User's Guide to the Beamer Class, Version 1.21-dev

http://latex-beamer.sourceforge.net

Till Tantau
tantau@users.sourceforge.net

January 24, 2004

Contents

1 Introduction

1.1 Overview

This user's guide explains the functionality of the BEAMER class. It is a LaTeX class that allows you to create a presentation with a projector. It can also be used to create slides. It behaves similarly to other packages like PROSPER, but has the advantage that it works together directly with `pdflatex`, but also with `dvips`.

To use the BEAMER class, proceed as follows:

1. Specify `beamer` as document class instead of `article`.
2. Structure your LaTeX text using `section` and `subsection` commands.
3. Place the text of the individual slides inside `frame` commands.
4. Run `pdflatex` on the text (or `latex`, `dvips`, and `ps2pdf`).

The BEAMER class has several useful features: You don't need any external programs to use it other than `pdflatex`, but it works also with `dvips`. You can easily and intuitively create sophisticated overlays. Finally, you can easily change the whole slide theme or only parts of it. The following code shows a typical usage of the class.

```
\documentclass{beamer}

\usepackage{beamerthemesplit}

\title{Example Presentation Created with the Beamer Package}
\author{Till Tantau}
\date{\today}

\begin{document}

\frame{\titlepage}

\section*{Outline}
\frame{\tableofcontents}

\section{Introduction}
\subsection{Overview of the Beamer Class}
\frame
{
  \frametitle{Features of the Beamer Class}

  \begin{itemize}
  \item<1-> Normal LaTeX class.
  \item<2-> Easy overlays.
```

```
  \item<3-> No external programs needed.
  \end{itemize}
}
\end{document}
```

Run `pdflatex` on this code (twice) and then use, for example, the Acrobat Reader to present the resulting `.pdf` file in a presentation. You can also, alternatively, use `dvips`; see Section 3.4.2 for details.

As can be seen, the text looks almost like a normal LaTeX text. The main difference is the usage of the `\frame` command. This command takes one parameter, which is the text that should be shown on the frame. Typically, the contents of a frame is shown on a single slide. However, in case you use overlay commands inside a frame, a single frame command may produce several slides. An example is the last frame in the above example. There, the `\item` commands are followed by *overlay specifications* like `<1->`, which means "from slide 1 on." Such a specification causes the item to be shown only on the specified slides of the frame (see Section 4 for details). In the above example, a total of five slides are produced: a title page slide, an outline slide, a slide showing only the first of the three items, a slide showing the first two of them, and a slide showing all three items.

To structure your text, you can use the commands `\section` and `\subsection`. These commands will not only create entries in the table of contents, but will also in the navigation bars.

1.2 How to Read this User's Guide

This user guide is both intended as a tutorial and as a reference guide. If you have not yet installed the package, please read Section 2 first. If you do not have much experience with preparing presentations, Section 3 might be especially helpful. The later sections explain the basic usage of the `beamer` class as well as advanced features. If you wish to adjust the way your presentations look (for example, if you wish to add a default logo of your institution to every presentation in the future), please read the section on customization.

In this guide you will find the descriptions of all "public" commands provided by the `beamer` class. In each such description, the described command, environment, or option is printed in red. Text shown in green is optional and can be left out.

2 Installation and Compatibility

To use the beamer class, you just need to put the files of the BEAMER package in a directory that is read by TeX. To uninstall the class, simply remove these files once more. The same is true of the PGF package, which you will also need.

Unfortunately, there are different ways of making TeX "aware" of the files in the BEAMER package. Which way you should choose depends on how permanently you intend to use the class.

2.1 Installing Debian and Red Hat Packages

Currently, there are no out-of-the-box Debian or Red Hat packages of the beamer class available.

2.2 Temporary Installation

If you only wish to install the beamer class for a quick appraisal, do the following: Obtain the latest source version (ending `.tar.gz`) of the BEAMER package from `http://sourceforge.net/projects/latex-beamer/` (most likely, you have already done this). Next, you also need at least version 0.50 of the PGF package, which can be found at the same place. Finally, you need at least version 1.06 of the XCOLOR package, which can also be found at that place (although the version on CTAN might be newer).

In all cases, the packages contain a bunch of files (for the BEAMER class, `beamer.cls` is one of these files and happens to be the most important one, for the PGF package `pgf.sty` is the most important file). Place all files in three directories. For example, `~/beamer/`, `~/pgf/`, and `~/xcolor/` would work fine for me. Then setup the environment variable called `TEXINPUTS` to be the following string (how exactly this is done depends on your operating system and shell):

```
.:~/beamer/base:~/beamer/art:~/beamer/themes:~/pgf:~/xcolor:
```

Naturally, if the `TEXINPUTS` variable is already defined differently, you should *add* the five directories to the list. Do not forget to place a colon at the end (corresponding to an empty path), which will include all standard directories.

2.3 Installation in a texmf Tree

For a more permanent installation, you can place the files of the BEAMER package and of the PGF package (see the previous subsection on how to obtain them) in an appropriate `texmf` tree.

When you ask TEX to use a certain class or package, it usually looks for the necessary files in so-called `texmf` trees. These trees are simply huge directories that contain these files. By default, TEX looks for files in three different `texmf` trees:

- The root `texmf` tree, which is usually located at `/usr/share/texmf/`, `c:\texmf\`, or `c:\Program Files\TeXLive\texmf\`.
- The local `texmf` tree, which is usually located at `/usr/local/share/texmf/`, `c:\localtexmf\`, or `c:\Program Files\TeXLive\texmf-local\`.
- Your personal `texmf` tree, which is located in your home directory.

You should install the packages either in the local tree or in your personal tree, depending on whether you have write access to the local tree. Installation in the root tree can cause problems, since an update of the whole TEX installation will replace this whole tree.

Inside whatever `texmf` directory you have chosen, create the sub-sub-sub-directories

- `texmf/tex/latex/beamer` and
- `texmf/tex/latex/pgf`
- `texmf/tex/latex/xcolor`

and place all files in these three directories.

Finally, you need to rebuild TEX's filename database. This done by running the command `texhash` or `mktexlsr` (they are the same). In MikTeX, there is a menu option to do this.

For a more detailed explanation of the standard installation process of packages, you might wish to consult `http://www.ctan.org/installationadvice/`. However, note that the BEAMER package does not come with a `.ins` file (simply skip that part).

2.4 Updating the Installation

To update your installation from a previous version, simply replace everything in the directories like `texmf/tex/latex/beamer` with the files of the new version. The easiest way to do this is to first delete the old version and then proceed as described above. Sometimes, there are changes in the syntax of certain command from version to version. If things no longer work that used to work, you wish to have a look at the release notes and at the change log.

2.5 Testing the Installation

To test your installation, copy the file `beamerexample1.tex` from the examples subdirectory to some place where you usually create presentations. Then run the command `pdflatex` several times on the file and check whether the resulting `beamerexample1.pdf` looks correct. If so, you are all set.

