Documentation of the `beamertools` package

Daniel Lohmann

2013-12-13

Purpose of the package

- The **`beamertools`** package provides a convenient interface to certain extensions and patches I have developed for my **`beamer`** presentations, especially the lecture slides for (G)SPiC and BS
- I created this package after figuring out, that
  - my **`preamble.tex`** files become way too long, way too redundant and way too complicated
  - I found certain repeating code patterns in my lecture presentations that could be shortened quite a bit by better abstractions
  - I always wanted to write an own LaTeX package :-)

Package loading and options

- Package options are processed with `pgfkeys`
  - Example: `\usepackage[autonotes,notikz]{beamertools}`

- The following options are available (sorry, no real docu yet):

```
\pgfset{
  /bt/.cd,
  framesintoc/.is if=btFramesInToC,                              % put
    frame titles as level 3 element in ToC
  framesinpdftoc/.is if=btFramesInPDFToC,                        % put
    frame titles as level 3 element in PDF ToC
  autonotes/.is if=btAutoNotes,                                  % add
    empty note to every slide
  physicalpagesinpdftoc/.is if=btPhysicalPagesInPDFToC,          % use
    physical page numbers (instead of labels) in PDF ToC
  nolistings/.is if=btNoListings,                                % do not
    include listing support (and related packages)
  noshortcuts/.is if=btNoShortcuts,                              % do not
    include shortcut macros (\bi \ii \ei and so on)
```

Package loading and options

- Package options are processed with `pgfkeys`
  - Example: `\usepackage[autonotes,notikz]{beamertools}`
  - All `pgfkeys` features (e.g., styles) can be employed:

    `\usepackage[spic]{beamertools}`
- The following styles are available (sorry, no real docu yet):

```
    {\pgfkeys {\pgfkeyscurrentpath /.code=\pgfkeysalso {#1}}}%
}}

% enable all package features (useful for debugging)
\btset{everything/.style={framesintoc, framesinpdftoc, autonotes,
  physicalpagesinpdftoc, woschblocks}}

% if the name of this style does not mean anything to you then just don't
  care
```

Shortcuts for List Environments

Shortcuts for the `itemize` environment: `\bi` … `\ii` … `\ei`

```
\bi
  \ii<+-> Level 1
    \bi
      \ii Level 2
    \ei
  \ii<+-> Level 1 again
\ei
```

- Level 1
  - Level 2

Variants to skip one or two levels (for compact lists):

```
\bii
  \ii This is a level 2 item
  \ii This is a level 2 item
\eii
\biii
  \ii This is a level 3 item
  \ii This is a level 3 item
\eiii
```

- This is a level 2 item
- This is a level 2 item

  – This is a level 3 item
  – This is a level 3 item

Variants for advantage/disadvantage lists (easy to redefine):

```
\bii
  \iiad This is an advantage
  \iida This is a disadvantage
\eii
```

+ This is an advantage
– This is a disadvantage

Shortcuts for List Environments

Shortcuts for the `itemize` environment: `\bi` … `\ii` … `\ei`

```
\bi
  \ii<+-> Level 1
    \bi
      \ii Level 2
    \ei
  \ii<+-> Level 1 again
\ei
```

- Level 1
    - Level 2
- Level 1 again

Variants to skip one or two levels (for compact lists):

```
\bii
  \ii This is a level 2 item
  \ii This is a level 2 item
\eii
\biii
  \ii This is a level 3 item
  \ii This is a level 3 item
\eiii
```

- This is a level 2 item
- This is a level 2 item

- This is a level 3 item
- This is a level 3 item

Variants for advantage/disadvantage lists (easy to redefine):

```
\bii
  \iiad This is an advantage
  \iida This is a disadvantage
\eii
```

+ This is an advantage
– This is a disadvantage

Spacing in List Environments I

Better spacing between items, weighted by the itemize level.

