beamer named overlay specifications with beanoves

Jérôme Laurens

v1.0 2024/01/11

Abstract

This package allows the management of multiple named overlay specifications in beamer documents. Named overlay specifications are very handy both during edition and to manage complex and variable beamer overlay specifications. In particular, they allow to replace raw numbers in beamer `<...>` overlay specifications by logical identifiers. Demonstration files are available for download as part of the development repository. This is a solution to this `latex.org` forum query.

Contents

1 Installation **2**
1.1 Package manager . . . 2
1.2 Manual installation . . . 2
1.3 Usage . . . 3

2 Minimal example **3**

3 Named overlay sets **4**
3.1 Presentation . . . 4
3.2 Named overlay reference . . . 4
3.3 Defining named overlay sets . . . 4
3.3.1 Basic specifiers . . . 5
3.3.2 List specifiers . . . 5

4 Resolution of `?(...)` query expressions **6**
4.1 Number and range overlay queries . . . 6
4.2 Counters . . . 7
4.3 Dotted paths . . . 9
4.4 The `beamerpauses` counter . . . 9
4.5 Multiple queries . . . 9
4.6 Frame id . . . 9
4.7 Resolution command . . . 10

5 Support **10**

1 Installation

1.1 Package manager

When not already available, beanoves package may be installed using a TEX distribution's package manager, either from the graphical user interface, or with the relevant command:

- TEX Live: `tlmgr install beanoves`
- MiKTEX: `mpm --admin --install=beanoves`

This should install files beanoves.sty and its debug version beanoves-debug.sty as well as beanoves-doc.pdf documentation.

1.2 Manual installation

The beanoves source files are available from the source repository. They can also be fetched from the CTAN repository.

1.3 Usage

The beanoves package is imported by putting `\RequirePackage{beanoves}` in the preamble of a LATEX document that uses the beamer class. Should the package cause problems, its features can be temporarily deactivated with simple commands `\BeanovesOff` and `\BeanovesOn`.

2 Minimal example

The LATEX document below is a contrived example to show how the `beamer` overlay specifications have been extended. More demonstration files are available from the beanoves source repository.

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

On line 4, we use the `\Beanoves` command to declare *named overlay sets*. On line 5, we declare an overlay set named 'A', which is a range starting at slide 1 and ending at slide 4. On line 12, the extended *named overlay specification* `?(A.1)` stands for 1 because 1 is the first index of the overlay set named A. On line 15, `?(A.2)` stands for 2 whereas on line 18, `?(A.next)` stands for 5. On line 6, we declare a second overlay set named 'B', starting after the 3 slides of 'A' namely 4. Its length is 3 meaning that its last slide number is 6, thus each `?(B.last)` is replaced by 6. The next slide number after slide range 'B' is 7 which is also the start of the third slide range due to line 7.

3 Named overlay sets

3.1 Presentation

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

3.2 Named overlay reference

`A.1`, `C.2` are *named overlay references*, as well as `A` and `Y!C.2`. More precisely, they are string identifiers, each one referencing a well defined static integer or range to be used in beamer overlay specifications. They can take one of the next forms.

⟨***short name***⟩ : like `A` and `C`,

⟨***frame id***⟩`!`⟨***short name***⟩ : denoted by *qualified names*, like `X!A` and `Y!C`.

⟨***short name***⟩⟨***dotted path***⟩ : denoted by *full names* like `A.1` and `C.2`,

⟨***frame id***⟩`!`⟨***short name***⟩⟨***dotted path***⟩ : denoted by *qualified full names* like `X!A.1` and `Y!C.2`.

The *short names* and *frame ids* are alphanumerical case sensitive identifiers, with possible underscores but with no space nor leading digit. Unicode symbols above `U+00A0` are allowed if the underlying TEX engine supports it.

The *dotted path* is a string `.`⟨c_1⟩`.`⟨c_2⟩`....`⟨c_j⟩. Each component ⟨c_i⟩ denotes a ⟨***short name***⟩ or a decimal integer. The *dotted path* can be empty for which j is 0.

Identifiers consisting only of lowercase letters may have special meaning as detailed below. This includes components ⟨*c*⟩s, unless explicitly documented like for "`n`".