2.6 Compatibility

When using certain packages together with the `beamer` class, extra options or precautions may be necessary.

`\usepackage[french]{babel}`

When using the French style, certain features that clash with the functionality of the beamer class will be turned off. For example, enumerations are still produced the way the theme dictates, not the way the French style does.

`\usepackage[spanish]{babel}`

When using the Spanish style, certain features that clash with the functionality of the beamer class will be turned off. In particular, the special behaviour of the pointed brackets < and > is deactivated.

`\usepackage{CJK}`

When using the `CJK` package, you must use the class option `CJK`. See `beamerexample4.tex` for an example.

`\usepackage{deluxetable}`

The caption generation facilities of `deluxetable` are deactivated. Instead, the caption template is used.

`\usepackage[T1]{fontenc}`

Use this option only with fonts that have outline fonts available in the T1 encoding like Times or the lmodern fonts. In a standard installation the standard Computer Modern fonts (the fonts Donald Knuth orginally designed and which are used by default) are *not* available in the T1 encoding. Using this option with them will result in very poor rendering of your presentation when viewed with PDF viewer applications like Acrobat or `xpdf`. To use the Computer Modern fonts with the T1 encoding, use the package `lmodern`. See also Section 8.1.5.

`\usepackage{fourier}`

The package switches to a T1 encoding, but it does not redefine all fonts such that outline fonts (non-bitmapped fonts) are used by default. For example, the sans-serif text and the typewriter text are not replaced. To use outline fonts for these, write `\usepackage{lmodern}` *before* including the `fourier` package.

`\usepackage{listings}`

Note that you must treat `lstlisting` environments exactly the same way as you would treat `verbatim` environments.

3 Workflow

This section presents a possible workflow for creating a beamer presentation and possibly a handout to go along with it. Technical questions are addressed, like which programs to call with which parameters, and hints are given on how to create a presentation. If you have already created numerous presentations, you may wish to skip the first of the following steps and only have a look at how to convert the `.tex` file into a `.pdf` or `.ps` file.

3.1 Step Zero: Know the Time Constraints

When you start to create a presentation, the very first thing you should worry about is the amount of time you have for your presentation. Depending on the occasion, this can be anything between 2 minutes and two hours. A simple rule for the number of frames is that you should have at most one frame per minute.

In most situations, you will have less time for your presentation that you would like. *Do not try to squeeze more into a presentation than time allows for.* No matter how important some detail seems to you, it is better to leave it out, but get the main message across, than getting neither the main message nor the detail across.

In many situations, a quick appraisal of how much time you have will show that you won't be able to mention certain details. Knowing this can save you hours of work on preparing slides that you would have to remove later anyway.

3.2 Step One: Setup the Files

It is advisable that you create a folder for each presentation. Even though your presentation will usually reside in a single file, TEX produces so many extra files that things can easily get very confusing otherwise. The folder's name should ideally start with the date of your talk in ISO format (like 2003-12-25 for a Christmas talk), followed by some reminder text of what the talk is all about. Putting the date at the front in this format causes your presentation folders to be listed nicely when you have several of them residing in one directory. If you use an extra directory for each presentation, you can call your main file `main.tex`.

To create an initial `main.tex` file for your talk, copy an existing file (like the file `beamerexample1.tex` that comes along with the contribution) and delete everything that is not going to be part of your talk. Adjust the `\author` and other fields as appropriate.

If you wish your talk to reside in the same file as some different, non-presentation article version of your text, it is advisable to setup a more elaborate file scheme. See Section 7.4.2 for details.

3.3 Step Two: Structure You Presentation

With the time constraints in mind, make a mental inventory of the things you can reasonably talk about within the time available. Then categorize the inventory into sections and subsections. For very long talks (like a 90 minute lecture), you might also divide your talk into independent parts (like a "review of the previous lecture part" and a "main part"). Put `\section` and `\subsection` commands into the (more or less empty) main file. Do not create any frames until you have a first working version of a possible table of contents. Do not feel afraid to change it later on as you work on the talk.

You should not use more than four sections and not less than two per part. Even four sections are usually too much, unless they follow a very easy pattern. Five and more sections are simply too hard to remember for the audience. After all, when you present the table of contents, the audience will not yet really be able to grasp the importance and relevance of the different sections and will most likely have forgotten them by the time you reach them.

Ideally, a table of contents should be understandable by itself. In particular, it should be comprehensible *before* someone has heard your talk. Keep section and subsection titles self-explaining. Note each part has its own table of contents.

Both the sections and the subsections should follow a logical pattern. Begin with an explanation of what your talk is all about. (Do not assume that everyone knows this. The Ignorant Audience Law states: The audience always knows less than you think it should know, even if you take the Ignorant Audience Law into account.) Then explain what you or someone else has found out concerning the subject matter. Always conclude your talk with a summary that repeats the main message of the talk in a short and simple way. People pay most attention at the beginning and at the end of talks. The summary is your "second chance" to get across a message.

You can also add an appendix part using the `\appendix` command. Put everything into this part which you do not actually intend to talk about, but which might come in handy in questions are asked.

3.4 Step Three: Creating a PDF or PostScript File

Once a first version of the structure is finished, you should create a first PDF or PostScript file of your (still empty) talk. This file will only contain the title page and the table of contents. The file might look like this:

```
\documentclass{beamer}
% This is the file main.tex

\usepackage{beamerthemesplit}

\title{Example Presentation Created with the Beamer Package}
\author{Till Tantau}
\date{\today}

\begin{document}

\frame{\titlepage}

\section*{Outline}
\frame{\tableofcontents}

\section{Introduction}
\subsection{Overview of the Beamer Class}
\subsection{Overview of Similar Classes}

\section{Usage}
\subsection{...}
\subsection{...}

\section{Examples}
\subsection{...}
\subsection{...}

\end{document}
```

3.4.1 Creating PDF

To create a PDF version of this file, run the program `pdflatex` on `main.tex` at least twice. Your need to run it twice, so that TEX can create the table of contents. (It may even be necessary to run it more often since all sorts of auxilliary files are created.) In the following example, the greater-than-sign is the prompt.

```
> pdflatex main.tex
    ... lots of output ...
> pdflatex main.tex
    ... lots of output ...
```

You can next use a program like the Acrobat Reader or `xpdf` to view the resulting presentation.

```
> acroread main.pdf
```

When printing a presentation using Acrobat, make sure that the option "expand small pages to paper size" is enabled. This is necessary, because slides are only 128mm times 96mm.

To put several slides onto one page (useful for the handout version) or to enlarge the slides, you can use the program `pdfnup`. Also, many commercial programs can perform this task.

3.4.2 Creating PostScript

To create a PostScript version, you should first ascertain that the HYPERREF package (which is automatically loaded by the BEAMER class) uses the option `dvips` or some compatible option, see the documentation of the HYPERREF package for details. Whether this is the case depends on the contents of your local `hyperref.cfg` file. You can enforce the usage of this option by passing `dvips` or a compatible option to the BEAMER class (write `\documentclass[dvips]{beamer}`), which will pass this option on to the HYPERREF package.