- The `\btAddExtraItemSep[<sep>=\smallskipamount]` command advances `\itemsep` by `<sep> * (3 - itemize level)`.

```
\bii
  \ii Normal Spacing
  \ii Normal Spacing
  \btAddExtraItemSep
  \ii Extended Spacing
    \bi
      \ii Normal Spacing
      \ii Normal Spacing
    \ei
  \ii Extended Spacing
\eii
```

- Normal Spacing
- Normal Spacing
- Extended Spacing
  - Normal Spacing
  - Normal Spacing
- Extended Spacing

- It has to be applied inside the `itemize` environment and only affects the current level.

Spacing in List Environments II

- The `\btUseExtraItemSep[<sep>=\smallskipamount]` command patches the `itemize` environment, so that `\btAddExtraItemSep[<sep>]` is invoked implicitly:

```
\btUseExtraItemSep[1ex]
\bi
  \ii Extended Spacing
  \ii Extended Spacing
    \bi
      \ii Extended Spacing
      \ii Extended Spacing
    \ei
  \ii Extended Spacing
\ei
```

- Extended Spacing
- Extended Spacing
  - Extended Spacing
  - Extended Spacing
- Extended Spacing

- If applied at the begin of a `frame` environement, it affects all lists on the frame.
- This can be great to fine-tune the spacing.

Additional text styles

Some additional variants of the `\alert`, `\structure`, and a (all new) `\sample` command. All accept an `<overlay spec>`:

```
\bii
  \ii This is \alert{text}
  \ii This is \Alert{text}
  \ii This is \ALERT{text}
\eii
```

- This is text
- This is *text*
- This is **text**

```
\bii
  \ii This is \structure{text}
  \ii This is \Structure{text}
  \ii This is \STRUCTURE{text}
\eii
```

- This is text
- This is *text*
- This is **text**

```
\bii
  \ii This is \sample{text}
  \ii This is \Sample{text}
  \ii This is \SAMPLE{text}
\eii
```

- This is text
- This is *text*
- This is **text**

Previous frame title

The macros `\btPrevFrameTitle`, `\btPrevFrameSubtitle`, `\btPrevShortFrametitle` provide the title, subtitle and short title of the previous frame (look back to see what was the title):

```
\bii
  \ii \btPrevFrameTitle
  \ii \btPrevFrameSubtitle
  \ii \btPrevShortFrameTitle
\eii
```

- Additional text styles
- Very useful
- Additional text styles

The btBlock Environment I

- General structure

```
\begin{btBlock}<overlay spec>[pgfkeys key=val list]{title}
  block content
\end{btBlock}
```

- Minimal Example

```
\begin{btBlock}[]{Block}
  Something important
\end{btBlock}
```

Block

Something important

- Using block types: `/bt/type=alert|example|normal`

```
\begin{btBlock}[type=alert]{Block}
  Something important
\end{btBlock}
```

Block

Something important

The btBlock Environment II

```
\begin{btBlock}[type=example]{Block}
  Something important
\end{btBlock}
```

> **Block**
>
> Something important

- Scaling: `/bt/scale content=` and `/bt/scale=`
  - `/bt/scale content=` keeps width, but scales block content so that more stuff fits into it

    ```
    \begin{btBlock}[scale content=0.7]{
      Block}
      more info
    \end{btBlock}
    ```

    > **Block**
    >
    > more info

  - `/bt/scale=` scales block "as is", so that block consumes less space

    ```
    \begin{btBlock}[scale=0.7]{Block}
      more info
    \end{btBlock}
    ```

    > **Block**
    >
    > more info

- Setting block width: `/bt/text width=`

The btBlock Environment III

```
\begin{btBlock}[text width=5cm]{Block}
  more info
\end{btBlock}
```

Block

more info

```
\begin{btBlock}[text width=0.8\textwidth]{Block}
  more info
\end{btBlock}
```

Block

more info

- Horizontal alignment: `/bt/align=left|right|center`

The btBlock Environment IV

```
\begin{btBlock}[text width=0.8\textwidth,align=right]{Block}
  more info
\end{btBlock}
```

Block

more info

```
\begin{btBlock}[scale=0.8, align=center]{Block}
  more info
\end{btBlock}
```

Block

more info

- Beamer-Block options: `/bt/rounded` and `/bt/shadow`

```
\begin{btBlock}[shadow=false]{Block}
  more info
\end{btBlock}
```