The mapping from *named overlay references* to integers is defined at the global TEX level to allow its use in `\begin{frame}<...>` and to share the same overlay sets between different frames. Hence the *frame id* due to the need to possibly target a particular frame.

3.3 Defining named overlay sets

In order to define *named overlay sets*, we can either execute the next `\Beanoves` command before a beamer frame environment, or use the new `beanoves` option of this environment.

`\Beanoves`
`\Beanoves*`

`\Beanoves{`⟨ref_1⟩`=`⟨$spec_1$⟩`, `⟨ref_2⟩`=`⟨$spec_2$⟩`,..., `⟨ref_j⟩`=`⟨$spec_j$⟩`}`

`beanoves`

`beanoves = {`⟨ref_1⟩`[=`⟨$spec_1$⟩`], `⟨ref_2⟩`[=`⟨$spec_2$⟩`],..., `⟨ref_j⟩`[=`⟨$spec_j$⟩`]}`

Each ⟨ref_i⟩ key is a *named overlay reference* whereas each ⟨***spec***⟩ value is an *overlay set specifier*. When the same ⟨***ref***⟩ key is used multiple times, only the last one is taken into account.

Notice that ⟨ref_i⟩`=1` can be shortened to ⟨ref_i⟩. The `\Beanoves` arguments take precedence over both the `\Beanoves*` arguments and the `beanoves` options. This allows to provide an overlay name only when not already defined, which is helpfull when the very same frame source is included multiple times in different contexts.

3.3.1 Basic specifiers

In the possible values for ⟨***spec***⟩ hereafter, ⟨***value***⟩, ⟨***first***⟩, ⟨***length***⟩ and ⟨***last***⟩ are numerical expression (with algebraic operators +, −, ...) possibly involving any *named overlay reference* defined above.

⟨***value***⟩**,** the simple *value specifiers* for the whole signed integers set. If only the ⟨***key***⟩ is provided, the ⟨***value***⟩ defaults to 1.

⟨***first***⟩**:** **and** ⟨***first***⟩**::,** for the infinite range of signed integers starting at and including ⟨***first***⟩.

:⟨***last***⟩**,** for the infinite range of signed integers ending at and including ⟨***last***⟩.

⟨*first*⟩`:`⟨*last*⟩, ⟨*first*⟩`::`⟨*length*⟩, `:`⟨*last*⟩`::`⟨*length*⟩, `::`⟨*length*⟩`:`⟨*last*⟩, are variants for the finite range of signed integers starting at and including ⟨*first*⟩, ending at and including ⟨*last*⟩. At least one of ⟨*first*⟩ or ⟨*last*⟩ must be provided. We always have ⟨*first*⟩ + ⟨*length*⟩ = ⟨*last*⟩ + 1.

When performed at the document level, the `\Beanoves` command starts by cleaning what was set by previous calls. When performed inside LaTeX environments, each new call cumulates with the previous one. Notice that the argument of this function can contain macros: they will be exhaustively expanded at resolution time[1].

3.3.2 List specifiers

Also possible values are *list specifiers* which are comma separated lists of ⟨*path*⟩=⟨*spec*⟩ definitions. The definition

⟨*ref*⟩`={{`⟨$path_1$⟩=⟨$spec_1$⟩`,` ⟨$path_2$⟩=⟨$spec_2$⟩`,...,` ⟨$path_j$⟩=⟨$spec_j$⟩`}}`

removes previous ⟨*ref*⟩ index definitions, and executes

⟨*ref*⟩`.`⟨$path_1$⟩=⟨$spec_1$⟩`,`
⟨*ref*⟩`.`⟨$path_2$⟩=⟨$spec_2$⟩`,`
`...,`
⟨*ref*⟩`.`⟨$path_j$⟩=⟨$spec_j$⟩`.`

The rules above can apply individually to each line. The ⟨*ref*⟩ counter defined below is left unmodified.

To support an array like syntax, we can omit the ⟨*path*⟩ key and only give the ⟨*spec*⟩ value. Each missing ⟨*path*⟩ key is replaced by the smallest index ⟨*j*⟩ such that ⟨*ref*⟩`.`⟨*j*⟩ is not already defined and ⟨*j*⟩ ≥ 1.

