⟩	$i+\langle integer\rangle-1$

In the frame example below, we use the `\BeanovesEval` command for the demonstration. It is mainly used for debugging and testing purposes.

```
\begin{frame} [
  beanoves = {
    A = 3:6,
  }
] {Frame \insertframenumber} {Slide \insertslidenumber}
\ttfamily
\BeanovesEval(A.1) ==3,
\BeanovesEval(A.2) ==4,
\BeanovesEval(A.-1)==1,
\end{frame}
```

When the slide range has been given a length or an end, like in the frame example below, we also have

⟨*name*⟩ == [i, $i+1$, ..., j]			
syntax	**meaning**	**example**	**output**
⟨*name*⟩.length	$j-i+1$	A.length	6
⟨*name*⟩.last	j	A.last	8
⟨*name*⟩.next	$j+1$	A.next	9
⟨*name*⟩.range	i ''-'' j	A.range	3-8

```
\begin{frame} [
  beanoves = {
    A = 3:6,
  }
] {Frame \insertframenumber} {Slide \insertslidenumber}
\ttfamily
\BeanovesEval(A.length) == 6,
\BeanovesEval(A.1)      == 3,
\BeanovesEval(A.2)      == 4,
\BeanovesEval(A.-1)     == 1,
\end{frame}
```

Using these specification on unfinite named slide ranges is unsupported. Finally each named slide range has a dedicated counter ⟨*name*⟩`.n` which is some kind of variable that can be used and incremented.

⟨*name*⟩`.n` : use the position of the counter

⟨*name*⟩`.n+=`⟨*integer*⟩ : advance the counter by ⟨*integer*⟩ and use the new position

`++`⟨*name*⟩`.n` : advance the counter by 1 and use the new position

Notice that "`.n`" can generally be omitted.

3.2 Named slide lists

After the definition
⟨*name*⟩`=[`⟨$spec_1$⟩`,`⟨$spec_2$⟩`,...,`⟨$spec_n$⟩`]`
the rules of the previous section apply recursively to each individual declaration
⟨*name*⟩`.`i`=`⟨$spec_i$⟩.

4 `?(...)` query expressions

This is the key feature of the beanoves package, extending beamer *overlay specifications* included between pointed brackets. Before the *overlay specifications* are processed by the beamer class, the beanoves package scans them for any occurrence of '`?(`⟨*queries*⟩`)`'. Each one is then evaluated and replaced by its static counterpart. The overall result is finally forwarded to the beamer class.

The ⟨*queries*⟩ argument is a comma separated list of individual ⟨*query*⟩'s of next table. Sometimes, using ⟨*name*⟩`.range` is not allowed as it would lead to an algeabraic difference instead of a range.

query	static value	limitation
`:`	`-`	
`::`	`-`	
⟨*first expr*⟩	⟨*first*⟩	
⟨*first expr*⟩`:`	⟨*first*⟩ `-`	no ⟨*name*⟩`.range`
⟨*first expr*⟩`::`	⟨*first*⟩ `-`	no ⟨*name*⟩`.range`
⟨*first expr*⟩`:`⟨*length expr*⟩	⟨*first*⟩ `-` ⟨*last*⟩	no ⟨*name*⟩`.range`
⟨*first expr*⟩`::`⟨*end expr*⟩	⟨*first*⟩ `-` ⟨*last*⟩	no ⟨*name*⟩`.range`

Here ⟨*first expr*⟩, ⟨*length expr*⟩ and ⟨*end expr*⟩ both denote algebraic expressions possibly involving named overlay specifications and counters. As integers, they respectively evaluate to ⟨*first*⟩, ⟨*length*⟩ and ⟨*last*⟩.

For example both `?(A.next)`, `?(A.last+1)`, `?(A.1+A.length)` give the same result as soon as the slide range named '`A`' has been properly defined with a length.

Notice that nesting `?(...)` expressions is not supported.

```
⟨*package⟩
```

5 Implementation

Identify the internal prefix (LaTeX3 DocStrip convention).

```
⟨@@=beanoves⟩
```

5.1 Package declarations

```
\NeedsTeXFormat{LaTeX2e}[2020/01/01]
\ProvidesExplPackage
  {beanoves}
  {2022/10/28}
  {1.0}
  {Named overlay specifications for beamer}
```

5.2 Local variables

We make heavy use of local variables and function scopes. Many functions are executed within a TeX group, which ensures no name collision with the caller stack. In that case, variables need not follow exactly the LaTeX3 naming convention: we do not specialize with the module name. On execution, next initialization instructions declare the variables as side effect.

```
\int_zero_new:N  \l__beanoves_split_int
\int_zero_new:N  \l__beanoves_depth_int
\int_zero_new:N  \g__beanoves_append_int
\bool_new:N \l__beanoves_no_counter_bool
\bool_new:N \l__beanoves_no_range_bool
\bool_new:N \l__beanoves_continue_bool
```

5.3 Overlay specification

5.3.1 In slide range definitions

`\g__beanoves_prop` ⟨*key*⟩–⟨*value*⟩ property list to store the named slide lists. The basic keys are, assuming ⟨*name*⟩ is a slide list identifier,

⟨***name***⟩**`/A`** for the first index

⟨***name***⟩**`/L`** for the length when provided

⟨***name***⟩**`/Z`** for the last index when provided

⟨***name***⟩**`/C`** for the counter value, when used

⟨***name***⟩**`/C0`** for initial value of the counter (when reset)

Other keys are eventually used to cache results when some attributes are defined from other slide ranges. They are characterized by a '//'.

⟨*name*⟩**//A** for the cached static value of the first index

⟨*name*⟩**//Z** for the cached static value of the last index

⟨*name*⟩**//L** for the cached static value of the length

⟨*name*⟩**//N** for the cached static value of the next index

The implementation is private, in particular, keys may change in future versions.

```
\prop_new:N \g__beanoves_prop
```

(*End definition for* `\g__beanoves_prop`.)

`\__beanoves_gput:nn`	`\__beanoves_gput:nn {⟨key⟩} {⟨value⟩}`
`\__beanoves_gput:nV`	
`\__beanoves_item:n`	`\__beanoves_item:n {⟨key⟩}`
`\__beanoves_get:nN`	`\__beanoves_get:n {⟨key⟩} ⟨tl variable⟩`
`\__beanoves_gremove:n`	`\__beanoves_gremove:n {⟨key⟩}`
`\__beanoves_gclear:n`	`\__beanoves_gclear:n {⟨key⟩}`
`\__beanoves_gclear:`	`\__beanoves_gclear:`

Convenient shortcuts to manage the storage, it makes the code more concise and readable.

```
\cs_new:Npn \__beanoves_gput:nn {
  \prop_gput:Nnn \g__beanoves_prop
}
\cs_new:Npn \__beanoves_item:n {
  \prop_item:Nn \g__beanoves_prop
}
\cs_new:Npn \__beanoves_get:nN {
  \prop_get:NnN \g__beanoves_prop
}
\cs_new:Npn \__beanoves_gremove:n {
  \prop_gremove:Nn \g__beanoves_prop
}
\cs_new:Npn \__beanoves_gclear:n #1 {
  \clist_map_inline:nn { A, L, Z, C, C0, /A, /L, /Z, /N } {
    \__beanoves_gremove:n { #1 / ##1 }
  }
}
\cs_new:Npn \__beanoves_gclear: {
  \prop_gclear:N \g__beanoves_prop
}
\cs_generate_variant:Nn \__beanoves_gput:nn { nV }
```

__beanoves_if_in_p:n ⋆
__beanoves_if_in_p:V ⋆
__beanoves_if_in:nTF ⋆
__beanoves_if_in:VTF ⋆

__beanoves_if_in_p:n {⟨*key*⟩}
__beanoves_if_in:nTF {⟨*key*⟩} {⟨*true code*⟩} {⟨*false code*⟩}

Convenient shortcuts to test for the existence of some key, it makes the code more concise and readable.

```
\prg_new_conditional:Npnn \__beanoves_if_in:n #1 { p, T, F, TF } {
  \prop_if_in:NnTF \g__beanoves_prop { #1 } {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
\prg_generate_conditional_variant:Nnn \__beanoves_if_in:n {V} { p, T, F, TF }
```

__beanoves_get:nNTF

__beanoves_get:nNTF {⟨*key*⟩} ⟨*tl variable*⟩ {⟨*true code*⟩} {⟨*false code*⟩}

Convenient shortcuts to retrieve the value with branching, it makes the code more concise and readable. Execute ⟨*true code*⟩ when the item is found, ⟨*false code*⟩ otherwise. In the latter case, the content of the ⟨*tl variable*⟩ is undefined.

```
\prg_new_conditional:Npnn \__beanoves_get:nN #1 #2 { p, T, F, TF } {
  \prop_get:NnNTF \g__beanoves_prop { #1 } #2 {
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
```

Utility message.

```
\msg_new:nnn { beanoves } { :n } { #1 }
```

5.3.2 Regular expressions

\c__beanoves_name_regex

The name of a slide range consists of a non void list of alphanumerical characters and underscore, but with no leading digit.

```
\regex_const:Nn \c__beanoves_name_regex {
  [[:alpha:]_][[:alnum:]_]*
}
```

(*End definition for* \c__beanoves_name_regex.)

\c__beanoves_path_regex

A sequence of .⟨*positive integer*⟩ items representing a path.

```
\regex_const:Nn \c__beanoves_path_regex {
  (?: \. \d+ )*
}
```

(*End definition for* \c__beanoves_path_regex.)

\c__beanoves_key_regex
\c__beanoves_A_key_Z_regex

A key is the name of a slide range possibly followed by positive integer attributes using a dot syntax. The 'A_key_Z' variant matches the whole string.

```
\regex_const:Nn \c__beanoves_key_regex {
  \ur{c__beanoves_name_regex} \ur{c__beanoves_path_regex}
}
\regex_const:Nn \c__beanoves_A_key_Z_regex {
  \A \ur{c__beanoves_key_regex} \Z
}
```

(*End definition for* `\c__beanoves_key_regex` *and* `\c__beanoves_A_key_Z_regex`.)

`\c__beanoves_dotted_regex` A specifier is the name of a slide range possibly followed by attributes using a dot syntax. This is a poor man version to save computations, a dedicated parser would help in error management.

```
\regex_const:Nn \c__beanoves_dotted_regex {
  \A \ur{c__beanoves_name_regex} (?: \. [^.]+ )* \Z
}
```

(*End definition for* `\c__beanoves_dotted_regex`.)

`\c__beanoves_colons_regex` For ranges defined by a colon syntax.

```
\regex_const:Nn \c__beanoves_colons_regex { :(:+)? }
```

(*End definition for* `\c__beanoves_colons_regex`.)

`\c__beanoves_int_regex` A decimal integer with an eventual leading sign next to the first digit.

```
\regex_const:Nn \c__beanoves_int_regex {
  (?:[-+])? \d+
}
```

(*End definition for* `\c__beanoves_int_regex`.)

`\c__beanoves_list_regex` A comma separated list between square brackets.

```
\regex_const:Nn \c__beanoves_list_regex {
  \A \[ \s*
```

Capture groups:

- 2: the content between the brackets, outer spaces trimmed out

```
    ( [^\]]*? )
  \s* \] \Z
}
```

(*End definition for* `\c__beanoves_list_regex`.)

`\c__beanoves_split_regex` Used to parse slide list overlay specifications in queries. Next are the 10 capture groups. Group numbers are 1 based because the regex is used in splitting contexts where only capture groups are considered and not the whole match.

```
\regex_const:Nn \c__beanoves_split_regex {
  \s* ( ? :
```

We start with '++' instrussions [1].