You can then run `latex` twice, followed by `dvips`.

```
> latex main.tex
    ... lots of output ...
> latex main.tex
    ... lots of output ...
> dvips -P pdf main.dvi
```

The option (`-P pdf`) tells `dvips` to use Type 1 outline fonts instead of the usual Type 3 bitmap fonts. You may wish to omit this option if there is a problem with it.

If you wish each slide to completely fill a letter-sized page, use the following commands instead:

```
> dvips -P pdf -tletter main.dvi -o main.temp.ps
> psnup -1 -W128mm -H96mm -pletter main.temp.ps main.ps
```

For A4-sized paper, use:

```
> dvips -P pdf -ta4 main.dvi -o main.temp.ps
> psnup -1 -W128mm -H96mm -pa4 main.temp.ps main.ps
```

In order to create a white margin around the whole page (which is sometimes useful for printing), add the option `-m 1cm` to the options of `psnup`.

To put two or four slides on one page, use `-2`, respectively `-4` instead of `-1` as the first parameter for `psnup`. In this case, you may wish to add the option `-b 1cm` to add a bit of space around the individual slides.

You can convert a PostScript file to a pdf file using

```
> ps2pdf main.ps main.pdf
```

3.5 Step Four: Create Frames

Once the table of contents looks satisfactory, start creating frames for your presentation. In the following, some guidelines that I stick to are given on what to put on slides and what not to put. You can certainly ignore any of these guideline, but you should be aware of it when you ignore a rule and you should be able to justify it to yourself.

3.5.1 Guidelines on What to Put on a Frame

- A frame with too little on it is better than a frame with too much on it.
- Do not assume that everyone in the audience is an expert on the subject matter. (Remember the Ignorant Audience Law.) Even if the people listening to you should be experts, they may last have heard about things you consider obvious several years ago. You should always have the time for a quick reminder of what exactly a "semantical complexity class" or an "ω-complete partial ordering" is.
- Never put anything on a slide that you are not going to explain during the talk, not even to impress anyone with how complicated your subject matter really is. However, you may explain things that are not on a slide.
- Keep it simple. Typically, your audience will see a slide for less than 50 seconds. They will not have the time to puzzle through long sentences or complicated formulas.

3.5.2 Guidelines on Text

- Put a title on each frame. The title explains the contents of the frame to people who did not follow all details on the slide.
- Ideally, titles on consecutive frames should "tell a story" all by themselves.
- *Never* use a smaller font size to "squeeze more on a frame."
- Prefer enumerations and itemize environments over plain text. Do not use long sentences.
- Do not hyphenate words. If absolutely necessary, hyphenate words "by hand," using the command `\-`.
- Break lines "by hand" using the command `\\`. Do not rely on automatic line breaking. Break where there is a logical pause. For example, good breaks in "the tape alphabet is larger than the input alphabet" are before "is" and before the second "the." Bad breaks are before either "alphabet" or before "larger."
- Text and numbers in figures should have the *same* size as normal text. Illegible numbers on axes usually ruin a chart and its message.

3.5.3 Guidelines on Graphics

- Put (at least) one graphic on each slide, whenever possible. Visualizations help an audience enormously.
- Usually, place graphics to the left of the text. (Use the `columns` environment.)
- Graphics should have the same typographic parameters as the text: Use the same fonts (at the same size) in graphics as in the main text. A small dot in a graphic should have exactly the same size as a small dot in a text. The line width should be the same as the stroke width used in creating the glyphs of the font. For example, an 11pt non-bold Computer Modern font has a stroke width of 0.4pt.
- While bitmap graphics, like photos, can be much more colorful than the rest of the text, vector graphics should follow the same "color logic" as the main text (like black = normal lines, red = hilighted parts, green = examples, blue = structure).
- Like text, you should explain everything that is shown on a graphic. Unexplained details make the audience puzzle whether this was something important that they have missed. Be careful when importing graphics from a paper or some other source. They usually have much more detail than you will be able to explain.

For technical hints on how to create graphics, see Section 6.1.

3.5.4 Guidelines on Colors

- Use colors sparsely. The prepared themes are already quite colorful (blue = structure, red = alert, green = example). If you add more colors, you should have a *very* good reason.
- Be careful when using bright colors on white background, *especially* when using green. What looks good on your monitor may look bad during a presentation due to the different ways monitors, beamers, and printers reproduce colors. Add lots of black to pure colors when you use them on bright backgrounds.
- Maximize contrast. Normal text should be black on white or at least something very dark on something very bright. *Never* do things like "light green text on not-so-light green background."
- Background shadings decrease the legibility without increasing the information content. Do not add a background shading just because it "somehow looks nicer." In the examples that come along with the BEAMER class, the backgrounds are intended as demonstrations, not as recommendations.
- Inverse video (bright text on dark background) can be a problem during presentations in bright environments since only a small percentage of the presentation area is light up by the beamer. Inverse video is harder to reproduce on printouts and on transparencies.

3.5.5 Guidelines on Animations and Special Effects

- Use animations to explain the dynamics of systems, algorithms, etc.
- Do *not* use animations just to attract the attention of your audience. This often distracts attention away from the main topic of the slide.
- Do *not* use distracting special effects like "dissolving" slides unless you have a very good reason for using them. If you use them, use them sparsely.

3.5.6 Using the Draft Option

While working on your presentation, it may sometimes be useful to TEX your `.tex` file quickly and have the presentation contain only the most important information. This is especially true if you have a slow machine. In this case, the `draft` class option is useful.

`\documentclass[draft]{beamer}`

Causes the head lines, foot lines, and sidebars to be replaced by gray rectangles (their sizes are still computed, though). Many other packages, including `pgf` and `hyperref`, also "speedup" when this option is given.

3.6 Step Five: Test Your Presentation

Always test your presentation. For this, you should vocalize or subvocalize your talk in a quiet environment. Typically, this will show that your talk is too long. You should then remove parts of the presentation, such that it fits into the allotted time slot. Do *not* attempt to talk faster in order to squeeze the talk into the given amount of time. You are almost sure to loose your audience this way.

Do not try to create the "perfect" presentation immediately. Rather, test and retest the talk and modify it as needed.

3.7 Step Six: Optionally Create a Handout or an Article Version

Once your talk is fixed, you can create a handout, if this seems appropriate. For this, use the class option `handout` as explained in Section 7.1. Typically, you might wish to put several handout slides on one page. See Section 3.4.2 on how to do this.

You may also wish to create an article version of your talk. An "article version" of your presentation is a normal TEX text typeset using, for example, the document class `article` or perhaps `lncs` or a similar document class. The BEAMER class offers facilities to have this version coexist with your presentation version in one file and to share code. Also, you can include slides of your presentation as figures in your article version. Details on how to setup the article version can be found in Section 7.4.