Notice that you can replace each opening pair `{{` by a single `[` and each closing pair `}}` by a single `]`. Anyway, delimiters should be properly balanced.

4 Resolution of `?(...)` query expressions

This is the key feature of the beanoves package, extending beamer *overlay specifications* normally 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 resolved static counterpart. The overall result is finally forwarded to the beamer class.

The ⟨*queries*⟩ argument is a comma separated list of individual ⟨*query*⟩'s processed from left to right as explained below. Notice that nesting a `?(...)` query expressions inside another query expression is not supported.

The named overlay sets defined above are queried for integer numerical values that will be passed to beamer. Turning an *overlay query* into the static expression it represents, as when above `?(A.1)` was replaced by `1`, is denoted by *overlay query resolution* or simply *resolution*. The process starts by replacing any *query reference* by its value as explained below until obtaining numerical expressions that are evaluated and finally rounded to the nearest integer to feed beamer with either a range or a number. When the *query reference* is a previously declared ⟨*ref*⟩, like `X` after `X=1`, it is simply replaced by the corresponding declared ⟨*value*⟩. Otherwise, we use *implicit overlay queries* and their *resolution rules* depending on the definition of the named overlay set. Here ⟨*i*⟩ denotes a signed integer

[1] Precision is needed for the exact time when the expansion occurs.

whereas ⟨*value*⟩, ⟨*first*⟩, ⟨*last*⟩ and ⟨*length*⟩ stand for raw integers or more general numerical expressions. We assume that ⟨*first*⟩ ≤ ⟨*last*⟩ and ⟨*length*⟩ ≥ 0.

Resolution occurs only when required and the result is cached for performance reason.

4.1 Number and range overlay queries

⟨*ref*⟩ `=` ⟨*value*⟩ For an unlimited range

overlay query	resolution
⟨*ref*⟩`.1`	⟨*value*⟩
⟨*ref*⟩`.2`	⟨*value*⟩ + 1
⟨*ref*⟩`.`⟨*i*⟩	⟨*value*⟩ + ⟨*i*⟩ − 1

⟨*ref*⟩ `=` ⟨*first*⟩`:` as well as ⟨*first*⟩`::`. For a range limited from below:

overlay query	resolution
⟨*ref*⟩`.1`	⟨*first*⟩
⟨*ref*⟩`.2`	⟨*first*⟩ + 1
⟨*ref*⟩`.`⟨*i*⟩	⟨*first*⟩ + ⟨*i*⟩ − 1
⟨*ref*⟩`.previous`	⟨*first*⟩ − 1
⟨*ref*⟩`.first`	⟨*first*⟩

Notice that ⟨*ref*⟩`.previous` and ⟨*ref*⟩`.0` are most of the time synonyms.

⟨*ref*⟩ `=` `:`⟨*last*⟩ For a range limited from above:

overlay query	resolution
⟨*ref*⟩`.1`	⟨*last*⟩
⟨*ref*⟩`.0`	⟨*last*⟩ − 1
⟨*ref*⟩`.`⟨*i*⟩	⟨*last*⟩ + ⟨*i*⟩ − 1
⟨*ref*⟩`.last`	⟨*last*⟩
⟨*ref*⟩`.next`	⟨*last*⟩ + 1

⟨*ref*⟩ `=` ⟨*first*⟩`:`⟨*last*⟩ as well as variants ⟨*first*⟩`::`⟨*length*⟩, `::`⟨*length*⟩`:`⟨*last*⟩ or `:`⟨*last*⟩`::`⟨*length*⟩, which are equivalent provided ⟨*first*⟩+⟨*length*⟩ = ⟨*last*⟩+1.

For a range limited from both above and below:

overlay query	resolution
⟨*ref*⟩`.1`	⟨*first*⟩
⟨*ref*⟩`.2`	⟨*first*⟩ + 1
⟨*ref*⟩`.`⟨*i*⟩	⟨*first*⟩ + ⟨*i*⟩ − 1
⟨*ref*⟩`.previous`	⟨*first*⟩ − 1
⟨*ref*⟩`.first`	⟨*first*⟩
⟨*ref*⟩`.last`	⟨*last*⟩
⟨*ref*⟩`.next`	⟨*last*⟩ + 1
⟨*ref*⟩`.length`	⟨*length*⟩
⟨*ref*⟩`.range`	max(0, ⟨*first*⟩) `''-''` max(0, ⟨*last*⟩)

Notice that the resolution of ⟨*ref*⟩`.range` is not an algebraic difference, and negative integers do not make sense there while in beamer context.