- 1: ⟨*name*⟩ of a slide range

```
    \+\+ ( \ur{c__beanoves_name_regex} )
```

- 2: optionally followed by an integer path

```
    ( \ur{c__beanoves_path_regex} ) (?: \. n )?
```

We continue with other expressions

[1] At the same time an instruction and an expression... this is a synonym of exprection

- 3: ⟨*name*⟩ of a slide range

```
  | ( \ur{c__beanoves_name_regex} )
```

- 4: optionally followed by an integer path

```
    ( \ur{c__beanoves_path_regex} )
```

Next comes another branching

```
    (?:
```

- 5: the ⟨*length*⟩ attribute

```
        \. l(e)ngth
```

- 6: the ⟨*last*⟩ attribute

```
    |  \. l(a)st
```

- 7: the ⟨*next*⟩ attribute

```
    |  \. ne(x)t
```

- 8: the ⟨*range*⟩ attribute

```
    |  \. (r)ange
```

- 9: the ⟨*n*⟩ attribute

```
    |  \. (n)
```

- 10: the poor man integer expression after '`+=`'. When it contains no parenthesis, it is an algebraic expression involving integers and ⟨*key*⟩'s. Otherwise it starts with a parenthesis and ends with the first parenthesis followed by a white space or the end of the text. This tricky definition allows quite any algebraic expression involving parenthesis. The problems may arise when dealing with nested expressions.

```
        (?: \s* \+= \s*
          ( (?: \ur{c__beanoves_int_regex} | \ur{c__beanoves_key_regex} )
            (?: [+\-*/] (?: \d+ | \ur{c__beanoves_key_regex}) )*
          | \( .*? \) (?: \Z | \s+ )
          )
        )?


    )?

  ) \s*
}
```

(*End definition for* `\c__beanoves_split_regex`.)

5.3.3 Defining named slide ranges

`\__beanoves_error:n` Prints an error message when a key only item is used.

```
\cs_new:Npn \__beanoves_error:n #1 {
  \msg_fatal:nnn { beanoves } { :n } { Missing~value~for~#1 }
}
```

`\__beanoves_parse:nn` `\__beanoves_parse:nn {⟨key⟩} {⟨definition⟩}`

Auxiliary function called within a group. ⟨*name*⟩ is the slide key, including eventually a dotted integer path, ⟨*definition*⟩ is the corresponding definition.

`\l_match_seq` Local storage for the match result.

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

__beanoves_range:nnnn
__beanoves_range:nVVV
__beanoves_range_alt:nnnn
__beanoves_range_alt:nVVV

__beanoves_range:nnnn {⟨*key*⟩} {⟨*first*⟩} {⟨*length*⟩} {⟨*last*⟩}
__beanoves_range_alt:nnnn {⟨*key*⟩} {⟨*first*⟩} {⟨*length*⟩} {⟨*last*⟩}

Auxiliary function called within a group. Setup the model to define a range. The alt variant does not override an already existing value.

```
\cs_new:Npn \__beanoves_range:nnnn #1 #2 #3 #4 {
  \__beanoves_gclear:n { #1 }
  \tl_if_empty:nTF { #2 } {
    \tl_if_empty:nTF { #3 } {
      \tl_if_empty:nTF { #4 } {
        \msg_error:nnn { beanoves } { :n } { Not~a~range:~:~#1 }
      } {
        \__beanoves_gput:nn { #1/Z } { #4 }
      }
    } {
      \__beanoves_gput:nn { #1/L } { #3 }
      \tl_if_empty:nF { #4 } {
        \__beanoves_gput:nn { #1/Z } { #4 }
        \__beanoves_gput:nn { #1/A } { #1.last - (#1.length) + 1 }
      }
    }
  } {
    \__beanoves_gput:nn { #1/A } { #2 }
    \tl_if_empty:nTF { #3 } {
      \tl_if_empty:nF { #4 } {
        \__beanoves_gput:nn { #1/Z } { #4 }
        \__beanoves_gput:nn { #1/L } { #1.last - (#1.1) + 1 }
      }
    } {
      \__beanoves_gput:nn { #1/L } { #3 }
      \__beanoves_gput:nn { #1/Z } { #1.1 + #1.length - 1 }
    }
  }
}
\cs_generate_variant:Nn \__beanoves_range:nnnn { nVVV }
\cs_new:Npn \__beanoves_range_alt:nnnn #1 {
  \__beanoves_if_in:nTF {#1/A} {
    \use_none:nnn
  } {
    \__beanoves_range:nnnn { #1 }
  }
}
\cs_generate_variant:Nn \__beanoves_range_alt:nnnn { nVVV }
```

```
\cs_generate_variant:Nn \tl_if_empty:nTF { xTF }
\cs_new:Npn \__beanoves_do_parse:Nnn #1 #2 #3 {
```

The first argument has signature `nVVV`. This is not a list.

```
  \tl_clear:N \l_a_tl
  \tl_clear:N \l_b_tl
  \tl_clear:N \l_c_tl
  \regex_split:NnN \c__beanoves_colons_regex { #3 } \l_split_seq
  \seq_pop_left:NNT \l_split_seq \l_a_tl {
```

`\l_a_tl` may contain the ⟨*start*⟩.

```
    \seq_pop_left:NNT \l_split_seq \l_b_tl {
      \tl_if_empty:NTF \l_b_tl {
```

This is a one colon range.

```
        \seq_pop_left:NN \l_split_seq \l_b_tl
```

`\l_b_tl` may contain the ⟨*length*⟩.

```
        \seq_pop_left:NNT \l_split_seq \l_c_tl {
          \tl_if_empty:NTF \l_c_tl {
```

A `::` was expected:

```
\msg_error:nnn { beanoves } { :n } { Invalid~range~expression(1):~#3 }
          } {
            \int_compare:nNnT { \tl_count:N \l_c_tl } > { 1 } {
\msg_error:nnn { beanoves } { :n } { Invalid~range~expression(2):~#3 }
            }
            \seq_pop_left:NN \l_split_seq \l_c_tl
```

`\l_c_tl` may contain the ⟨*end*⟩.

```
            \seq_if_empty:NF \l_split_seq {
\msg_error:nnn { beanoves } { :n } { Invalid~range~expression(3):~#3 }
            }
          }
        }
      } {
```

This is a two colon range.

```
        \int_compare:nNnT { \tl_count:N \l_b_tl } > { 1 } {
\msg_error:nnn { beanoves } { :n } { Invalid~range~expression(4):~#3 }
        }
        \seq_pop_left:NN \l_split_seq \l_c_tl
```

`\l_c_tl` contains the ⟨*end*⟩.

```
        \seq_pop_left:NNTF \l_split_seq \l_b_tl {
          \tl_if_empty:NTF \l_b_tl {
            \seq_pop_left:NN \l_split_seq \l_b_tl
```

`\l_b_tl` may contain the ⟨*length*⟩.

```
            \seq_if_empty:NF \l_split_seq {
\msg_error:nnn { beanoves } { :n } { Invalid~range~expression(5):~#3 }
            }
          } {
\msg_error:nnn { beanoves } { :n } { Invalid~range~expression(6):~#3 }
          }
        } {
          \tl_clear:N \l_b_tl
        }
      }
    }
  }
```

Prividing both the ⟨*start*⟩, ⟨*length*⟩ and ⟨*end*⟩ of a range is not allowed, even if they happen to be consistent.

```
  \bool_if:nF {
    \tl_if_empty_p:N \l_a_tl
    || \tl_if_empty_p:N \l_b_tl
    || \tl_if_empty_p:N \l_c_tl
```

```
  } {
\msg_error:nnn { beanoves } { :n } { Invalid~range~expression(7):~#3 }
  }
  #1 { #2 } \l_a_tl \l_b_tl \l_c_tl
}
```

```
\cs_new:Npn \__beanoves_parse:Nnn #1 #2 #3 {
  \group_begin:
  \regex_match:NnTF \c__beanoves_A_key_Z_regex { #2 } {
```

We got a valid key.

```
    \regex_extract_once:NnNTF \c__beanoves_list_regex { #3 } \l_match_seq {
```

This is a comma separated list, extract each item and go recursive.

```
      \exp_args:NNx
      \seq_set_from_clist:Nn \l_match_seq {
        \seq_item:Nn \l_match_seq { 2 }
      }
      \seq_map_indexed_inline:Nn \l_match_seq {
        \__beanoves_do_parse:Nnn #1 { #2.##1 } { ##2 }
      }
    } {
      \__beanoves_do_parse:Nnn #1 { #2 } { #3 }
    }
  } {
    \msg_error:nnn { beanoves } { :n } { Invalid~key:~#1 }
  }
  \group_end:
}
```

\Beanoves

`\Beanoves {`⟨*key--value list*⟩`}`

The keys are the slide range specifiers. We do not accept key only items, they are managed by `\__beanoves_error:n`. On the contrary, ⟨*key–value*⟩ items are parsed by `\__beanoves_parse:Nnn`.

```
\cs_new:Npn \__beanoves:n #1 {
  \keyval_parse:nnn { \__beanoves_error:n } { \__beanoves_parse:Nnn \__beanoves_range:nVVV }
}
\NewDocumentCommand \Beanoves { m } {
  \keyval_parse:nnn { \__beanoves_error:n } { \__beanoves_parse:Nnn \__beanoves_range_alt:nVV
  \ignorespaces
}
```

If we use this command in the frame body, it will be executed for each different frame. If we use the frame option `beanoves` instead, the command is executed only once, at the cost of a more verbose code.

```
\define@key{beamerframe}{beanoves}{\Beanoves{#1}}
```

5.3.4 Scanning named overlay specifications

Patch some beamer command to support `?(...)` instructions in overlay specifications.

`\beamer@masterdecode` `\beamer@masterdecode {⟨overlay specification⟩}`

Preprocess ⟨*overlay specification*⟩ before beamer uses it.

`\l_ans_tl` Storage for the translated overlay specification, where `?(...)` instructions are replaced by their static counterparts.

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

Save the original macro `\beamer@masterdecode` and then override it to properly preprocess the argument.

```
\cs_set_eq:NN \__beanoves_beamer@masterdecode \beamer@masterdecode
\cs_set:Npn \beamer@masterdecode #1 {
  \group_begin:
  \tl_clear:N \l_ans_tl
  \__beanoves_scan:nNN { #1 } \__beanoves_eval:nN \l_ans_tl
  \exp_args:NNV
  \group_end:
  \__beanoves_beamer@masterdecode \l_ans_tl
}
```

`\__beanoves_scan:nNN` `\__beanoves_scan:nNN {⟨named overlay expression⟩} ⟨eval⟩ ⟨tl variable⟩`

Scan the ⟨*named overlay expression*⟩ argument and feed the ⟨*tl variable*⟩ replacing `?(...)` instructions by their static counterpart with help from the ⟨*eval*⟩ function, which is `\__beanoves_eval:nN`. A group is created to use local variables:

`\l_ans_tl`: is the token list that will be appended to ⟨*tl variable*⟩ on return.

`\l__beanoves_depth_int` Store the depth level in parenthesis grouping used when finding the proper closing parenthesis balancing the opening parenthesis that follows immediately a question mark in a `?(...)` instruction.

(*End definition for* `\l__beanoves_depth_int`.)

`g__beanoves_append_int` Decremented each time `\__beanoves_append:nN` is called. To avoid catch circular definitions.

(*End definition for* `g__beanoves_append_int`.)

`\l_query_tl` Storage for the overlay query expression to be evaluated.

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

`\l_token_seq` The ⟨*overlay expression*⟩ is split into the sequence of its tokens.

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

`\l_ask_bool` Whether a loop may continue. Controls the continuation of the main loop that scans the tokens of the ⟨*named overlay expression*⟩ looking for a question mark.

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

`\l_query_bool` Whether a loop may continue. Controls the continuation of the secondary loop that scans the tokens of the ⟨*overlay expression*⟩ looking for an opening parenthesis follow the question mark. It then controls the loop looking for the balanced closing parenthesis.

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

`\l_token_tl` Storage for just one token.