4 Frames and Overlays

4.1 Frames

4.1.1 Frame Creation

A presentation consists of a series of frames. Each frame consists of a series of slides. You create a frame using the command `\frame`. This command takes one parameter, namely the contents of the frame. All of this text that is not tagged by overlay specifications (see Section 4.2.2) is shown on all slides of the frame.

`\frame<`⟨*overlay specification*⟩`>[`⟨*options*⟩`]{`⟨*frame text*⟩`}`

The ⟨*overlay specification*⟩ dictates which slides of a frame are to be shown, see Section 4.1.3 for details. The ⟨*frame text*⟩ can be normal LaTeX text, but may not contain `\verb` commands or `verbatim` environments, unless special precautions are taken, see Section 4.1.4.

Example:

```
\frame
{
  \frametitle{A title}
  Some content.
}
```

The following ⟨*options*⟩ may be given:

- `plain` causes the head lines, foot lines, and side bars are suppressed. This is useful for creating single frames with different head and foot lines or for creating frames showing big pictures that completely fill the frame.

  Example: A frame with a picture completely filling the frame:

  ```
  \frame[plain]{\hfill\pgfimage[height=9.6cm]{bigimagefilename}\hfill}
  ```

  Example: A title page, in which the head and foot lines are replaced by two graphics.

  ```
  \usetitlepagetemplate{
    \beamerline{\pgfuseimage{toptitle}}
    \vskip0pt plus 1filll

    \begin{centering}
      \Large{\textbf{\inserttitle}}

      \insertdate
    \end{centering}

    \vskip0pt plus 1filll
    \beamerline{\pgfuseimage{bottomtitle}}
  }
  \frame[plain]{\titlepage}
  ```

- `label=`⟨*name*⟩ causes the frame's contents to be stored under the name ⟨*name*⟩ for later resumption using the command `\resumeframe`. If this option is given, you cannot include verbatim text in the frame, even if you specify an overlay specification like `<1>`. The frame is still rendered normally. See also `\resumeframe`.

  Furthermore, on each slide of the frame a label with the name ⟨*name*⟩`<`⟨*slide number*⟩`>` is created. Note that labels in general, and these labels in particular, can be used as targets for hyperlinks.

For compatibility with earlier versions, you can also give an overlay specification in square brackets. If the sole argument to the `\frame` command is an argument in square brackets, the BEAMER class will try to check whether this argument "looks like" an overlay specification. If so, it is assumed to be an overlay specification.

Note that there is *no* environment for creating frames. The reason is that I have simply not been able to come up with any idea of how to implement it.

`\resumeframe<`⟨*overlay specification*⟩`>[`⟨*options*⟩`]{`⟨*name*⟩`}`

Resumes a frame that was previously created using `\frame` with the option `label=`⟨*name*⟩. You can use this command to "continue" a frame that has been interrupted by another frame. The effect of this command is to call the `\frame` command with the given ⟨*overlay specification*⟩ and ⟨*options*⟩ (if present) and with the original frame's contents.

Example:

```
\frame<1-2>[label=myframe]
{
  \begin{itemize}
  \item \alert<1>{First subject.}
  \item \alert<2>{Second subject.}
  \item \alert<3>{Third subject.}
  \end{itemize}
}

\frame
{
  Some stuff explaining more on the second matter.
}

\resumeframe<3>{myframe}
```

The effect of the above code is to create four slides. In the first two, the items 1 and 2 are hilighted. The third slide contains the text "Some stuff explaining more on the second matter." The fourht slide is identical to the first two slides, except that the third point is now hilighted.

Example:

```
\frame<1>[label=Cantor]
{
  \frametitle{Main Theorem}

  \begin{Theorem}
    $\alpha < 2^\alpha$ for all ordinals~$\alpha$.
  \end{Theorem}

  \only<1>
  {
    \hyperlink{Cantor<2>}{\beamergotobutton{Proof details}}
  }
  \only<2>
  {% this is only shown in the appendix, where this frame is resumed.
    \begin{proof}
      As shown by Cantor, ...
    \end{proof}

    \hfill\hyperlink{Cantor<1>}{\beamerreturnbutton{Return}}
  }
}

...
\appendix

\resumeframe<2>{Cantor}
```

In this example, the proof details are deferred to a slide in the appendix. Hyperlinks are setup, so that one can jump to the proof and go back.

4.1.2 Components of a Frame

Each frame consists of several components:

1. a head line,
2. a foot line,

3. a left side bar,
4. a right side bar,
5. navigation symbols,
6. a logo,
7. a frame title, and
8. some frame contents.

A frame need not have all of these components. Usually, the first six components are automatically setup by the theme you are using. To change them, you must install an appropriate template, see Section 8.4.8 for the head and foot lines and Section 8.4.9 for the side bars. To install a logo, invoke the following command in the preamble, *after* having loaded the theme:

`\logo{`⟨*logo text*⟩`}`

The ⟨*logo text*⟩ is usually a command for including a graphic.

Example:

```
\pgfdeclareimage[height=0.5cm]{logo}{tu-logo}
\logo{\pgfuseimage{logo}}
```

The frame title is shown prominently at the top of the frame and can be specified with the following command:

`\frametitle{`⟨*frame title text*⟩`}`

You should end the ⟨*frame title text*⟩ with a period, if the title is a proper sentence. Otherwise, there should not be a period.

Example:

```
\frame{
  \frametitle{A Frame Title is Important.}

  Frame contents.
}
```

Be default, all material for a slide is vertically centered. You can change this using the following class options:

`\documentclass[slidestop]{beamer}`

Place text of slides at the (vertical) top of the slides. This corresponds to a vertical "flush."

`\documentclass[slidescentered]{beamer}`

Place text of slides at the (vertical) center of the slides. This is the default.

4.1.3 Restricting the Slides of a Frame

The number of slides in a frame is automatically calculated. If the largest number mentioned in any overlay specification inside the frame is 4, four slides are introduced (despite the fact that a specification like `<4->` might suggest that more than four slides would be possible).

You can also specify the number of slides in the frame "by hand." To do so, you pass an overlay specification the `\frame` command. The frame will contain only the slides specified in this argument. Consider the following example.

```
\frame<1-2,4->
{
  This is slide number \only<1>{1}\only<2>{2}\only<3>{3}%
  \only<4>{4}\only<5>{5}.
}
```

This command will create a frame containing four slides. The first will contain the text "This is slide number 1," the second "This is slide number 2," the third "This is slide number 4," and the fourth "This is slide number 5."

A useful specification is just `<0>`, which causes the frame to have to no slides at all. For example, `\frame<handout:0>` causes the frame to be suppressed in the handout version, but to be shown normally in all other versions.