Block

more info

The btBlock Environment V

```
\begin{btBlock}[rounded=false]{
 Block}
  more info
\end{btBlock}
```

Block

more info

- Setting defaults: The `/bt/every block` style

```
\btset{every block/.style={
  rounded, shadow=false,
  scale=0.8, center}
}
\begin{btBlock}{Block}
  more info
\end{btBlock}

\bigskip

\btset{every block/.append style={
 shadow, alert}}
\begin{btBlock}{Block}
  more info
\end{btBlock}
```

Block

more info

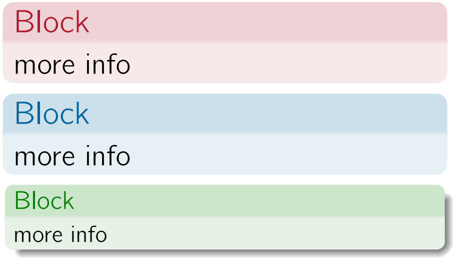
Block

more info

Wosch-compatible blocks

- If you load the package with the `/bt/woschblocks` option, the following environments will be defined on the base of **`btBlock`**.
  (Note that **`btBlock`** options can still be specified)

```
\begin{bearblock}{Block}
  more info
\end{bearblock}
\medskip
\begin{ovalblock}{Block}
  more info
\end{ovalblock}
\medskip
\def\shadow{true}
\begin{codeblock}[scale content
  =0.8]{Block}
  more info
\end{codeblock}
```



Block
more info

Block
more info

Block
more info

- These should be fully compatible to the ones Wosch uses in his slides (including handling of the **`\shadow`** macro)

Additional styles for TikZ

- Add to current font (instead of replacing it) `/tikz/add font=`*`font command`*
- Scale inner content of a node `/tikz/scale content=`*`factor`*
- Use beamer overlays with TikZ styles `/tikz/onslide=`

```
\tikz\node[%
    font=\ttfamily,
    onslide=<1>{draw=blue},
    onslide=<2->{fill=red!50, add font=\bfseries},
    onslide=<3>{scale content=1.5}
  ]{Attention!};
```

`Attention!`

Additional styles for TikZ

- Add to current font (instead of replacing it) `/tikz/add font=`*`font command`*
- Scale inner content of a node `/tikz/scale content=`*`factor`*
- Use beamer overlays with TikZ styles `/tikz/onslide=`

```
\tikz\node[%
    font=\ttfamily,
    onslide=<1>{draw=blue},
    onslide=<2->{fill=red!50, add font=\bfseries},
    onslide=<3>{scale content=1.5}
  ]{Attention!};
```

`Attention!`

Additional styles for TikZ

- Add to current font (instead of replacing it) `/tikz/add font=`*`font command`*
- Scale inner content of a node `/tikz/scale content=`*`factor`*
- Use beamer overlays with TikZ styles `/tikz/onslide=`

```
\tikz\node[%
    font=\ttfamily,
    onslide=<1>{draw=blue},
    onslide=<2->{fill=red!50, add font=\bfseries},
    onslide=<3>{scale content=1.5}
  ]{Attention!};
```

`Attention!`

Piecewise appearing for TikZ

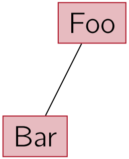
- Use beamer overlays for visibility `/tikz/visible on=`

```
\begin{tikzpicture}[every node/.style={fill=i4red!30, draw=i4red}]
  \node{Foo}
    child[visible on=<2->]{node {Bar}}
    child[visible on=<3->]{node {Baz}}
  ;
\end{tikzpicture}
```

Piecewise appearing for TikZ

- Use beamer overlays for visibility `/tikz/visible on=`

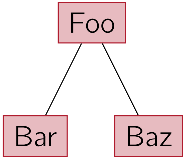
```
\begin{tikzpicture}[every node/.style={fill=i4red!30, draw=i4red}]
  \node{Foo}
    child[visible on=<2->]{node {Bar}}
    child[visible on=<3->]{node {Baz}}
  ;
\end{tikzpicture}
```