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

```
\cs_new:Npn \__beanoves_scan:nNN #1 #2 #3 {
  \group_begin:
  \tl_clear:N \l_ans_tl
  \int_zero:N \l__beanoves_depth_int
  \seq_clear:N \l_token_seq
```

Explode the ⟨*named overlay expression*⟩ into a list of tokens:

```
  \regex_split:nnN {} { #1 } \l_token_seq
```

Run the top level loop to scan for a '?':

```
  \bool_set_true:N  \l_ask_bool
  \bool_while_do:Nn \l_ask_bool {
    \seq_pop_left:NN \l_token_seq \l_token_tl
    \quark_if_no_value:NTF \l_token_tl {
```

We reached the end of the sequence (and the token list), we end the loop here.

```
      \bool_set_false:N \l_ask_bool
    } {
```

`\l_token_tl` contains a 'normal' token.

```
      \tl_if_eq:NnTF \l_token_tl { ? } {
```

We found a '`?`', we first gobble tokens until the next '`(`', whatever they may be. In general, no tokens should be silently ignored.

```
        \bool_set_true:N \l_query_bool
        \bool_while_do:Nn \l_query_bool {
```

Get next token.

```
          \seq_pop_left:NN \l_token_seq \l_token_tl
          \quark_if_no_value:NTF \l_token_tl {
```

No opening parenthesis found, raise.

```
            \msg_fatal:nnx { beanoves } { :n } {Missing~'('%---)
              ~after~a~?:~#1}
          } {
            \tl_if_eq:NnT \l_token_tl { ( %)
            } {
```

We found the '`(`' after the '`?`'. Increment the parenthesis depth to 1 (on first passage).

```
              \int_incr:N \l__beanoves_depth_int
```

Record the forthcomming content in the `\l_query_tl` variable, up to the next balancing '`)`'.

```
              \tl_clear:N \l_query_tl
              \bool_while_do:Nn \l_query_bool {
```

Get next token.

```
                \seq_pop_left:NN \l_token_seq \l_token_tl
                \quark_if_no_value:NTF \l_token_tl {
```

We reached the end of the sequence and the token list with no closing '`)`'. We raise and end both bool while loops. As recovery we feed `\l_query_tl` with the missing '`)`'. `\l_@@_depth_int` is 0 whenever `\l_query_bool` is false.

```
                  \msg_error:nnx { beanoves } { :n } {Missing~%(---
                    `)':~#1 }
                  \int_do_while:nNnn \l__beanoves_depth_int > 1 {
                    \int_decr:N \l__beanoves_depth_int
                    \tl_put_right:Nn \l_query_tl {%(---
                    )}
                  }
                  \int_zero:N \l__beanoves_depth_int
                  \bool_set_false:N \l_query_bool
                  \bool_set_false:N \l_ask_bool
                } {
                  \tl_if_eq:NnTF \l_token_tl { ( %---)
                  } {
```

We found a '`(`', increment the depth and append the token to `\l_query_tl`.

```
                    \int_incr:N \l__beanoves_depth_int
                    \tl_put_right:NV \l_query_tl \l_token_tl
                  } {
```

This is not a '('.

```
                    \tl_if_eq:NnTF \l_token_tl { %(
                      )
                    } {
```

We found a ')', decrement the depth.

```
                      \int_decr:N \l__beanoves_depth_int
                      \int_compare:nNnTF \l__beanoves_depth_int = 0 {
```

The depth level has reached 0: we found our balancing parenthesis of the `?(...)` instruction. We can append the evaluated slide ranges token list to `\l_ans_tl` and stop the inner loop.

```
  \exp_args:NV #2 \l_query_tl \l_ans_tl
  \bool_set_false:N \l_query_bool
                      } {
```

The depth has not yet reached level 0. We append the ')' to `\l_query_tl` because it is not the end of sequence marker.

```
                        \tl_put_right:NV \l_query_tl \l_token_tl
                      }
```

Above ends the code for a positive depth.

```
                    } {
```

The scanned token is not a '(' nor a ')', we append it as is to `\l_query_tl`.

```
                      \tl_put_right:NV \l_query_tl \l_token_tl
                    }
                  }
                }
```

Above ends the code for Not a '('

```
              }
            }
```

Above ends the code for: Found the '(' after the '?'

```
          }
```

Above ends the code for not a no value quark.

```
        }
```

Above ends the code for the bool while loop to find the '(' after the '?'.

If we reached the end of the token list, then end both the current loop and its containing loop.

```
        \quark_if_no_value:NT \l_token_tl {
          \bool_set_false:N \l_query_bool
          \bool_set_false:N \l_ask_bool
        }
      } {
```

This is not a '?', append the token to right of `\l_ans_tl` and continue.

```
        \tl_put_right:NV \l_ans_tl \l_token_tl
      }
```

Above ends the code for the bool while loop to find a '(' after the '?'

```
    }
  }
```

Above ends the outer bool while loop to find '?' characters. We can append our result to ⟨*tl variable*⟩

```
  \exp_args:NNNV
  \group_end:
  \tl_put_right:Nn #3 \l_ans_tl
}
```

Each new frame has its own set of slide ranges, we clear the property list on entering a new frame environment. Frame environments nested into other frame environments are not supported.

```
\AddToHook
  { env/beamer@framepauses/before }
  { \prop_gclear:N \g__beanoves_prop }
```

5.3.5 Evaluation bricks

`\__beanoves_fp_round:nN`
`\__beanoves_fp_round:N`

`\__beanoves_fp_round:nN` {⟨*expression*⟩} ⟨*tl variable*⟩
`\__beanoves_fp_round:N` ⟨*tl variable*⟩

Shortcut for `\fp_eval:n{round(`⟨*expression*⟩`)}` appended to ⟨*tl variable*⟩. The second variant replaces the variable content with its rounded floating point evaluation.

```
\cs_new:Npn \__beanoves_fp_round:nN #1 #2 {
  ROUND:\tl_to_str:n{#1}/\string#2=\tl_to_str:V #2\\
  \tl_if_empty:nTF { #1 } {
    VOID\\
  } {
    \msg_term:nnx { beanoves } { :n } { ROUND:~\exp_args:Nx\tl_to_str:n{#1} ^^J }
    \tl_put_right:Nx #2 {
      \fp_eval:n { round(#1) }
    }
  }
}
\cs_generate_variant:Nn \__beanoves_fp_round:nN { VN, xN }
\cs_new:Npn \__beanoves_fp_round:N #1 {
  ROUND:\string#1=\tl_to_str:V #1\\
  \tl_if_empty:VTF #1 {
    VOID\\
  } {
    \msg_term:nnx { beanoves } { :n } { ROUND:~\exp_args:Nx\tl_to_str:n{#1} ^^J }
    \tl_set:Nx #1 {
      \fp_eval:n { round(#1) }
    }
  }
}
```

`\__beanoves_if_first_p:nN` ⋆
`\__beanoves_if_first:nNTF` ⋆

`\__beanoves_if_first:nNTF` {⟨*name*⟩} ⟨*tl variable*⟩ {⟨*true code*⟩} {⟨*false code*⟩}

Append the first index of the ⟨*name*⟩ slide range to the ⟨*tl variable*⟩. Cache the result. Execute ⟨*true code*⟩ when there is a ⟨*first*⟩, ⟨*false code*⟩ otherwise.

```
\prg_new_conditional:Npnn \__beanoves_if_first:nN #1 #2 { p, T, F, TF } {
  IF_FIRST:\tl_to_str:n{#1}/\string #2=\tl_to_str:V #2\\
```

```
\__beanoves_if_in:nTF { #1//A } {
  CACHED\\
  \tl_put_right:Nx #2 { \__beanoves_item:n { #1//A } }
  \prg_return_true:
} {
  \group_begin:
  \cs_set:Npn \prg_return_true: {
    \tl_if_empty:NTF \l_ans_tl {
      \group_end:
      \prg_return_false:
    } {
      \__beanoves_fp_round:N \l_ans_tl
      \__beanoves_gput:nV { #1//A } \l_ans_tl
      \exp_args:NNNV
      \group_end:
      \tl_put_right:Nn #2 \l_ans_tl
      \prg_return_true:
    }
  }
  \cs_set:Npn \prg_return_false: {
    \group_end:
    \prg_return_false:
  }
  \tl_clear:N \l_ans_tl
  \__beanoves_get:nNTF { #1/A } \l_a_tl {
    \__beanoves_if_append:VNTF \l_a_tl \l_ans_tl {
      \prg_return_true:
    } {
      \prg_return_false:
    }
  } {
    \bool_if:nTF {
         \__beanoves_get_p:nN { #1/L } \l_a_tl
      && \__beanoves_get_p:nN { #1/Z } \l_b_tl
    } {
      \__beanoves_if_append:xNTF {
        \l_b_tl - ( \l_a_tl - 1 )
      } \l_ans_tl {
        \prg_return_true:
      } {
        \prg_return_false:
      }
    } {
      \__beanoves_get:nNTF { #1/C } \l_a_tl {
        \bool_set_true:N \l_no_counter_bool
        \__beanoves_if_append:xN \l_a_tl \l_ans_tl {
          \prg_return_true:
        } {
          \prg_return_false:
        }
      } {
        \prg_return_false:
      }
    }
```

```
    }
  }
}
```

__beanoves_first:nN
__beanoves_first:VN

__beanoves_first:nN {⟨*name*⟩} ⟨*tl variable*⟩

Append the start of the ⟨*name*⟩ slide range to the ⟨*tl variable*⟩. Cache the result.

```
\cs_new:Npn \__beanoves_first:nN #1 #2 {
  \__beanoves_if_first:nNF { #1 } #2 {
    \msg_error:nnn { beanoves } { :n } { Range~with~no~first:~#1 }
  }
}
\cs_generate_variant:Nn \__beanoves_first:nN { VN }
```

IF_FIRST:Y/\l_ans_tl=
*******************/X=123**
IF_FIRST:X/\l_ans_tl=

FAILURE ‘’!=‘123’

Test `\__beanoves_first:nN` **1**

*******************/X=123**
IF_FIRST:X/\l_ans_tl=

FAILURE ‘’!=‘123’

Test `\__beanoves_first:nN` **2**

*******************/A=11,X=:A::NONE**
IF_FIRST:X/\l_ans_tl=

FAILURE ‘’!=‘-9’

Test `\__beanoves_first:nN` **3**

*******************/A=11,X=::C:A**
IF_FIRST:X/\l_ans_tl=

FAILURE ‘’!=‘-9’

Test `\__beanoves_first:nN` **4**

*******************/A=11,X=A.5**
IF_FIRST:X/\l_ans_tl=

FAILURE ‘’!=‘15’

Test `\__beanoves_first:nN` **5**

*******************/X.123=456**
IF_FIRST:X.123/\l_ans_tl=

⛔ **FAILURE ‘’!=‘456’**

⛔ **Test `\__beanoves_first:nN` 6**

`\__beanoves_if_length_p:nN` ⋆
`\__beanoves_if_length:nNTF` ⋆

`\__beanoves_if_length_p:nN` {⟨*name*⟩} ⟨*tl variable*⟩
`\__beanoves_if_length:nNTF` {⟨*name*⟩} ⟨*tl variable*⟩ {⟨*true code*⟩} {⟨*false code*⟩}

Append the length of the ⟨*name*⟩ slide range to ⟨*tl variable*⟩ Execute ⟨*true code*⟩ when there is a ⟨*length*⟩, ⟨*false code*⟩ otherwise.

```
\prg_new_conditional:Npnn \__beanoves_if_length:nN #1 #2 { p, T, F, TF } {
  \__beanoves_if_in:nTF { #1//L } {
    \tl_put_right:Nx #2 { \__beanoves_item:n { #1//L } }
    \prg_return_true:
  } {
    \__beanoves_gput:nn { #1//L } { 0 }
    \group_begin:
    \cs_set:Npn \prg_return_true: {
      \tl_if_empty:NTF \l_ans_tl {
        \group_end:
        \prg_return_false:
      } {
        \__beanoves_fp_round:N \l_ans_tl
        \__beanoves_gput:nV { #1//A } \l_ans_tl
        \exp_args:NNNV
        \group_end:
        \tl_put_right:Nn #2 \l_ans_tl
        \prg_return_true:
      }
    }
    \cs_set:Npn \prg_return_false: {
      \group_end:
      \prg_return_false:
    }
    \tl_clear:N \l_ans_tl
    \__beanoves_if_in:nTF { #1/L } {
      \__beanoves_if_append:xNTF {
        \__beanoves_item:n { #1/L }
      } \l_ans_tl {
        \prg_return_true:
      } {
        \prg_return_false:
      }
    } {
      \__beanoves_get:nNTF { #1/A } \l_a_tl {
        \__beanoves_get:nNTF { #1/Z } \l_b_tl {
          \__beanoves_if_append:xNTF {
            \l_b_tl - (\l_a_tl - 1)
          } \l_ans_tl {
            \prg_return_true:
          } {
            \prg_return_false:
          }
        } {
```