4.1.4 Verbatim Commands and Listings inside Frames

The `\verb` command, the `verbatim` environment, the `lstlisting` environment, and related environments that allow you to typeset arbitrary text work only in frames that contain a single slide or that are suppressed altogether. Furthermore, you must explicitly specify that the frame contains only one slide; like this:

```
\frame<all:1>
{
  \frametitle{Our Search Procedure}

\begin{verbatim}
  int find(int* a, int n, int x)
  {
    for (int i = 0; i<n; i++)
      if (a[i] == x)
        return i;
  }
\end{verbatim}

}
```

Instead of `\frame<all:1>` you could also have specified `\frame<1>`, but this works only for the presentation version of the talk, not for the handout version. To make verbatim accessible also in the handout version, you would have to specify `\frame<1| handout: 1>` and even more if you also have a transparencies version. The specification `\frame<all:1>` states that the frame has just one slide in all versions. You may *not* use the `label=`⟨*label name*⟩ option if you have a verbatim text on a slide.

If you need to use verbatim commands in frames that contain several slides or on a frame that uses the `label` option, you must *declare* your verbatim texts before the frame starts. This is done using two special commands:

`\defverb{`⟨*command name*⟩`}*`⟨*delimiter symbol*⟩⟨*verbatim text*⟩⟨*delimiter symbol*⟩

Declares a verbatim text for later use. The declaration should be done outside the frame. Once declared, the text can be used in overlays like normal text. The one-line ⟨*verbatim text*⟩ must be delimited by a special ⟨*delimiter symbol*⟩ (works like the `\verb` command). Adding a star makes spaces visible.

Example:

```
\defverb\mytext!int main (void) { ...!
\defverb\mytextspaces*!int  main  (void ){  ...!

\frame
{
  \begin{itemize}
  \item<1-> In C you need a main function.
  \item<2-> It is declare like this: \mytext
  \item<3-> Spaces are not important: \mytextspaces
  \end{itemize}
}
```

`\defverbatim{`⟨*command name*⟩`}{`⟨*text*⟩`}`

The ⟨*text*⟩ may contain a `verbatim`, `verbatim*`, `lstlisting`, or a related environment. The command `{`⟨*command name*⟩`}` can be used later inside frames. The declaration should be done outside the frame. Once declared, the text can be used in overlays like normal text.

Example:

```
\defverbatim\algorithm{
\begin{verbatim}
int main (void)
{
  cout << "Hello world." << endl;
  return 0;
}
```

```
\end{verbatim}
}

\frame
{
  Our algorithm:
  \alert<1>{\algorithm}
  \uncover<2>{Note the return value.}
}
```

4.2 Overlays

4.2.1 The Pauses Environment

The `pauses` environment offers an easy, but not very flexible way of creating frames that are uncovered piecewise. The environment itself does not have an immediate effect. But if you use the command `\pause` inside the environment, only the text of the environment up to the `\pause` command is shown on the first slide. On the second slide, everything is shown up to the second `\pause`, and so forth. Note that the `\pause` command can only be used on the same level of nesting as the `pauses` environment.

A much more fine-grained control over what is shown on each slide can be attained using overlay specifications, see the next subsections. However, for many simple cases the `\pause` command is sufficient.

If you use multiple `pauses` environments on one frame, the slide counting for the second environment starts where the first one left off, see the following example. You can nest `pauses` environments, but this will not always have the effect you might expect.

```
\frame{
  \begin{pauses}
    Shown from first slide on.
    \pause
    Shown from second slide on.
    \pause
    Shown from third slide on.
  \end{pauses}

  Shown from first slide on (not affected by the environment).

  \begin{pauses}
    Shown from third slide on. (continued from above)
    \pause
    Shown from fourth slide on.
  \end{pauses}
}
```

As a convenience, a `pauses` environment is automatically setup inside each frame, each `itemize`, each `description`, and each `enumerate`. Thus, by simply using the `\pause` command on the outermost level of any frame or after items in lists or descriptions, you uncover the rest of the frame or list only on the next slide.

`\begin{pauses}`[⟨*start slide number*⟩]
⟨*environment contents*⟩
`\end{pauses}`

The content of the environment is shown piecewise. Each `\pause` command used inside uncovers a bit more of the environment's text. The main use of ⟨*start slide number*⟩ is to set it to 0. The effect of this is that the first `\pause` has no effect, which can be useful if the `pauses` environment immediately starts with a `\pause` command. This happens sometimes when the environment's content is created automatically.

Example:

```
\frame
{
  \begin{pauses}
    Shown from slide 1 onward.
```

```
    \pause

    Shown from slide 2 onward.
  \end{pauses}
}
```

As mentioned above, in the above example the `pauses` environment could also have been omitted, as the `\frame` command inserts it automatically.

`\pause`

When used inside a `pauses` environment, this command causes the text following it to be shown only from the next slide on.

Example:

```
\frame
{
  \begin{itemize}
  \item
    A
    \pause
  \item
    B
    \pause
  \item
    C
  \end{itemize}
}
```

4.2.2 Commands with Overlay Specifications

An overlay specification is a comma-separated list of slides and ranges. Ranges are specified like this: `2-5`, which means slide two through to five. The start or the beginning of a range can be omitted. For example, `3-` means "slides three, four, five, and so on" and `-5` means the same as `1-5`. A complicated example is `-3,6-8,10,12-15`, which selected the slides 1, 2, 3, 6, 7, 8, 10, 12, 13, 14, and 15.

Overlay specifications can be written behind certain commands. If such an overlay specification is present, the command will only "take effect" on the specified slides. What exactly "take effect" means depends on the command. Consider the following example.

```
\frame
{
  \textbf{This line is bold on all three slides.}
  \textbf<2>{This line is bold only on the second slide.}
  \textbf<3>{This line is bold only on the third slide.}
}
```

For the command `\textbf`, the overlay specification causes the text to be set in boldface only on the specified slides. On all other slides, the text is set in a normal font.

You cannot add an overlay specification to every command, but only to those listed in the following. However, it is quite easy to redefine a command such that it becomes "overlay specification aware."

For the following commands, adding an overlay specification causes the command to be simply ignored on slides that are not included in the specification: `\textbf`, `\textit`, `\textsl`, `\textrm`, `\textsf`, `\color`, `\alert`, `\structure`. If a command takes several arguments, like `\color`, the specification directly follows the command as in the following example.

```
\frame
{
  \color<2-3>[rgb]{1,0,0} This text is red on slides 2 and 3, otherwise black.
}
```

For the following commands, the effect of an overlay specification is special:

`\only<`⟨*overlay specification*⟩`>{`⟨*text*⟩`}`

If the ⟨*overlay specification*⟩ is present, the ⟨*text*⟩ is inserted only into the specified slides. For other slides, the text is simply thrown away. In particular, it occupies no space.

Example: `\only<3->{Text inserted from slide 3 on.}`

There exists a variant of `\only`, namely `\pgfonly`, that should be used inside PGF pictures instead of `\only`. The command `\pgfonly` inserts appropriate `\ignorespaces` commands that are needed by PGF.