Piecewise appearing for TikZ

- Use beamer overlays for visibility `/tikz/visible on=`

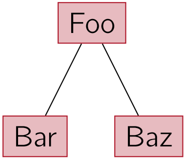
```
\begin{tikzpicture}[every node/.style={fill=i4red!30, draw=i4red}]
  \node{Foo}
    child[visible on=<2->]{node {Bar}}
    child[visible on=<3->]{node {Baz}}
  ;
\end{tikzpicture}
```



Piecewise appearing for TikZ

- Use beamer overlays for visibility `/tikz/visible on=`

```
\begin{tikzpicture}[every node/.style={fill=i4red!30, draw=i4red}]
  \node{Foo}
    child[visible on=<2->]{node {Bar}}
    child[visible on=<3->]{node {Baz}}
  ;
\end{tikzpicture}
```



Advantage: Elements are always there

- Image size does not depend on the overlay step
- Named nodes are always defined (for coordinate calculation)

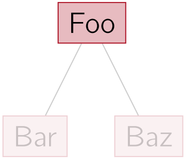
- Default implementation is based on `/tikz/opacity=0`:

```
\tikzset{
  invisible/.style={opacity=0},
  visible on/.style={alt=#1{}{invisible}},
}
```

Piecewise appearing for TikZ (cont.)

- By overriding the `/tikz/invisible` style, the "invisible" appearance can be customized (e.g., to dim elements instead)

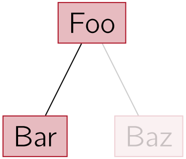
```
\tikzset{invisible/.style={opacity=0.2}}
\begin{tikzpicture}[every node/.style={fill=i4red!30, draw=i4red}]
  \node{Foo}
    child[visible on=<2->]{node {Bar}}
    child[visible on=<3->]{node {Baz}}
  ;
\end{tikzpicture}
```



Piecewise appearing for TikZ (cont.)

- By overriding the `/tikz/invisible` style, the "invisible" appearance can be customized (e.g., to dim elements instead)

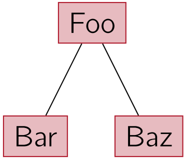
```
\tikzset{invisible/.style={opacity=0.2}}
\begin{tikzpicture}[every node/.style={fill=i4red!30, draw=i4red}]
  \node{Foo}
    child[visible on=<2->]{node {Bar}}
    child[visible on=<3->]{node {Baz}}
  ;
\end{tikzpicture}
```



Piecewise appearing for TikZ (cont.)

- By overriding the `/tikz/invisible` style, the "invisible" appearance can be customized (e.g., to dim elements instead)

```
\tikzset{invisible/.style={opacity=0.2}}
\begin{tikzpicture}[every node/.style={fill=i4red!30, draw=i4red}]
  \node{Foo}
    child[visible on=<2->]{node {Bar}}
    child[visible on=<3->]{node {Baz}}
  ;
\end{tikzpicture}
```



Highlighting lines in Listings

```
\lstset{language=C, numbers=left}
\begin{lstlisting}[
  autogobble,
  linebackgroundcolor={%
    \btLstHL{4}%
    \btLstHL<1>{1-2,5-6}%
    \btLstHL<2>{7}%
  }]
    /**
    * Prints Hello World.
    **/
    #include <stdio.h>

    int main(void) {
       printf("Hello World!");
       return 0;
    }
\end{lstlisting}
```

```
1 /**
2 * Prints Hello World.
3 **/
4 #include <stdio.h>
5
6 int main(void) {
7    printf("Hello World!");
8    return 0;
9 }
```

Highlighting lines in Listings

```
\lstset{language=C, numbers=left}
\begin{lstlisting}[
  autogobble,
  linebackgroundcolor={%
    \btLstHL{4}%
    \btLstHL<1>{1-2,5-6}%
    \btLstHL<2>{7}%
  }]
    /**
    * Prints Hello World.
    **/
    #include <stdio.h>

    int main(void) {
       printf("Hello World!");
       return 0;
    }
\end{lstlisting}
```

```
1 /**
2 * Prints Hello World.
3 **/
4 #include <stdio.h>
5
6 int main(void) {
7    printf("Hello World!");
8    return 0;
9 }
```