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

__beanoves_length:nN
__beanoves_length:VN

__beanoves_length:nN {⟨*name*⟩} ⟨*tl variable*⟩

Append the length of the ⟨*name*⟩ slide range to ⟨*tl variable*⟩

********** **1:2**
X.1+X.length-1-(1-1)

- **FAILURE ‘’!=‘2’**
- **Test** `\__beanoves_length:nN` **1-a**
- **FAILURE ‘0’!=‘2’**
- **Test** `\__beanoves_length:nN` **1-b**

********** **:2::3**
3-(X.last-(X.length)+1-1)

- **FAILURE ‘’!=‘2’**
- **Test** `\__beanoves_length:nN` **2-a**
- **FAILURE ‘0’!=‘2’**
- **Test** `\__beanoves_length:nN` **2-b**

********** **::3:2**
3-(X.last-(X.length)+1-1)

- **FAILURE ‘’!=‘2’**
- **Test** `\__beanoves_length:nN` **3-a**
- **FAILURE ‘0’!=‘2’**
- **Test** `\__beanoves_length:nN` **3-b**

********** **A:B,A=1,B=2,C=3**
X.1+X.length-1-(A-1)

- **FAILURE ‘’!=‘2’**
- **Test** `\__beanoves_length:nN` **4-a**
- **FAILURE ‘0’!=‘2’**

⛔ **Test __beanoves_length:nN 4-b**

********** :B::C,A=1,B=2,C=3
C-(X.last-(X.length)+1-1)

⛔ **FAILURE ‘’!=‘2’**

⛔ **Test __beanoves_length:nN 5-a**

⛔ **FAILURE ‘0’!=‘2’**

⛔ **Test __beanoves_length:nN 5-b**

********** ::C:B,A=1,B=2,C=3
C-(X.last-(X.length)+1-1)

⛔ **FAILURE ‘’!=‘2’**

⛔ **Test __beanoves_length:nN 6-a**

⛔ **FAILURE ‘0’!=‘2’**

⛔ **Test __beanoves_length:nN 6-b**

```
\cs_new:Npn \__beanoves_length:nN #1 #2 {
  \__beanoves_if_length:nNF { #1 } #2 {
    \msg_error:nnn { beanoves } { :n } { Range~with~no~length:~#1 }
  }
}
\cs_generate_variant:Nn \__beanoves_length:nN { VN }
```

__beanoves_if_last_p:nN ⋆	__beanoves_if_last_p:nN {⟨*name*⟩} ⟨*tl variable*⟩
__beanoves_if_last:nN*TF* ⋆	__beanoves_if_last:nNTF {⟨*name*⟩} ⟨*tl variable*⟩ {⟨*true code*⟩} {⟨*false code*⟩}

```
\prg_new_conditional:Npnn \__beanoves_if_last:nN #1 #2 { p, T, F, TF } {
  IF_LAST/#1\\
  \__beanoves_if_in:nTF { #1//Z } {
    \tl_put_right:Nx #2 { \__beanoves_item:n { #1//Z } }
    \prg_return_true:
  }  {
    \__beanoves_gput:nn { #1//Z } { 1 }
    \group_begin:
    \cs_set:Npn \prg_return_true: {
      \tl_if_empty:NTF \l_ans_tl {
        \group_end:
        \prg_return_false:
      } {
        \__beanoves_fp_round:N \l_ans_tl
        \__beanoves_gput:nV { #1//Z } \l_ans_tl
        \exp_args:NNNV
        \group_end:
        \tl_put_right:Nn #2 \l_ans_tl
        \prg_return_true:
```

```
      }
    }
    \cs_set:Npn \prg_return_false: {
      \group_end:
      \prg_return_false:
    }
    \tl_clear:N \l_ans_tl
    \__beanoves_if_in:nTF { #1/Z } {
      NORMAL_LAST:\exp_args:Nx \tl_to_str:n { \__beanoves_item:n { #1/Z } }\\
      \__beanoves_if_append:xNTF {
        \__beanoves_item:n { #1/Z }
      } \l_ans_tl {
        \prg_return_true:
      } {
        \prg_return_false:
      }
    } {
      \__beanoves_get:nNTF { #1/A } \l_a_tl {
        \__beanoves_get:nNTF { #1/L } \l_b_tl {
          \__beanoves_if_append:xNTF {
            \l_a_tl + \l_b_tl - 1
          } \l_ans_tl {
            \prg_return_true:
          } {
            \prg_return_false:
          }
        } {
          \prg_return_false:
        }
      } {
        \prg_return_false:
      }
    }
  }
}
```

`\__beanoves_last:nN`
`\__beanoves_last:VN`

`\__beanoves_last:nN` {⟨*name*⟩} ⟨*tl variable*⟩

Append the last index of the ⟨*name*⟩ slide range to ⟨*tl variable*⟩

```
\cs_new:Npn \__beanoves_last:nN #1 #2 {
  \__beanoves_if_last:nNF { #1 } #2 {
    \msg_error:nnn { beanoves } { :n } { Range~with~no~last:~#1 }
  }
}
\cs_generate_variant:Nn \__beanoves_last:nN { VN }
```

********** **::4**

IF_LAST/X

⛔ **FAILURE ‘’!=‘4’**

⛔ **Test `\__beanoves_last:nN` 1-a**

IF_LAST/X

- **FAILURE ‘1’!=‘4’**
- **Test** `\__beanoves_last:nN` **1-c**

IF_LAST/Y
IF_LAST/X

- **FAILURE ‘1’!=‘4’**
- **Test** `\__beanoves_last:nN` **1-a**

IF_LAST/X

- **FAILURE ‘1’!=‘4’**
- **Test** `\__beanoves_last:nN` **1-c**

IF_LAST/Y

- **FAILURE ‘FAILURE’!=‘SUCCESS’**
- **Test** `\__beanoves_last:nN` **1-d**

********** **2::4**
IF_LAST/X
2+X.last-(X.1)+1-1

- **FAILURE ‘’!=‘4’**
- **Test** `\__beanoves_last:nN` **2-a**

IF_LAST/X

- **FAILURE ‘1’!=‘4’**
- **Test** `\__beanoves_last:nN` **2-c**

IF_LAST/Y

- **FAILURE ‘FAILURE’!=‘SUCCESS’**
- **Test** `\__beanoves_last:nN` **2-d**

IF_LAST/X

- **FAILURE ‘1’!=‘4’**
- **Test** `\__beanoves_last:nN` **2-a**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **2-c**

IF_LAST/Y

⛔ **FAILURE ‘FAILURE’!=‘SUCCESS’**

⛔ **Test** `\__beanoves_last:nN` **2-d**

********** **:2::4**
IF_LAST/X
X.last-(X.length)+1+2-1

⛔ **FAILURE ‘’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **3-a**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **3-c**

IF_LAST/Y

⛔ **FAILURE ‘FAILURE’!=‘SUCCESS’**

⛔ **Test** `\__beanoves_last:nN` **3-d**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **3-a**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **3-c**

IF_LAST/Y

⛔ **FAILURE ‘FAILURE’!=‘SUCCESS’**

⛔ **Test** `\__beanoves_last:nN` **3-d**

********** **::4:2**
IF_LAST/X
X.last-(X.length)+1+2-1

⛔ **FAILURE ‘’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **4-a**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **4-c**

IF_LAST/Y

⛔ **FAILURE ‘FAILURE’!=‘SUCCESS’**

⛔ **Test** `\__beanoves_last:nN` **4-d**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **4-a**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **4-c**

IF_LAST/Y

⛔ **FAILURE ‘FAILURE’!=‘SUCCESS’**

⛔ **Test** `\__beanoves_last:nN` **4-d**

********** **2:3**
IF_LAST/X
2+3-1

⛔ **FAILURE ‘’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **5-a**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **5-c**

IF_LAST/Y

⛔ **FAILURE ‘FAILURE’!=‘SUCCESS’**

⛔ **Test** `\__beanoves_last:nN` **5-d**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **5-a**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **5-c**

IF_LAST/Y

⛔ **FAILURE ‘FAILURE’!=‘SUCCESS’**

⛔ **Test** `\__beanoves_last:nN` **5-d**

********** **::C,A=2,B=3,C=4**
IF_LAST/X

⛔ **FAILURE ‘’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **6-a**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **6-c**

IF_LAST/Y

⛔ **FAILURE ‘FAILURE’!=‘SUCCESS’**

⛔ **Test** `\__beanoves_last:nN` **6-d**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **6-a**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **6-c**

IF_LAST/Y

⛔ **FAILURE ‘FAILURE’!=‘SUCCESS’**

⛔ **Test** `\__beanoves_last:nN` **6-d**

********** ::C
IF_LAST/X

FAILURE ‘’!=‘4’
Test __beanoves_last:nN 7-a

IF_LAST/X

FAILURE ‘1’!=‘4’
Test __beanoves_last:nN 7-c

IF_LAST/Y

FAILURE ‘FAILURE’!=‘SUCCESS’
Test __beanoves_last:nN 7-d

IF_LAST/X

FAILURE ‘1’!=‘4’
Test __beanoves_last:nN 7-a

IF_LAST/X

FAILURE ‘1’!=‘4’
Test __beanoves_last:nN 7-c

IF_LAST/Y

FAILURE ‘FAILURE’!=‘SUCCESS’
Test __beanoves_last:nN 7-d

********** A::C
IF_LAST/X
A+X.last-(X.1)+1-1

FAILURE ‘’!=‘4’
Test __beanoves_last:nN 8-a

IF_LAST/X

FAILURE ‘1’!=‘4’
Test __beanoves_last:nN 8-c

IF_LAST/Y

⛔ **FAILURE ‘FAILURE’!=‘SUCCESS’**

⛔ **Test** `\__beanoves_last:nN` **8-d**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **8-a**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **8-c**

IF_LAST/Y

⛔ **FAILURE ‘FAILURE’!=‘SUCCESS’**

⛔ **Test** `\__beanoves_last:nN` **8-d**

********** **A::C**
IF_LAST/X
A+X.last-(X.1)+1-1

⛔ **FAILURE ‘’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **9-a**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **9-c**

IF_LAST/Y

⛔ **FAILURE ‘FAILURE’!=‘SUCCESS’**

⛔ **Test** `\__beanoves_last:nN` **9-d**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **9-a**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **9-c**

IF_LAST/Y

⛔ **FAILURE ‘FAILURE’!=‘SUCCESS’**

⛔ **Test** `\__beanoves_last:nN` **9-d**

********** **A:B**
IF_LAST/X
A+B-1

⛔ **FAILURE ‘’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **10-a**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **10-c**

IF_LAST/Y

⛔ **FAILURE ‘FAILURE’!=‘SUCCESS’**

⛔ **Test** `\__beanoves_last:nN` **10-d**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **10-a**