`\uncover<`⟨*overlay specification*⟩`>{`⟨*text*⟩`}`

If the ⟨*overlay specification*⟩ is present, the ⟨*text*⟩ is shown ("uncovered") only on the specified slides. On other slides, the text still occupies space and it is still typeset, but it is not shown or only shown as if transparent. For details on how to specify whether the text is invisible or just transparent, see Section 6.2.3.

Example: `\uncover<3->{Text shown from slide 3 on.}`

`\invisible<`⟨*overlay specification*⟩`>{`⟨*text*⟩`}`

The ⟨*text*⟩ occupies space and it is typeset, but it is not shown. If the ⟨*overlay specification*⟩ is given, this command takes effect only on the specified slides. This command is a conter-part to `\uncover`, but not quite: unlike `\uncover`, invisible text is never shown in a transparent way, but is guaranteed to really be invisible.

Example: `\invisible<-2>{Text shown from slide 3 on.}`

`\alt<`⟨*overlay specification*⟩`>{`⟨*default text*⟩`}{`⟨*alternative text*⟩`}`

The default text is shown on the specified slides, otherwise the alternative text. The specification must always be present.

Example: `\alt<2>{On Slide 2}{Not on slide 2.}`

`\temporal<`⟨*overlay specification*⟩`>{`⟨*before slide text*⟩`}{`⟨*default text*⟩`}{`⟨*after slide text*⟩`}`

This command alternates between three different texts, depending on whether the current slide is temporally before the specified slides, is one of the specified slides, or comes after them. If the ⟨*overlay specification*⟩ is not an interval (that is, if it has a "hole"), the "hole" is considered to be part of the before slides.

Example:

```
\temporal<3-4>{Shown on 1, 2}{Shown on 3, 4}{Shown 5, 6, 7, ...}
\temporal<3,5>{Shown on 1, 2, 4}{Shown on 3, 5}{Shown 6, 7, 8, ...}
```

As a possible application of the `\temporal` command consider the following example:

Example:

```
\def\colorize<#1>{%
  \temporal<#1>{\color{structure!50}}{\color{black}}{\color{black!50}}}

\frame{
  \begin{itemize}
    \colorize<1> \item First item.
    \colorize<2> \item Second item.
    \colorize<3> \item Third item.
    \colorize<4> \item Fourth item.
  \end{itemize}
}
```

`\item<`⟨*overlay specification*⟩`>[`⟨*item label*⟩`]`

Adding an ⟨*overlay specification*⟩ to an item in a list causes this item to be uncovered only on the specified slides. This is useful for creating lists that are uncovered piecewise. Note that you are not required to stick to an order in which items are uncovered. If present, the optional ⟨*item label*⟩ comes after the overlay specification.

Example:

```
\frame
{
  \begin{itemize}
  \item<1-> First point, shown on all slides.
  \item<2-> Second point, shown on slide 2 and later.
  \item<2-> Third point, also shown on slide 2 and later.
  \item<3-> Fourth point, shown on slide 3.
  \end{itemize}
}

\frame
{
  \begin{enumerate}
  \item<3->[0.] A zeroth point, shown at the very end.
  \item<1-> The first an main point.
  \item<2-> The second point.
  \end{enumerate}
}
```

Example: In the following example a list is uncovered item-wise. The last uncovered item is furthermore hilighted.

```
\frame
{
  The advantages of the beamer class are
  \begin{enumerate}
  \item<1-> \alert<1>{It is easy to use.}
  \item<2-> \alert<2>{It is easy to extend.}
  \item<3-> \alert<3>{It works together with \texttt{pdflatex}.}
  \item<4-> \alert<4>{It has nice overlays.}
  \end{enumerate}
}
```

The related command `\bibitem` is also overlay-specification-aware in the same way as `\item`.

`\label<`⟨*overlay specification*⟩`>{`⟨*label name*⟩`}`

If the ⟨*overlay specification*⟩ is present, the label is only inserted on the specified slide. Inserting a label on more than one slide will cause a 'multiple labels' warning. *However*, if no overlay specification is present, the specification is automatically set to just '1' and the label is thus inserted only on the first slide. This is typically the desired behaviour since it does not really matter on which slide the label is inserted, *except* if you use an `\only` command. Then you need to specifiy a slide.

Example:

```
\frame
{
  \begin{align}
    a &= b + c   \label{first}\\ % no specification needed
    c &= d + e   \label{second}\\% no specification needed
  \end{align}

  Blah blah, \uncover<2>{more blah blah.}

  \only<3>{Specification is needed now.\label<3>{mylabel}}
}
```

4.2.3 Environments with Overlay Specifications

Environments can also be equipped with overlay specifications. For most of the predefined environments, see subsection 5.4.2, adding an overlay specifications causes the whole environment to be uncovered only on the specified slides. This is useful for showing things incrementally as in the following example.

```
\frame
```

```
{
  \frametitle{A Theorem on Infinite Sets}

  \begin{theorem}<1->
    There exists an infinite set.
  \end{theorem}

  \begin{proof}<3->
    This follows from the axiom of infinity.
  \end{proof}

  \begin{example}<2->
    The set of natural numbers is infinite.
  \end{example}
}
```

In the example, the first slide only contains the theorem, on the second slide an example is added, and on the third slide the proof is also shown.

The two special environments `onlyenv` and `uncoverenv` are "environment versions" of the commands `\only` and `\uncover`.

`\begin{onlyenv}`<⟨*overlay specification*⟩>
⟨*environment contents*⟩
`\end{onlyenv}`

If the ⟨*overlay specification*⟩ is given, the contents of the environment is inserted into the text only on the specified slides.

Example:

```
\frame
{
  This line is always shown.
  \begin{onlyenv}<2>
    This line is inserted on slide 2.
  \end{onlyenv}
}
```

`\begin{uncoverenv}`<⟨*overlay specification*⟩>
⟨*environment contents*⟩
`\end{uncoverenv}`

If the ⟨*overlay specification*⟩ is given, the contents of the environment is shown only on the specified slides. It still occupies space on the other slides.

Example:

```
\frame
{
  This word is
  \begin{uncoverenv}<2>
    visible
  \end{uncoverenv}
  only on slide 2.
}
```

4.2.4 Dynamically Changing Text

You may sometimes wish to have some part of a frame change dynamically from slide to slide. On each slide of the frame, something different should be shown inside this area. You could achieve the effect of dynamically changing text by giving a list of `\only` commands like this:

```
\only<1>{Initial text.}
\only<2>{Replaced by this on second slide.}
\only<3>{Replaced again by this on third slide.}
```

The trouble with this approach is that it may lead to slight, but annoying differences in the heights of the lines, which may cause the whole frame to "whobble" from slide to slide. This problem becomes much more severe if the replacement text is several lines long.

To solve this problem, you can use two environments: `overlayarea` and `overprint`. The first is more flexible, but less user-friendly.