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

```
\Beanoves {
A = 3:8, % or similarly A = 3::6, A = ::6:8 and A = :8::6
}
\begin{frame} {Frame \insertframenumber} {Slide \insertslidenumber}
\ttfamily
\BeanovesResolve[show](A.1)        == 3,
\BeanovesResolve[show](A.-1)       == 1,
\BeanovesResolve[show](A.previous) == 2,
\BeanovesResolve[show](A.first)    == 3,
\BeanovesResolve[show](A.last)     == 8,
\BeanovesResolve[show](A.next)     == 9,
\BeanovesResolve[show](A.length)   == 6,
\BeanovesResolve[show](A.range)    == 3-8,
\end{frame}
```

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 starting value and a length, and not overriden.

4.2 Counters

Each named overlay set defined has a dedicated value counter which is some kind of integer variable that can be used and incremented. A standalone ⟨*name*⟩ *overlay query* is resolved into the position of this value counter. For each frame, this variable is initialized to the first available resolution amongst ⟨*value*⟩, ⟨*name*⟩`.first`, ⟨*name*⟩`.1` or ⟨*name*⟩`.last`. If none is available, the counter is initialized to 1.

Additionnaly, resolution rules are provided for dedicated *overlay queries*:

⟨***name***⟩**=**⟨***integer expression***⟩**,** resolve ⟨*integer expression*⟩ into ⟨*integer*⟩, set the value counter to ⟨*integer*⟩ and use the new position. Here ⟨*integer expression*⟩ is the longest character sequence with no space[2].

⟨***name***⟩**+=**⟨***integer expression***⟩**,** resolve ⟨*integer expression*⟩ into ⟨*integer*⟩, advance the value counter by ⟨*integer*⟩ and use the new position.

++⟨***name***⟩**,** advance the value counter for ⟨*name*⟩ by 1 and use the new position.

⟨***name***⟩**++,** use the actual position and advance the value counter for ⟨*name*⟩ by 1.

For each named overlay set defined, we also have an implicit index counter always starting at 1, its actual value is an integer denoted ⟨*n*⟩ in the sequel. The ⟨*name*⟩`.n` *named index reference* is resolved into ⟨*name*⟩`.`⟨*n*⟩, which in turn is resolved according to the preceding rules.

We have resolution rules as well for the *named index references*:

[2]The parser for algebraic expression is very rudimentary.

⟨*name*⟩`.n=`⟨*integer expression*⟩**,** resolve ⟨*integer expression*⟩ into ⟨*integer*⟩, set the implicit index counter associate to ⟨*name*⟩ to ⟨*integer*⟩ and use the resolution of ⟨*name*⟩`.n`.

Here again, ⟨*integer expression*⟩ denotes the longest character sequence with no space.

⟨*name*⟩`.n+=`⟨*integer expression*⟩**,** resolve ⟨*integer expression*⟩ into ⟨*integer*⟩, advance the implicit index counter associate to ⟨*name*⟩ by ⟨*integer*⟩ and use the resolution of ⟨*name*⟩`.n`.

⟨*name*⟩`.++n`**,** `++`⟨*name*⟩`.n`**,** advance the implicit index counter associate to ⟨*key*⟩ by 1 and use the resolution of ⟨*name*⟩`.n`,

⟨*name*⟩`.n++`**,** use the resolution of ⟨*name*⟩`.n` and increment the implicit index counter associate to ⟨*name*⟩ by 1.

In order to decrement a counter, one can increment with a negative value, no dedicated syntax is provided yet.

These counters are reset to their default value for each new frame, which is 1 for the ⟨*name*⟩`.n` counter, and whichever ⟨*name*⟩ first or last value is defined for the ⟨*name*⟩ counter. Sometimes, resetting the counter manually is necessary, for example when managing tikz overlay material.