IF_LAST/X

⛔ **FAILURE ‘1’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **10-c**

IF_LAST/Y

⛔ **FAILURE ‘FAILURE’!=‘SUCCESS’**

⛔ **Test** `\__beanoves_last:nN` **10-d**

********** **A:B**
IF_LAST/X
A+B-1

⛔ **FAILURE ‘’!=‘4’**

⛔ **Test** `\__beanoves_last:nN` **11-a**

IF_LAST/X

⛔ **FAILURE '1'!='4'**

⛔ **Test __beanoves_last:nN 11-c**

IF_LAST/Y

⛔ **FAILURE 'FAILURE'!='SUCCESS'**

⛔ **Test __beanoves_last:nN 11-d**

IF_LAST/X

⛔ **FAILURE '1'!='4'**

⛔ **Test __beanoves_last:nN 11-a**

IF_LAST/X

⛔ **FAILURE '1'!='4'**

⛔ **Test __beanoves_last:nN 11-c**

IF_LAST/Y

⛔ **FAILURE 'FAILURE'!='SUCCESS'**

⛔ **Test __beanoves_last:nN 11-d**

`\__beanoves_if_next_p:nN` ⋆ `\__beanoves_if_next_p:nN` {⟨*name*⟩} ⟨*tl variable*⟩

`\__beanoves_if_next:nNTF` ⋆ `\__beanoves_if_next:nNTF` {⟨*name*⟩} ⟨*tl variable*⟩ {⟨*true code*⟩} {⟨*false code*⟩}

Append the index after the ⟨*name*⟩ slide range to the ⟨*tl variable*⟩. Execute ⟨*true code*⟩ when there is a ⟨*next*⟩ index, ⟨*false code*⟩ otherwise.

```
\prg_new_conditional:Npnn \__beanoves_if_next:nN #1 #2 { p, T, F, TF } {
  \__beanoves_if_in:nTF { #1//N } {
    \tl_put_right:Nx #2 { \__beanoves_item:n { #1//N } }
    \prg_return_true:
  } {
    \group_begin:
    \cs_set:Npn \prg_return_true: {
      \tl_if_empty:NTF \l_ans_tl {
        \group_end:
        \prg_return_false:
      } {
        \__beanoves_fp_round:N \l_ans_tl
        \__beanoves_gput:nV { #1//N } \l_ans_tl
        \exp_args:NNNV
        \group_end:
        \tl_put_right:Nn #2 \l_ans_tl
        \prg_return_true:
      }
```

```
    }
    \cs_set:Npn \prg_return_false: {
      \group_end:
      \prg_return_false:
    }
    \tl_clear:N \l_a_tl
    \__beanoves_if_last:nNTF { #1 } \l_a_tl {
      \__beanoves_if_append:xNTF {
        \l_a_tl + 1
      } \l_ans_tl {
        \prg_return_true:
      } {
        \prg_return_false:
      }
    } {
      \prg_return_false:
    }
  }
}
```

__beanoves_next:nN
__beanoves_next:VN

__beanoves_next:nN {⟨*name*⟩} ⟨*tl variable*⟩

Append the index after the ⟨*name*⟩ slide range to the ⟨*tl variable*⟩.

```
\cs_new:Npn \__beanoves_next:nN #1 #2 {
  \__beanoves_if_next:nNF { #1 } #2 {
    \msg_error:nnn { beanoves } { :n } { Range~with~no~next:~#1 }
  }
}
\cs_generate_variant:Nn \__beanoves_next:nN { VN }
```

********** ::4

IF_LAST/X

⛔ **FAILURE ‘’!=‘5’**

⛔ **Test __beanoves_next:nN 1-a**

IF_LAST/X

⛔ **FAILURE ‘FAILURE’!=‘SUCCESS’**

⛔ **Test __beanoves_next:nN 1-b**

⛔ **FAILURE ‘’!=‘5’**

⛔ **Test __beanoves_next:nN 1-c**

IF_LAST/Y
IF_LAST/X

⛔ **FAILURE ‘’!=‘5’**

⛔ **Test __beanoves_next:nN 1-a**

IF_LAST/X

⛔ FAILURE ‘FAILURE’!=‘SUCCESS’

⛔ Test __beanoves_next:nN 1-b

⛔ FAILURE ‘’!=‘5’

⛔ Test __beanoves_next:nN 1-c

IF_LAST/Y
********** 2::4
IF_LAST/X
2+X.last-(X.1)+1-1
2+1

⛔ FAILURE ‘’!=‘5’

⛔ Test __beanoves_next:nN 2-a

IF_LAST/X
2+X.last-(X.1)+1-1
2+1

⛔ FAILURE ‘2’!=‘SUCCESS’

⛔ Test __beanoves_next:nN 2-b

⛔ FAILURE ‘’!=‘5’

⛔ Test __beanoves_next:nN 2-c

IF_LAST/Y
IF_LAST/X
2+X.last-(X.1)+1-1
2+1

⛔ FAILURE ‘’!=‘5’

⛔ Test __beanoves_next:nN 2-a

IF_LAST/X
2+X.last-(X.1)+1-1
2+1

⛔ FAILURE ‘2’!=‘SUCCESS’

⛔ Test __beanoves_next:nN 2-b

⛔ FAILURE ‘’!=‘5’

⛔ Test __beanoves_next:nN 2-c

IF_LAST/Y
********** :2::4
IF_LAST/X
X.last-(X.length)+1+2-1
X.last-(X.length)+1+1

FAILURE ‘’!=‘5’

Test __beanoves_next:nN 3-a

IF_LAST/X
X.last-(X.length)+1+2-1
X.last-(X.length)+1+1

FAILURE ‘X.last-(X.length)+1’!=‘SUCCESS’

Test __beanoves_next:nN 3-b

FAILURE ‘’!=‘5’

Test __beanoves_next:nN 3-c

IF_LAST/Y
IF_LAST/X
X.last-(X.length)+1+2-1
X.last-(X.length)+1+1

FAILURE ‘’!=‘5’

Test __beanoves_next:nN 3-a

IF_LAST/X
X.last-(X.length)+1+2-1
X.last-(X.length)+1+1

FAILURE ‘X.last-(X.length)+1’!=‘SUCCESS’

Test __beanoves_next:nN 3-b

FAILURE ‘’!=‘5’

Test __beanoves_next:nN 3-c

IF_LAST/Y
********** ::4:2
IF_LAST/X
X.last-(X.length)+1+2-1
X.last-(X.length)+1+1

FAILURE ‘’!=‘5’

Test __beanoves_next:nN 4-a

```
IF__LAST/X
X.last-(X.length)+1+2-1
 X.last-(X.length)+1+1

⛔ FAILURE ‘X.last-(X.length)+1’!=‘SUCCESS’
⛔ Test \__beanoves_next:nN 4-b

⛔ FAILURE ‘’!=‘5’
⛔ Test \__beanoves_next:nN 4-c

IF__LAST/Y
IF__LAST/X
X.last-(X.length)+1+2-1
 X.last-(X.length)+1+1

⛔ FAILURE ‘’!=‘5’
⛔ Test \__beanoves_next:nN 4-a

IF__LAST/X
X.last-(X.length)+1+2-1
 X.last-(X.length)+1+1

⛔ FAILURE ‘X.last-(X.length)+1’!=‘SUCCESS’
⛔ Test \__beanoves_next:nN 4-b

⛔ FAILURE ‘’!=‘5’
⛔ Test \__beanoves_next:nN 4-c

IF__LAST/Y
******************** 2:3
IF__LAST/X
2+3-1
 2+1

⛔ FAILURE ‘’!=‘5’
⛔ Test \__beanoves_next:nN 5-a

IF__LAST/X
2+3-1
 2+1

⛔ FAILURE ‘2’!=‘SUCCESS’
⛔ Test \__beanoves_next:nN 5-b

⛔ FAILURE ‘’!=‘5’
⛔ Test \__beanoves_next:nN 5-c
```

IF_LAST/Y
IF_LAST/X
2+3-1
2+1

- FAILURE ‘’!=‘5’
- Test `\__beanoves_next:nN` 5-a

IF_LAST/X
2+3-1
2+1

- FAILURE ‘2’!=‘SUCCESS’
- Test `\__beanoves_next:nN` 5-b
- FAILURE ‘’!=‘5’
- Test `\__beanoves_next:nN` 5-c

IF_LAST/Y
********** ::C,A=2,B=3,C=4
IF_LAST/X

- FAILURE ‘’!=‘5’
- Test `\__beanoves_next:nN` 6-a

IF_LAST/X

- FAILURE ‘FAILURE’!=‘SUCCESS’
- Test `\__beanoves_next:nN` 6-b
- FAILURE ‘’!=‘5’
- Test `\__beanoves_next:nN` 6-c

IF_LAST/Y
IF_LAST/X

- FAILURE ‘’!=‘5’
- Test `\__beanoves_next:nN` 6-a

IF_LAST/X

- FAILURE ‘FAILURE’!=‘SUCCESS’
- Test `\__beanoves_next:nN` 6-b
- FAILURE ‘’!=‘5’