`\begin{overlayarea}{`⟨*area width*⟩`}{`⟨*area height*⟩`}`
⟨*environment contents*⟩
`\end{overlayarea}`

Everything within the environment will be placed in a rectangular area of the specified size. The area will have the same size on all slides of a frame, regardless of its actual contents.

Example:

```
\begin{overlayarea}{\textwidth}{3cm}
  \only<1>{Some text for the first slide.\\Possibly several lines long.}
  \only<2>{Replacement on the second slide.}
\end{overlayarea}
```

`\begin{overprint}[`⟨*area width*⟩`]`
⟨*environment contents*⟩
`\end{overprint}`

The ⟨*area width*⟩ defaults to the text width. Inside the environment, use `\onslide` commands to specify different things that should be shown for this environment on different slides. The `\onslide` commands are used like `\item` commands. Everything within the environment will be placed in a rectangular area of the specified width. The height and depth of the area are chosen large enough to accommodate the largest contents of the area. The overlay specifications of the `\onslide` commands must be disjoint. This may be a problem for handouts, since, there, all overlay specifications defaul to `1`. If you use the option `handout`, you can disable all but one `\onslide` by setting the others to `0`.

Example:

```
\begin{overprint}
  \onslide<1| handout:1>
    Some text for the first slide.\\
    Possibly several lines long.
  \onslide<2| handout:0>
    Replacement on the second slide. Supressed for handout.
\end{overprint}
```

4.3 Making Commands and Environments Overlay-Specification-Aware

This subsection explains how you can make your own commands overlay-specification-aware. Also, it explains how to setup counters correctly that should be increased from frame to frame (like equation numbering), but not from slide to slide. You may wish to skip this section, unless you want to write your own extensions to the BEAMER class.

You can define a new command that is overlay-specification-aware using the following command.

`\newoverlaycommand{`⟨*command name*⟩`}{`⟨*default text*⟩`}{`⟨*alternative text*⟩`}`

Declares the new command named ⟨*command name*⟩. If this command is encountered by TEX, it is checked whether an overlay specification follows. If not, the ⟨*default text*⟩ is inserted. If there is a specification, the ⟨*default text*⟩ is also inserted if the current slide is specified, otherwise the ⟨*alternative text*⟩ is inserted.

Example:

```
\newoverlaycommand{\SelectRedAsColor}{\color[rgb]{1,0,0}}{}

\frame
{
  \SelectRedAsColor<2>
  The second slide of this frame is all in red.
}
```

\renewoverlaycommand{⟨*existing command name*⟩}{⟨*default text*⟩}{⟨*alternative text*⟩}

Redeclares a command that already exists in the same way as `\newoverlaycommand`. Inside the parameters, you can still access to original definitions using the command `\beameroriginal`, see the example.

Example:

```
\renewoverlaycommand{\tiny}{\beameroriginal{\tiny}}{}

\frame
{
  \tiny<2>This text is tiny on slide 2.
}
```

\newoverlayenvironment{⟨*environment name*⟩}[⟨*parameter number*⟩]{⟨*default begin*⟩}{⟨*default end*⟩}
{⟨*alternative begin*⟩}{⟨*alternative end*⟩}

Declares a new environment that is overlay specification aware. If this environment encountered, it is checked whether an overlay specification follows. If not or if it is found and the current slide is specified, the default begin and end are used. Otherwise, the alternative begin and end are used.

If the ⟨*parameter number*⟩ is specified, it must currently be 1. In this case, the begin commands must take one parameter. This parameter will *preceed* the overlay specification, see the examples.

Example:

```
\newoverlayenvironment{mytheorem}{\alert{Theorem}:}{}{Theorem:}{}

\frame
{
  \begin{mytheorem}<2>
    This theorem is hilighted on the second slide.
  \end{mytheorem}
}

\newoverlayenvironment{mytheorem}[1]{\alert{Theorem #1}:}{}{Theorem #1:}{}

\frame
{
  \begin{mytheorem}{of Tantau}<2>
    This theorem is hilighted on the second slide.
  \end{mytheorem}
}
```

The following two commands can be used to ensure that a certain counter is automatically reset on subsequent slides of a frame. This is necessary for example for the equation count. You might want this count to be increased from frame to frame, but certainly not from overlay slide to overlay slide. For equation counters and footnote counters (you should not use footnotes), these commands have already been invoked.

\resetcounteronoverlays{⟨*counter name*⟩}

After you have invoked this command, the value of the specified counter will be the same on all slides of every frame.

Example: `\resetcounteronoverlays{equation}`

\resetcountonoverlays{⟨*count register name*⟩}

The same as `\resetcounteronoverlays`, except that this command should be used with counts that have been created using the TEX primitive `\newcount` instead of LATEX's `\definecounter`.

Example:

```
\newcount\mycount
\resetcountonoverlays{mycount}
```

5 Structuring a Presentation

5.1 Global Structure of Presentations

Ideally, during most presentations you would like to present your slides in a perfectly linear fashion, presumably by pressing the page-down-key once for each slide. However, there are different reasons why you might have to deviate from this linear order:

- Your presentation may contain "different levels of detail" that may or may not be skipped or expanded, depending on the audience's reaction.
- You are asked questions and wish to show supplementary slides.
- You are asked questions about an earlier slide, which forces you to find and then jump to that slide.

You cannot really prepare against the last kind of questions. In this case, you can use the navigation bars and symbols to find the slide you are interested in, see 5.3.

Concerning the first two kinds of deviations, the BEAMER class offers several ways of preparing such "planned detours" or "planned short cuts".

- You can easily add predefined "skip buttons." When such a button is pressed, you jump over a well-defined part of your talk. Skip button have two advantages over just pressing the forward key is rapid succession: first, you immediately end up at the correct position and, second, the button's label can give the audience a visual feedback of what exactly will be skipped. For example, when you press a skip button labeled "Skip proof" nobody will start puzzling over what he or she has missed.
- You can add an appendix to your talk. The appendix is kept "perfectly separated" from the main talk. Only once you "enter" the appendix part (presumably by hyperjumping into it), does the appendix structure become visible. You can put all frames that you do not intend to show during the normal course of your talk, but which you would like to have handy in case someone asks, into this appendix.
- You can add "goto buttons" and "return buttons" to create detours. Pressing a goto button will jump to a certain part of the presentation where extra details can be shown. In this part, there is a return button present on each slide that will jump back to the place where the goto button was pressed.
- You can use the `\resumeframe` command to "continue" frames that you previously started somewhere, but where certain details have been suppressed. You can use the `\resumeframe` command at a much later point, for example only in the appendix to show to additional slides there.

5.2 Commands for Creating the Global Structure

5.2.1 Adding a Title Page

You can use the `\titlepage` command to insert a title page into a frame.

The `\titlepage` command will arrange the following elements on the title page: the document title, the author(s)'s names, their affiliation, a title graphic, and a date.