`\BeanovesReset` `\BeanovesReset [`⟨*options*⟩`] {`⟨ref_1⟩`=`⟨$spec_1$⟩`,` ⟨ref_2⟩`=`⟨$spec_2$⟩`,...,` ⟨ref_j⟩`=`⟨$spec_j$⟩`}`

This command is very similar to `\Beanoves`, except that a standalone ⟨ref_i⟩ resets the counter to its default value and that it is meant to be used inside a `frame` environment. When the `all` option is provided, some internals that were cached for performance reasons are cleared.

4.3 Dotted paths

In previous overlay queries, ⟨*name*⟩ can be formally replaced by ⟨*name*⟩`.`⟨c_1⟩`.`⟨c_2⟩`...`⟨c_j⟩. If it does not correspond to a definition or an assignment, the longest qualified full name ⟨*name*⟩`.`⟨c_1⟩`.`⟨c_2⟩`...`⟨c_k⟩ where 0≤k≤j is first replaced by its definition ⟨*name'*⟩`.`⟨c'_1⟩`...`⟨c'_l⟩ if any and then the modified overlay query is resolved with preceding rules as well as this one. For example, with `\Beanoves{A.B=D, D.C=E}`, `A.B.C` is resolved like `E`. Inside a `frame` environment, when the instruction `\Beanoves{`⟨*ref*⟩`=pauses}` is executed, it saves the current value of the beamer pauses counter into the ⟨*ref*⟩ counter. Later on, `?(`⟨*ref*⟩`)` can refer to this value.

4.4 The `beamerpauses` counter

While inside a `frame` environment, it is possible to save the current value of the `beamerpauses` counter that controls whether elements should appear on the current slide. For that, we can execute one of `\Beanoves{`⟨*ref*⟩`=pauses}` or in a query `?(...(`⟨*ref*⟩`=pauses)...)`. Then later on, we can use `?(...`⟨*ref*⟩`...)` to refer to this saved value in the same frame[3]. Next frame source is an example of usage.

[3] See stackexchange for an alternative that needs at least two passes.

```
1 \begin{frame}
2 \visible<+->{A}\\
3 \visible<+->{B\Beanoves{afterB=pauses}}\\
4 \visible<+->{C}\\
5 \visible<?(afterB)>{other C}\\
6 \visible<?(afterB.previous)>{other B}\\
7 \end{frame}
```

"A" first appears on slide 1, "B" on slide 2 and "C" on slide 3. On line 2, `afterB` takes the value of the `beamerpauses` counter once updated, *id est* 3. "B" and "other B" as well as "C" and "other C" appear at the same time.

4.5 Multiple queries

It is possible to replace the comma separated list `?(`⟨*query*$_1$⟩`),...?(`⟨*query*$_j$⟩`)` with the shorter `?(`⟨*query*$_1$⟩`,...`⟨*query*$_j$⟩`)`.

4.6 Frame id

Except for very special situations, the *frame ids* can be left unspecified. When no *frame id* was explicitly provided, beanoves uses the *last frame id*. At the beginning of each frame, the *last frame id* is set to the *frame id* of the current frame, which is denoted *current frame id* and defaults to `?`. Then it gets updated after each named reference resolution. For example, the first time `A.1` reference is resolved within a given frame, it is first translated to ⟨*current frame id*⟩`!A.1`, but when used just after `Y!C.2`, for example, it becomes a shortcut to `Y!A.1` because the *last frame id* is then `Y`.

In order to set the *frame id* of the current frame to ⟨*frame id*⟩, use the new `beanoves id` option of the beamer frame environment.

`beanoves id` `beanoves id=`⟨*frame id*⟩`,`

We can use the same *frame id* for different frames to share named overlay sets.

4.7 Resolution command

`\BeanovesResolve` `\BeanovesResolve [`⟨*setup*⟩`] {`⟨*overlay queries*⟩`}`

This function resolves the ⟨*overlay queries*⟩, which are like the argument of `?(...)` instructions: a comma separated list of single ⟨*overlay query*⟩'s. The optional ⟨*setup*⟩ is a key–value:

`show` the result is left into the input stream

`in:N=`⟨*command*⟩ the result is stored into ⟨*command*⟩.

5 Support

See the source repository. One can report issues there.