⛔ **Test __beanoves_next:nN 6-c**

IF_LAST/Y
********** **::C**
IF_LAST/X

⛔ **FAILURE ‘’!=‘5’**

⛔ **Test __beanoves_next:nN 7-a**

IF_LAST/X

⛔ **FAILURE ‘FAILURE’!=‘SUCCESS’**

⛔ **Test __beanoves_next:nN 7-b**

⛔ **FAILURE ‘’!=‘5’**

⛔ **Test __beanoves_next:nN 7-c**

IF_LAST/Y
IF_LAST/X

⛔ **FAILURE ‘’!=‘5’**

⛔ **Test __beanoves_next:nN 7-a**

IF_LAST/X

⛔ **FAILURE ‘FAILURE’!=‘SUCCESS’**

⛔ **Test __beanoves_next:nN 7-b**

⛔ **FAILURE ‘’!=‘5’**

⛔ **Test __beanoves_next:nN 7-c**

IF_LAST/Y
********** **A::C**
IF_LAST/X
A+X.last-(X.1)+1-1
A+1

⛔ **FAILURE ‘’!=‘5’**

⛔ **Test __beanoves_next:nN 8-a**

IF_LAST/X
A+X.last-(X.1)+1-1
A+1

⛔ **FAILURE ‘A’!=‘SUCCESS’**

⛔ **Test __beanoves_next:nN 8-b**

⛔ **FAILURE ‘’!=‘5’**

⛔ **Test** `\__beanoves_next:nN` **8-c**

IF_LAST/Y
IF_LAST/X
A+X.last-(X.1)+1-1
A+1

⛔ **FAILURE ‘’!=‘5’**

⛔ **Test** `\__beanoves_next:nN` **8-a**

IF_LAST/X
A+X.last-(X.1)+1-1
A+1

⛔ **FAILURE ‘A’!=‘SUCCESS’**

⛔ **Test** `\__beanoves_next:nN` **8-b**

⛔ **FAILURE ‘’!=‘5’**

⛔ **Test** `\__beanoves_next:nN` **8-c**

IF_LAST/Y
********** **A::C**
IF_LAST/X
A+X.last-(X.1)+1-1
A+1

⛔ **FAILURE ‘’!=‘5’**

⛔ **Test** `\__beanoves_next:nN` **9-a**

IF_LAST/X
A+X.last-(X.1)+1-1
A+1

⛔ **FAILURE ‘A’!=‘SUCCESS’**

⛔ **Test** `\__beanoves_next:nN` **9-b**

⛔ **FAILURE ‘’!=‘5’**

⛔ **Test** `\__beanoves_next:nN` **9-c**

IF_LAST/Y
IF_LAST/X
A+X.last-(X.1)+1-1
A+1

⛔ **FAILURE ‘’!=‘5’**

⛔ **Test** `\__beanoves_next:nN` **9-a**

IF_LAST/X
A+X.last-(X.1)+1-1
A+1

⛔ **FAILURE 'A'!='SUCCESS'**

⛔ **Test** `\__beanoves_next:nN` **9-b**

⛔ **FAILURE ''!='5'**

⛔ **Test** `\__beanoves_next:nN` **9-c**

IF_LAST/Y
********** **A:B**
IF_LAST/X
A+B-1
A+1

⛔ **FAILURE ''!='5'**

⛔ **Test** `\__beanoves_next:nN` **10-a**

IF_LAST/X
A+B-1
A+1

⛔ **FAILURE 'A'!='SUCCESS'**

⛔ **Test** `\__beanoves_next:nN` **10-b**

⛔ **FAILURE ''!='5'**

⛔ **Test** `\__beanoves_next:nN` **10-c**

IF_LAST/Y
IF_LAST/X
A+B-1
A+1

⛔ **FAILURE ''!='5'**

⛔ **Test** `\__beanoves_next:nN` **10-a**

IF_LAST/X
A+B-1
A+1

⛔ **FAILURE 'A'!='SUCCESS'**

⛔ **Test** `\__beanoves_next:nN` **10-b**

⛔ **FAILURE ''!='5'**

⛔ **Test __beanoves_next:nN 10-c**

IF_LAST/Y
********** **A:B**
IF_LAST/X
A+B-1
A+1

⛔ **FAILURE ‘’!=‘5’**

⛔ **Test __beanoves_next:nN 11-a**

IF_LAST/X
A+B-1
A+1

⛔ **FAILURE ‘A’!=‘SUCCESS’**

⛔ **Test __beanoves_next:nN 11-b**

⛔ **FAILURE ‘’!=‘5’**

⛔ **Test __beanoves_next:nN 11-c**

IF_LAST/Y
IF_LAST/X
A+B-1
A+1

⛔ **FAILURE ‘’!=‘5’**

⛔ **Test __beanoves_next:nN 11-a**

IF_LAST/X
A+B-1
A+1

⛔ **FAILURE ‘A’!=‘SUCCESS’**

⛔ **Test __beanoves_next:nN 11-b**

⛔ **FAILURE ‘’!=‘5’**

⛔ **Test __beanoves_next:nN 11-c**

IF_LAST/Y

`\__beanoves_if_free_counter_p:Nn` ⋆
`\__beanoves_if_free_counter_p:NV` ⋆
`\__beanoves_if_free_counter:NnTF` ⋆
`\__beanoves_if_free_counter:NVTF` ⋆

`\__beanoves_if_free_counter_p:Nn` ⟨*tl variable*⟩ {⟨*name*⟩}
`\__beanoves_if_free_counter_p:NnTF` ⟨*tl variable*⟩ {⟨*name*⟩} {⟨*true code*⟩} {⟨*false code*⟩}

Set the ⟨*tl variable*⟩ to the value of the counter associated to the {⟨*name*⟩} slide range. There is no branching variant because, we always return some value, ‘1’ by default.

```
\prg_new_conditional:Npnn \__beanoves_if_free_counter:Nn #1 #2 { p, T, F, TF } {
  FREE:\string #1/\tl_to_str:n{#2}/\__beanoves_item:n {#2/C}/\\
  \group_begin:
  \cs_set:Npn \prg_return_true: {
    FREE_2:\string \l_ans_tl/\tl_to_str:V \l_ans_tl/\\
    \tl_if_empty:NTF \l_ans_tl {
      \group_end:
      \regex_match:NnTF \c__beanoves_A_key_Z_regex { #2 } {
        \__beanoves_gput:nn { #2/C } { 1 }
        \tl_set:Nn #1 { 1 }
        EERF_ON:\string #1=\tl_to_str:V #1 /\tl_to_str:n{#2}\\
        \prg_return_true:
      } {
        EERF_OFF:\string #1=\tl_to_str:V #1/\tl_to_str:n{#2}\\
        \prg_return_false:
      }
    } {
      \__beanoves_gput:nV { #2/C } \l_ans_tl
      \exp_args:NNNV
      \group_end:
      \tl_set:Nn #1 \l_ans_tl
      EERF:\string #1=\tl_to_str:V #1/\tl_to_str:n{#2}\\
      \prg_return_true:
    }
  }
  \cs_set:Npn \prg_return_false: {
    \group_end:
    \prg_return_false:
  }
  \tl_clear:N \l_ans_tl
  \__beanoves_get:nNTF { #2/C } \l_ans_tl {
    \prg_return_true:
  } {
    \bool_if:nTF {
      \__beanoves_if_in_p:n { #2/A }
      || \__beanoves_if_in_p:n { #2/L }
      && \__beanoves_if_in_p:n { #2/Z }
    } {
      \__beanoves_if_first:nNTF { #2 } \l_ans_tl {
        \prg_return_true:
      } {
        \prg_return_false:
      }
    } {
      \__beanoves_if_last:nNTF { #2 } \l_ans_tl {
        \prg_return_true:
      } {
        \prg_return_false:
      }
    }
  }
}
\prg_generate_conditional_variant:Nnn \__beanoves_if_free_counter:Nn { NV } { p, T, F, TF }
**********1
```

FREE:\l_ans_tl/X//
IF_FIRST:X/\l_ans_tl=

FAILURE ‘’!=‘1’

Test __beanoves_if_free_counter:NnTF 1-a

FAILURE ‘’!=‘SUCCESS’

Test __beanoves_if_free_counter:NnTF 1-a

**********123
FREE:\l_ans_tl/X//
IF_FIRST:X/\l_ans_tl=

FAILURE ‘’!=‘123’

Test __beanoves_if_free_counter:NnTF 2-a

FAILURE ‘’!=‘SUCCESS’

Test __beanoves_if_free_counter:NnTF 2-a

**********::123
FREE:\l_ans_tl/X//
IF_LAST/X

FAILURE ‘’!=‘123’

Test __beanoves_if_free_counter:NnTF 3-a

FAILURE ‘’!=‘SUCCESS’

Test __beanoves_if_free_counter:NnTF 3-a

**********A+B+C
FREE:\l_ans_tl/X//
IF_FIRST:X/\l_ans_tl=

FAILURE ‘’!=‘3’

Test __beanoves_if_free_counter:NnTF 4-a

FAILURE ‘’!=‘SUCCESS’

Test __beanoves_if_free_counter:NnTF 4-a

**********::A+B
FREE:\l_ans_tl/X//
IF_LAST/X

FAILURE ‘’!=‘2’

Test __beanoves_if_free_counter:NnTF 5-a

⛔ **FAILURE ''!='SUCCESS'**

⛔ **Test __beanoves_if_free_counter:NnTF 5-a**

__beanoves_if_counter_p:nN ⋆ __beanoves_if_counter_p:nN {⟨*name*⟩} ⟨*tl variable*⟩
__beanoves_if_counter_p:VN ⋆ __beanoves_if_counter:nNTF {⟨*name*⟩} ⟨*tl variable*⟩ {⟨*true code*⟩} {⟨*false code*⟩}
__beanoves_if_counter:nN*TF* ⋆
__beanoves_if_counter:VN*TF* ⋆

Append the value of the counter associated to the {⟨*name*⟩} slide range to the right of ⟨*tl variable*⟩. The value always lays in between the range, whenever possible.

```
\prg_new_conditional:Npnn \__beanoves_if_counter:nN #1 #2 { p, T, F, TF } {
  COUNTER:\tl_to_str:n{#1}/\string #2=\tl_to_str:V #2\\
  \group_begin:
  \__beanoves_if_free_counter:NnTF \l_ans_tl { #1 } {
```

If there is a ⟨*first*⟩, use it to bound the result from below.

```
    \tl_clear:N \l_a_tl
    \__beanoves_if_first:nNT { #1 } \l_a_tl {
      \fp_compare:nNnT { \l_ans_tl } < { \l_a_tl } {
        \tl_set:NV \l_ans_tl \l_a_tl
      }
    }
```

If there is a ⟨*last*⟩, use it to bound the result from above.

```
    \tl_clear:N \l_a_tl
    \__beanoves_if_last:nNT { #1 } \l_a_tl {
      \fp_compare:nNnT { \l_ans_tl } > { \l_a_tl } {
        \tl_set:NV \l_ans_tl \l_a_tl
      }
    }
    \exp_args:NNx
    \group_end:
    \__beanoves_fp_round:nN \l_ans_tl #2
    RETNUOC:\tl_to_str:n{#1}/\string #2=\tl_to_str:V #2\\
    \prg_return_true:
  } {
    \prg_return_false:
  }
}
\prg_generate_conditional_variant:Nnn \__beanoves_if_counter:nN { VN } { p, T, F, TF }
```

COUNTER:X/\l_ans_tl=
FREE:\l_ans_tl/X//
IF_FIRST:X/\l_ans_tl=

⛔ **FAILURE ''!='1'**

⛔ **Test __beanoves_if_counter:nNTF 1-a**

⛔ **FAILURE ''!='SUCCESS'**

⛔ **Test __beanoves_if_counter:nNTF 1-b**

COUNTER:X/\l_ans_tl=
FREE:\l_ans_tl/X//
IF_FIRST:X/\l_ans_tl=

- **FAILURE ‘’!=‘123’**
- **Test** `\__beanoves_if_counter:nNTF` **2-a**
- **FAILURE ‘’!=‘SUCCESS’**
- **Test** `\__beanoves_if_counter:nNTF` **2-b**

COUNTER:X/\l_ans_tl=
FREE:\l_ans_tl/X//
IF_LAST/X

- **FAILURE ‘’!=‘123’**
- **Test** `\__beanoves_if_counter:nNTF` **3-a**
- **FAILURE ‘’!=‘SUCCESS’**
- **Test** `\__beanoves_if_counter:nNTF` **3-b**

`\__beanoves_index:nnN`
`\__beanoves_index:VVN`

`\__beanoves_index:nnN` {⟨*name*⟩} {⟨*integer path*⟩} ⟨*tl variable*⟩

Appendthevalueofthecounterassociatedtothe{⟨*name*⟩}sliderangetotherightof⟨*tlvariable*⟩.The

```
633 \cs_new:Npn \__beanoves_index:nnN #1 #2 #3 {   \group_begin:   INDEX:#1/#2/\string#3/\\   \t
```

INDEX:A/.1/\l_ans_tl/
IF_FIRST:A.1/\l_b_tl=
IF_FIRST:A/\l_b_tl=
INDEX:\l_ans_tl=A+1-1
ROUND:A+1-1/\l_ans_tl=

- **FAILURE ‘0’!=‘1’**
- **Test** `\__beanoves_index:nnN` **1**

INDEX:A/.2/\l_ans_tl/
IF_FIRST:A.2/\l_b_tl=
IF_FIRST:A/\l_b_tl=
INDEX:\l_ans_tl=A+2-1
ROUND:A+2-1/\l_ans_tl=

- **FAILURE ‘0’!=‘2’**
- **Test** `\__beanoves_index:nnN` **2**

INDEX:B/.2.3/\l_ans_tl/
INDEX/SPLIT/2/\l_name_tl=A
IF_FIRST:A.3/\l_b_tl=
IF_FIRST:A/\l_b_tl=
INDEX:\l_ans_tl=A+3-1
ROUND:A+3-1/\l_ans_tl=

FAILURE '0'!='3'

Test __beanoves_index:nnN 3

INDEX:B/.2.4/\l_ans_tl/
INDEX/SPLIT/2/\l_name_tl=A
IF_FIRST:A.4/\l_b_tl=
IF_FIRST:A/\l_b_tl=
INDEX:\l_ans_tl=A+4-1
ROUND:A+4-1/\l_ans_tl=

FAILURE '0'!='5'

Test __beanoves_index:nnN 4

INDEX:B/.2.3.4/\l_ans_tl/
INDEX/SPLIT/2/\l_name_tl=B.2
INDEX/SPLIT/3/\l_name_tl=A
IF_FIRST:A.4/\l_b_tl=
IF_FIRST:A/\l_b_tl=
INDEX:\l_ans_tl=A+4-1
ROUND:A+4-1/\l_ans_tl=

FAILURE '0'!='5'

Test __beanoves_index:nnN 5

INDEX:B/.2.3.4/\l_ans_tl/
INDEX/SPLIT/2/\l_name_tl=B.2
INDEX/SPLIT/3/\l_name_tl=A
IF_FIRST:A.4/\l_b_tl=
IF_FIRST:A/\l_b_tl=
INDEX:\l_ans_tl=A+4-1
ROUND:A+4-1/\l_ans_tl=

FAILURE '0'!='25'