Highlighting lines in listings from external files

```
\btLstInputEmph[language=C, numbers=
  left]{3,6-7}{hello.c}
```

```
1 /**
2 * Prints Hello World.
3 **/
4 #include <stdio.h>
5
6 int main(void) {
7    printf("Hello World!");
8    return 0;
9 }
```

Highlighting single elements in listings

`\btHL<`*`overlay spec`*`>[`*`tikz key=val list`*`]` highlights till the end of a group (no line breaks, though). Hence, it can be as a ordinary font command with listings:

```
\bii
  \ii Some {text mit \btHL highlighting}, overlays are {\btHL<2>[red!20]also}
    possible.
\eii
\lstset{language=C, autogobble}
\begin{lstlisting}[
    moredelim={**[is][\btHL<1->]{@1}{@}},
    moredelim={**[is][{\btHL<2>}]{@2}{@}}
  ]
    #include @2<stdio.h>@

    int @1main@(void) {
       printf("Hello World!");
       return 0;
    }
\end{lstlisting}
```

- Some text mit highlighting, overlays are also possible.

```
#include <stdio.h>

int main(void) {
   printf("Hello World!");
   return 0;
}
```

Highlighting single elements in listings

\btHL<*overlay spec*>[*tikz key=val list*] highlights till the end of a group (no line breaks, though). Hence, it can be as a ordinary font command with listings:

```
\bii
  \ii Some {text mit \btHL highlighting}, overlays are {\btHL<2>[red!20]also}
    possible.
\eii
\lstset{language=C, autogobble}
\begin{lstlisting}[
    moredelim={**[is][\btHL<1->]{@1}{@}},
    moredelim={**[is][{\btHL<2>}]{@2}{@}}
  ]
    #include @2<stdio.h>@

    int @1main@(void) {
       printf("Hello World!");
       return 0;
    }
\end{lstlisting}
```

- Some text mit highlighting, overlays are also possible.

```
#include <stdio.h>

int main(void) {
   printf("Hello World!");
   return 0;
}
```

Highlighting single elements in listings

- `\btHL<overlay spec>[tikz key=val list]` actually draws the content inside a TikZ node, so you can play with named nodes and other options:

```
\begin{lstlisting}[language=C, autogobble, numbers=left,
    moredelim={**[is][{%
      \btHL[name=X, remember picture, onslide=<2->{fill=red!50}]%
    }]{@}{@}},
  ]
    @int main (void)@ {
       printf("Hello World!");
       return 0;
    }
\end{lstlisting}
% main() is typset into the node (X):
\tikz[remember picture, overlay]{
  \path<2> node[red, above right=3mm of X](L){This is the entry point};
  \draw<2>[->, red, shorten >=5pt] (L.west)--(X);
}
```

```
int main (void) {
   printf("Hello World!");
   return 0;
}
```

Highlighting single elements in listings

- `\btHL<overlay spec>[tikz key=val list]` actually draws the content inside a TikZ node, so you can play with named nodes and other options:

```
\begin{lstlisting}[language=C, autogobble, numbers=left,
    moredelim={**[is][{%
      \btHL[name=X, remember picture, onslide=<2->{fill=red!50}]%
    }]{@}{@}},
  ]
    @int main (void)@ {
       printf("Hello World!");
       return 0;
    }
\end{lstlisting}
% main() is typset into the node (X):
\tikz[remember picture, overlay]{
  \path<2> node[red, above right=3mm of X](L){This is the entry point};
  \draw<2>[->, red, shorten >=5pt] (L.west)--(X);
}
```

This is the entry point

```
int main (void) {
   printf("Hello World!");
   return 0;
}
```

Miscellaneous

- Dimension conversions with `\btConvertTo{dim}{dim value}`:

```
100pt=\btConvertTo{mm}{100pt}mm
```

100pt=35.14616mm

- Get file modification date of some file (ISO format) with `\btInsertFileModDate{file}`:

```
This document was changed on
\btInsertFileModDate{\jobname.tex}
```

This document was changed on 2013-12-13

- Real vertical fill to bottom with `\btVFill`, stackable

```
\btVFill
\fbox{Always at Bottom}
```

Always at Bottom

Always at Bottom