Test __beanoves_index:nnN 6

__beanoves_incr:nn
__beanoves_incr:nnN
__beanoves_incr:(VnN|VVN)

__beanoves_incr:nn {⟨*name*⟩} {⟨*offset*⟩} __beanoves_incr:nnN {⟨*name*⟩} {⟨*offset*⟩} ⟨*tl variable*⟩

Incrementthefreecounterpositionaccordingly.Whenrequested,puttheresultinthe⟨*tlvariable*⟩.T

```
634 \prg_new_conditional:Npnn \__beanoves_if_incr:nn #1 #2 { p, T, F, TF } {   IF_INCR:\tl_to_st
```

FREE:\l_ans_tl/X//
IF_FIRST:X/\l_ans_tl=

- **FAILURE ‘’!=‘123’**
- **Test** `\__beanoves_if_incr:nNTF` **1-a**

  COUNTER:X/\l_ans_tl=
  FREE:\l_ans_tl/X//
  IF_FIRST:X/\l_ans_tl=

- **FAILURE ‘’!=‘123’**
- **Test** `\__beanoves_if_incr:nNTF` **1-b**

  IF_INCR:X/-100
  FREE:\l_a_tl/X//
  IF_FIRST:X/\l_ans_tl=

- **FAILURE ‘’!=‘SUCCESS’**
- **Test** `\__beanoves_if_incr:nNTF`

  FREE:\l_ans_tl/X//
  IF_FIRST:X/\l_ans_tl=

- **FAILURE ‘’!=‘23’**
- **Test** `\__beanoves_if_incr:nNTF` **2-a**

  COUNTER:X/\l_ans_tl=
  FREE:\l_ans_tl/X//
  IF_FIRST:X/\l_ans_tl=

- **FAILURE ‘’!=‘123’**
- **Test** `\__beanoves_if_incr:nNTF` **2-b**

  FREE:\l_ans_tl/X//
  IF_LAST/X

- **FAILURE ‘’!=‘123’**
- **Test** `\__beanoves_if_incr:nNTF` **3-a**

  COUNTER:X/\l_ans_tl=
  FREE:\l_ans_tl/X//
  IF_LAST/X
  IF_FIRST:X/\l_a_tl=
  IF_LAST/X
  ROUND:1/\l_ans_tl=
  RETNUOC:X/\l_ans_tl=1

⛔ **FAILURE '1'!='123'**

⛔ **Test** `\__beanoves_if_incr:nNTF` **3-b**

IF_INCR:X/100
FREE:\l_a_tl/X//
IF_LAST/X

⛔ **FAILURE ''!='SUCCESS'**

⛔ **Test** `\__beanoves_if_incr:nNTF`

FREE:\l_ans_tl/X//
IF_LAST/X

⛔ **FAILURE '1'!='223'**

⛔ **Test** `\__beanoves_if_incr:nNTF` **4-a**

COUNTER:X/\l_ans_tl=
FREE:\l_ans_tl/X//
IF_LAST/X
IF_FIRST:X/\l_a_tl=
IF_LAST/X
ROUND:1/\l_ans_tl=
RETNUOC:X/\l_ans_tl=1

⛔ **FAILURE '1'!='123'**

⛔ **Test** `\__beanoves_if_range:nNTF` **4-b**

`\__beanoves_if_range_p:nN` ⋆ `\__beanoves_if_range_p:nN {⟨name⟩} ⟨tl variable⟩`
`\__beanoves_if_range:nNTF` ⋆ `\__beanoves_if_range:nNTF {⟨name⟩} ⟨tl variable⟩ {⟨true code⟩} {⟨false code⟩}`

Appendtherangeofthe⟨*name*⟩sliderangetothe⟨*tlvariable*⟩.Execute⟨*truecode*⟩whenthereisa⟨*ran*

```
\prg_new_conditional:Npnn \__beanoves_if_range:nN #1 #2 { p, T, F, TF } {   \bool_if:NTF \l_
```

`\__beanoves_range:nN` `\__beanoves_range:nN {⟨name⟩} ⟨tl variable⟩`
`\__beanoves_range:VN`

Appendtherangeofthe⟨*name*⟩sliderangetothe⟨*tlvariable*⟩.

```
\cs_new:Npn \__beanoves_range:nN #1 #2 {   \__beanoves_if_range:nNF { #1 } #2 {     \msg_err
```

IF_FIRST:S/\l_ans_tl=

⛔ **FAILURE ''!='1'**

⛔ **Test** `\__beanoves_range:nN` **0-a**

IF_FIRST:X/\l_ans_tl=

⛔ **FAILURE ''!='1'**

⛔ **Test __beanoves_range:nN 0-b**

IF_FIRST:X/\l_a_tl=
IF_LAST/X

⛔ **FAILURE ‘’!=‘1-’**

⛔ **Test __beanoves_range:nN 1**

IF_FIRST:X/\l_a_tl=
IF_LAST/X

⛔ **FAILURE ‘’!=‘-111’**

⛔ **Test __beanoves_range:nN 2**

IF_FIRST:X/\l_a_tl=
IF_LAST/X
S.1+X.last-(X.1)+1-1
RANGE:\l_b_tl=X.last-(X.1)+1

⛔ **FAILURE ‘’!=‘1-111’**

⛔ **Test __beanoves_range:nN 1**

S=1,L=11,E=111,X=S.1:L.1X1-113S=1,L=11,E=111,X=:L.1::E.1X101-1111

5.3.6 Evaluation

__beanoves_resolve:nnN
__beanoves_resolve:VVN
__beanoves_resolve:nnNN
__beanoves_resolve:VVNN

__beanoves_resolve:nnN {⟨*name*⟩} {⟨*path*⟩} ⟨*tl variable*⟩ __beanoves_resolve:nnN {⟨*name*⟩} {⟨*path*⟩} ⟨*tl name variable*⟩ ⟨*tl last variable*⟩

Resolvethe⟨*name*⟩and⟨*path*⟩intoakeythatisputintothe⟨*tlnamevariable*⟩.Inthesecondversion,th

```
\cs_new:Npn \__beanoves_resolve:nnN #1 #2 #3 {   \group_begin:   \tl_set:Nn \l_a_tl { #1 }
```

__beanoves_if_append_p:nN ⋆ __beanoves_if_append_p:nN {⟨*key*⟩} ⟨*tl variable*⟩
__beanoves_if_append_p:VN ⋆ __beanoves_if_append:nNTF {⟨*key*⟩} ⟨*tl variable*⟩ {⟨*true code*⟩} {⟨*false*
__beanoves_if_append:nNTF ⋆ *code*⟩}
__beanoves_if_append:VNTF ⋆

Evaluatesthe⟨*integerexpression*⟩,replacingallthenamedspecificationsbytheirstaticcounterpar
eval_query:nN,where⟨*integerexpression*⟩wasinitiallyenclosedin'|?(...)|'.Localvariables:

\l_ans_tl Tofeed⟨*tlvariable*⟩with.

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

\l_split_seq Thesequenceofcatchedquerygroupsandnonqueries.

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

\l__beanoves_split_int Istheindexofthenonqueries,beforeallthecatchedgroups.

(*End definition for* \l__beanoves_split_int.)

\l_name_tl Storagefor\l_split_seqitemsthatrepresentnames.

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

\l_path_tl Storagefor\l_split_seqitemsthatrepresentintegerpaths.

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

```
\prg_new_conditional:Npnn \__beanoves_if_append:nN #1 #2 { p, T, F, TF } {   IF_APPEND:\tl_t
```

Localvariables:

```
    \int_zero:N  \l__beanoves_split_int     \seq_clear:N \l_split_seq     \tl_clear:N  \l_na
```

Implementation:

```
    \regex_split:NnN \c__beanoves_split_regex { #1 } \l_split_seq     SPLIT_SEQ: / \seq_use:
```

\switch:nTF \switch:nTF {⟨*capture group number*⟩} {⟨*black code*⟩} {⟨*white code*⟩}

Helperfunctiontolocallysetthe||variabletothecapturedgroup⟨*capturegroupnumber*⟩andbranch

```
    \cs_set:Npn \switch:nNTF ##1 ##2 ##3 ##4 {       \tl_set:Nx ##2 {         \seq_item:Nn \
```

\prg_return_true:and\prg_return_false:areredefinedlocallytoclosethegroupan-
dreturnthepropervalue.

```
    \cs_set:Npn \prg_return_true: {       \exp_args:NNNV       \group_end:       \tl_put_rig
```

Mainloop.

```
    \bool_set_true:N \l__beanoves_continue_bool     \bool_while_do:Nn \l__beanoves_continue_
```

- Case++⟨*name*⟩⟨*integerpath*⟩.n.

```
        \switch:nNTF 2 \l_path_tl {            \__beanoves_resolve:VVN \l_name_tl \l_path
```

X.n

FAILURE ''!='123'

Test __beanoves_resolve:nnN 1

++X.n

FAILURE ‘’!=‘124’

Test __beanoves_resolve:nnN 2

X.n

FAILURE ‘’!=‘124’

Test __beanoves_resolve:nnN 3

- Cases⟨*name*⟩⟨*integerpath*⟩....

```
\tl_set:Nn \l_b_tl {                \switch:nNTF 4 \l_path_tl {                \
```

- Case...length.

```
\l_b_tl                \__beanoves_if_length:VNF \l_name_tl \l_ans_tl {
```

**********11:4

A.length

FAILURE ‘’!=‘4’

Test __beanoves_append:nN/1-length

**********11::12

A.length

FAILURE ‘’!=‘2’

Test __beanoves_append:nN/2-length

**********:2::12

A.length

FAILURE ‘’!=‘2’

Test __beanoves_append:nN/3-length

**********S:L,S=11,L=2,E=12

A.length

FAILURE ‘’!=‘2’

Test __beanoves_append:nN/4-length

**********S::E,S=11,L=2,E=12

A.length

FAILURE ‘’!=‘2’

Test __beanoves_append:nN/5-length

**********:L::E,S=11,L=2,E=12

A.length

FAILURE ‘’!=‘2’

Test __beanoves_append:nN/6-length

- Case...last.

```
\l_b_tl                \__beanoves_if_last:VNF \l_name_tl \l_ans_tl {
```

**********11:4

A.last

FAILURE ‘’!=‘14’

⛔ **Test __beanoves_append:nN/1-last**
********************************11::12**
A.last
⛔ **FAILURE ‘’!=‘12’**
⛔ **Test __beanoves_append:nN/2-last**
********************************:2::12**
A.last
⛔ **FAILURE ‘’!=‘12’**
⛔ **Test __beanoves_append:nN/3-last**
********************************S:L,S=11,L=2,E=12**
A.last
⛔ **FAILURE ‘’!=‘12’**
⛔ **Test __beanoves_append:nN/4-last**
********************************S::E,S=11,L=2,E=12**
A.last
⛔ **FAILURE ‘’!=‘12’**
⛔ **Test __beanoves_append:nN/5-last**
********************************:L::E,S=11,L=2,E=12**
A.last
⛔ **FAILURE ‘’!=‘12’**
⛔ **Test __beanoves_append:nN/6-last**

- **Case...next.**

```
648                     \l_b_tl                    \__beanoves_if_next:VNF \l_name_tl \l_ans_tl {
```

********************************11:4**
A.next
⛔ **FAILURE ‘’!=‘15’**
⛔ **Test __beanoves_append:nN/1-next**
********************************11::12**
A.next
⛔ **FAILURE ‘’!=‘13’**
⛔ **Test __beanoves_append:nN/2-next**
********************************:2::12**
A.next
⛔ **FAILURE ‘’!=‘13’**
⛔ **Test __beanoves_append:nN/3-next**
********************************S:L,S=11,L=2,E=12**
A.next
⛔ **FAILURE ‘’!=‘13’**
⛔ **Test __beanoves_append:nN/4-next**
********************************S::E,S=11,L=2,E=12**
A.next

⛔ **FAILURE ‘’!=‘13’**

⛔ **Test __beanoves_append:nN/5-next**

******************************:L::E,S=11,L=2,E=12**

A.next

⛔ **FAILURE ‘’!=‘13’**

⛔ **Test __beanoves_append:nN/6-next**

- **Case...range.**

```
649                     \l_b_tl                     \__beanoves_if_range:VNF \l_name_tl \l_ans_t
```

**************************************11:4**

A.range

⛔ **FAILURE ‘’!=‘11-14’**

⛔ **Test __beanoves_append:nN/1-range**

**************************************11::12**

A.range

⛔ **FAILURE ‘’!=‘11-12’**

⛔ **Test __beanoves_append:nN/2-range**

********************************:2::12**

A.range

⛔ **FAILURE ‘’!=‘11-12’**

⛔ **Test __beanoves_append:nN/3-range**

************************************S:L,S=11,L=2,E=12**

A.range

⛔ **FAILURE ‘’!=‘11-12’**

⛔ **Test __beanoves_append:nN/4-range**

************************************S::E,S=11,L=2,E=12**

A.range

⛔ **FAILURE ‘’!=‘11-12’**

⛔ **Test __beanoves_append:nN/5-range**

************************************:L::E,S=11,L=2,E=12**

A.range

⛔ **FAILURE ‘’!=‘11-12’**

⛔ **Test __beanoves_append:nN/6-range**

```
650                     } {                     \switch:nNTF 9 \l_a_tl {
```

- **Case...n.**

```
651                     \l_b_tl                     \switch:nNTF { 10 } \l_a_tl {
```

- **Case...+=⟨*integer*⟩.**

```
652                     \__beanoves_if_incr:VVNF \l_name_tl \l_a_tl \l_ans_tl {
```

********** A=1
A.n
FAILURE ‘’!=‘1’
Test __beanoves_append:nN/1
********** A.1=101
A.1.n
FAILURE ‘’!=‘101’
Test __beanoves_append:nN/2
********** A=11
A.n+=100
FAILURE ‘’!=‘111’
Test __beanoves_append:nN/3
A.n+=100A.n+=-100
FAILURE ‘’!=‘11’
Test __beanoves_append:nN/4/+=

```
                    } {
```

- Case...⟨*integerpath*⟩.

```
                      \switch:nNTF 4 \l_path_tl {                    \exp_args:NVV
```

A.10
FAILURE ‘’!=‘10’
Test __beanoves_append:nN/1
A.10
FAILURE ‘’!=‘110’
Test __beanoves_append:nN/2
A.1
FAILURE ‘’!=‘101’
Test __beanoves_append:nN/3
A.1.10
FAILURE ‘’!=‘110’
Test __beanoves_append:nN/4

```
                    }                 }                 }              }            }
```

Noname.

```
          }         }        \int_add:Nn \l__beanoves_split_int { 11 }        \tl_put_righ
```

__beanoves_eval_query:nN __beanoves_eval_query:nN {⟨*overlay query*⟩} ⟨*seq variable*⟩

Evaluatesthesingle⟨*overlayquery*⟩,whichisexpectedtocontainnocomma.Extractarangespecific

\l_a_tl Storageforthefirstindexofarange.

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

\l_b_tl Storageforthelastindexofarange,oritslength.

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

\c__beanoves_A_cln_Z_regex Usedtoparsesliderangeoverlayspecifications.Nextarethecapturegroups.

(*End definition for* \c__beanoves_A_cln_Z_regex.)

```
\regex_const:Nn \c__beanoves_A_cln_Z_regex {   \A \s* (?:
```

- 2:⟨*first*⟩

```
        ( [^:]* ) \s* :
```

- 3:secondoptionalcolon

```
        (:)? \s*
```

- 4:⟨*length*⟩

```
        ( [^:]* )
```

- 5:standalone⟨*first*⟩

```
      | ( [^:]+ )   ) \s* \Z }
```

```
\cs_new:Npn \__beanoves_eval_query:nN #1 #2 {   EVAL_QUERY:\tl_to_str:n{#1}/\string#2=:\tl_t
```

\switch:nNTF \switch:nNTF {⟨*capture group number*⟩} ⟨*tl variable*⟩ {⟨*black code*⟩} {⟨*white code*⟩}

Helperfunctiontolocallysetthe⟨*tlvariable*⟩tothecapturedgroup⟨*capturegroupnumber*⟩andbran

```
    \cs_set:Npn \switch:nNTF ##1 ##2 ##3 ##4 {       SWITCH:##1/       \tl_set:Nx ##2 {
```

Singleexpression

```
        \bool_set_false:N \l__beanoves_no_range_bool       \__beanoves_if_append:VNTF \l_a_tl
```

⟨*first*⟩::⟨*last*⟩range

```
            \__beanoves_if_append:VNTF \l_a_tl \l_ans_tl {               \tl_put_right:Nn \l
```

⟨*first*⟩:⟨*length*⟩range

```
            \__beanoves_if_append:VNTF \l_a_tl \l_ans_tl {               \tl_put_right:Nx \l
```

⟨*first*⟩:and⟨*first*⟩::range

```
          \__beanoves_if_append:VNTF \l_a_tl \l_ans_tl {             \tl_put_right:Nn \l_ans
```

::⟨*last*⟩range

```
            \tl_put_right:Nn \l_ans_tl { - }             \__beanoves_if_append:VNTF \l_a_tl
```

:or::range

```
\seq_put_right:Nn #2 { - }        }      }     }   } {
```

Error

```
\msg_error:nnn { beanoves } { :n } { Syntax~error:~#1 }   } }
```

__beanoves_if_eval_query_p:nN ⋆
__beanoves_if_eval_query:nNTF ⋆

__beanoves_if_eval_query_p:nN {⟨*overlay query*⟩} ⟨*tl variable*⟩
__beanoves_if_eval_query:nNTF {⟨*overlay query*⟩} ⟨*tl variable*⟩ {⟨*true code*⟩} {⟨*false code*⟩}

Evaluatesthesingle⟨*overlayquery*⟩,whichisexpectedtocontainnocomma.Extractarangespecific

\l_a_tl Storageforthefirstindexofarange.

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

\l_b_tl Storageforthelastindexofarange,oritslength.

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

\c__beanoves_A_cln_Z_regex Usedtoparsesliderangeoverlayspecifications.

(*End definition for* \c__beanoves_A_cln_Z_regex.)

```
\prg_new_conditional:Npnn \__beanoves_if_eval_query:nN #1 #2 { p, T, F, TF } {   EVAL_QUERY:
```

\switch:nNTF

\switch:nNTF {⟨*capture group number*⟩} ⟨*tl variable*⟩ {⟨*black code*⟩} {⟨*white code*⟩}

Helperfunctiontolocallysetthe⟨*tlvariable*⟩tothecapturedgroup⟨*capturegroupnumber*⟩andbran

```
\cs_set:Npn \switch:nNTF ##1 ##2 ##3 ##4 {       SWITCH:##1/       \tl_set:Nx ##2 {
```

Singleexpression

```
\bool_set_false:N \l__beanoves_no_range_bool       \__beanoves_if_append:VNTF \l_a_tl
```

⟨*first*⟩::⟨*last*⟩range

```
\__beanoves_if_append:VNTF \l_a_tl #2 {               \tl_put_right:Nn #2 { - }
```

⟨*first*⟩:⟨*length*⟩range

```
\__beanoves_if_append:VNTF \l_a_tl #2 {               \tl_put_right:Nx #2 { - }
```

⟨*first*⟩:and⟨*first*⟩::range

```
\__beanoves_if_append:VNTF \l_a_tl #2 {             \tl_put_right:Nn #2 { - }
```

::⟨*last*⟩range

```
\tl_put_right:Nn #2 { - }             \__beanoves_if_append:VNTF \l_a_tl #2 {
```

:or::range

```
\seq_put_right:Nn #2 { - }        }      }     }   } {
```

Error

```
679     \msg_error:nnn { beanoves } { :n } { Syntax~error:~#1 }   } }
```

****
A=100
EVAL_QUERY:1/\l_tmpa_tl=,
EQ:\l_match_seq/1////1/
SWITCH:5/\l_a_tl/1/
IF_APPEND:1/\l_tmpa_tl
APPEND:1/\l_tmpa_tl
SPLIT_SEQ:/1/
ROUND:1/\l_ans_tl=
DENPPA:1/\l_tmpa_tl=1
****
A=100
EVAL_QUERY:1+1/\l_tmpa_tl=,
EQ:\l_match_seq/1+1////1+1/
SWITCH:5/\l_a_tl/1+1/
IF_APPEND:1+1/\l_tmpa_tl
APPEND:1+1/\l_tmpa_tl
SPLIT_SEQ:/1+1/
ROUND:1+1/\l_ans_tl=
DENPPA:1+1/\l_tmpa_tl=2

`\__beanoves_eval:nN` `\__beanoves_eval:nN {⟨overlay query list⟩} ⟨tl variable⟩`

Thisiscalledbythe*namedoverlayspecifications*scanner.Evaluatesthecommaseparatedlistof⟨*ov*
`eval_query:nN`,thenappendtheresulttotherightofthe⟨*tlvariable*⟩.Thisisexecutedwithinalocalgr

`\l_query_seq` Storageforasequenceof⟨*query*⟩'sobtainedbysplittingacommaseparatedlist.

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

`\l_ans_seq` Storageoftheevaluatedresult.

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

`\c__beanoves_comma_regex` Usedtoparsesliderangeoverlayspecifications.

```
680 \regex_const:Nn \c__beanoves_comma_regex { \s* , \s* }
```

(*End definition for* `\c__beanoves_comma_regex`.)

Noothervariableisused.

```
681 \cs_new:Npn \__beanoves_eval:nN #1 #2 {   EVAL:\tl_to_str:n{#1}/\string#2=\tl_to_str:V #2\\
```

Localvariablesdeclaration

```
682   \seq_clear:N \l_ans_seq
```

Inthismainevaluationstep,weevaluatetheintegerexpressionandputtheresultinavariablewhichcontentwillbecopiedafterthegroupisclosed.Weauthorizecommaseparatedexpressionsand⟨*first*⟩::⟨*last*⟩rangeexpressionsaswell.Wefirstsplittheexpressionaroundco
`query_seq`.

```
683   \regex_split:NnN \c__beanoves_comma_regex { #1 } \l_query_seq
```

Theneachcomponentisevaluatedandtheresultisstoredin\l_ans_seqthatwehaveclear-beforeuse.

```
\seq_map_inline:Nn \l_query_seq {     \tl_clear:N \l_ans_tl     \__beanoves_if_eval_query:
```

Wehavemanagedallthecommaseparatedcomponents,wecollectthembackandappendthemto⟨*tl*

```
\exp_args:NNNx   \group_end:   \tl_put_right:Nn #2 { \seq_use:Nn \l_ans_seq , } } \cs_gene
```

\BeanovesEval \BeanovesEval [⟨*tl variable*⟩] {⟨*overlay queries*⟩}

⟨*overlayqueries*⟩istheargumentof|?(...)|instructions.Thisisacommaseparatedlistofsingle⟨*over*
eval:nNwithinagroup.\l_ans_tlisusedlocallytostoretheresult.

```
\NewExpandableDocumentCommand \BeanovesEval { s o m } {   \group_begin:   \tl_clear:N \l_ans
```

5.3.7 Resetingslideranges

\BeanovesReset \beanovesReset [⟨*first value*⟩] {⟨*Slide list name*⟩}

```
\NewDocumentCommand \BeanovesReset { O{1} m } {   \__beanoves_reset:nn { #1 } { #2 }   \igno
```

Forwardsto__beanoves_reset:nn.

__beanoves_reset:nn __beanoves_reset:nn {⟨*first value*⟩} {⟨*slide list name*⟩}

Resetthecountertothegiven⟨*firstvalue*⟩.Cleanthecachedvaluesalso(notusefull).

```
\cs_new:Npn \__beanoves_reset:nn #1 #2 {   \bool_if:nTF {     \__beanoves_if_in_p:n { #2/A }
\makeatother \ExplSyntaxOff
%</package>
```