`\titlepage`

Inserts the text of a title page into the current frame.

Example: `\frame{\titlepage}`

Before you invoke the title page command, you must specify all elements you wish to be shown. This is done using the following commands:

`\title[`⟨*short title*⟩`]{`⟨*title*⟩`}`

The ⟨*short tile*⟩ is used in head lines and foot lines. Inside the ⟨*title*⟩ line breaks can be inserted using the double-backslash command.

Example:

```
\title{The Beamer Class}
\title[Short Version]{A Very Long Title\\Over Several Lines}
```

\author[⟨*short author names*⟩]{⟨*author names*⟩}

The names should be separated using the command \and. In case authors have different affiliations, they should be suffixed by the command \inst with different parameters.

Example: \author[Hemaspaandra et al.]{L. Hemaspaandra\inst{1} \and T. Tantau\inst{2}}

\institute[⟨*short institute*⟩]{⟨*institute*⟩}

If more than one institute is given, they should be separated using the command \and and they should be prefixed by the command \inst with different parameters.

Example:

```
\institute[Universities of Rochester and Berlin]{
  \inst{1}Department of Computer Science\\
  University of Rochester
  \and
  \inst{2}Fakult\"at f\"ur Elektrotechnik und Informatik\\
  Technical University of Berlin}
```

\date[⟨*short date*⟩]{⟨*date*⟩}

Example: \date{\today} or \date[STACS 2003]{STACS Conference, 2003}.

\titlegraphic{⟨*text*⟩}

The ⟨*text*⟩ is shown as title graphic. Typically, a picture environment is used as ⟨*text*⟩.

Example: \titlegraphic{\pgfuseimage{titlegraphic}}

5.2.2 Adding Sections and Subsections

You can structure your text using the commands \section and \subsection. Unlike standard LaTeX, these commands will not create a heading at the position where you use them. Rather, they will add an entry to the table of contents and also to the navigation bars.

In order to create a line break in the table of contents (usually not a good idea), you can use the command \breakhere. Note that the standard command \\ does not work.

\section[⟨*short section name*⟩]{⟨*section name*⟩}

Starts a section. No heading is created. The ⟨*section name*⟩ is shown in the table of contents and in the navigation bars, except if ⟨*short section name*⟩ is specified. In this case, ⟨*short section name*⟩ is used in the navigation bars instead.

Example: \section[Summary]{Summary of Main Results}

\section*{⟨*section name*⟩}

Starts a section without an entry in the table of contents. No heading is created, but the ⟨*section name*⟩ is shown in the navigation bar.

Example: \section*{Outline}

\subsection[⟨*short subsection name*⟩]{⟨*subsection name*⟩}

This command works the same way as the \section command.

Example: \subsection[Applications]{Applications to the Reduction of Pollution}

\subsection*{⟨*subsection name*⟩}

Like \subsection*, except for subsections.

Example: \subsection*{Further Reading}

Often, you may want a certain type of frame to be shown directly after a section or subsection starts. For example, you may wish every subsection to start with a frame showing the table of contents with the current subsection hilighted. To facilitate this, you can use the following two commands.

`\AtBeginSection[`⟨*special star text*⟩`]{`⟨*text*⟩`}`

The given text will be inserted at the beginning of every section. If the ⟨*special star text*⟩ parameter is specified, this text will be used for starred sections instead. Different calls of this command will not "add up" the given texts (like the `\AtBeginDocument` command does), but will overwrite any previous text.

Example:

```
\AtBeginSection[] % Do nothing for \section*
{
  \frame<handout:0>
  {
    \frametitle{Outline}
    \tableofcontents[current]
  }
}
```

`\AtBeginSubsection[`⟨*special star text*⟩`]{`⟨*text*⟩`}`

The given text will be inserted at the beginning of every subsection. If the ⟨*special star text*⟩ parameter is specified, this text will be used for starred subsections instead. Different calls of this command will not "add up" the given texts.

Example:

```
\AtBeginSubsection[] % Do nothing for \subsection*
{
  \frame<handout:0>
  {
    \frametitle{Outline}
    \tableofcontents[current,currentsubsection]
  }
}
```

5.2.3 Adding Parts

If you give a long talk (like a lecture), you may wish to break up your talk into several parts. Each such part acts like a little "talk of its own" with its own table of contents, its own navigation bars, and so on. Inside one part, the sections and subsections of the other parts are not shown at all.

To create a new part, use the `\part` command. All sections and subsections following this command will be "local" to that part. Like the `\section` and `\subsection` command, the `\part` command does not cause any frame or special text to be produced. However, it is often advisable for the start of a new part to use the command `\partpage` to insert some text into a frame that "advertises" the beginning of a new part. See `beamerexample3.tex` for an example.

`\part[`⟨*short part name*⟩`]{`⟨*part name*⟩`}`

Starts a part. The ⟨*part name*⟩ will be shown when the `\partpage` command is used. The ⟨*shown part name*⟩ is not shown anywhere by default, but it is accessible via the command `\insertshortpart`.

Example:

```
\begin{document}
  \frame{\titlepage}

  \section*{Outlines}
  \subsection{Part I: Review of Previous Lecture}
  \frame{
    \frametitle{Outline of Part I}
    \tableofcontents[part=1]}
  \subsection{Part II: Today's Lecture}
  \frame{
    \frametitle{Outline of Part II}
    \tableofcontents[part=2]}

  \part{Review of Previous Lecture}
```

```
  \frame{\partpage}
  \section[Previous Lecture]{Summary of the Previous Lecture}
  \subsection{Topics}
  \frame{...}
  \subsection{Learning Objectives}
  \frame{...}

  \part{Today's Lecture}
  \frame{\partpage}
  \section{Topic A}
  \frame{\tableofcontents[current]}
  \subsection{Foo}
  \frame{...}
  \section{Topic B}
  \frame{\tableofcontents[current]}
  \subsection{bar}
  \frame{...}
\end{document}
```

\partpage

Works like \titlepage, only that the current part, not the current presentation is "advertised." The appearance can be changed by adjusting the part page template, see Section 8.4.3.

Example: \frame{\partpage}

\AtBeginPart{⟨*text*⟩}

The given text will be inserted at the beginning of every part.

Example:

```
\AtBeginPart{\frame{\partpage}}
```

5.2.4 Adding a Table of Contents

You can create a table of contents using the command \tableofcontents. Unlike the normal LaTeX table of contents command, this command takes an optional parameter in square brackets that can be used to create certain special effects.

\tableofcontents[⟨*comma-separated option list*⟩]

Inserts a table of contents into the current frame. To change how the table of contents is typeset, you need to modify the appropriate templates, see Section 8.4.5.

Example:

```
\section*{Outline}
\frame{\tableofcontents}

\section{Introduction}
\frame{\tableofcontents[current]}
\subsection{Why?}
\frame{...}
\frame{...}
\subsection{Where?}
\frame{...}

\section{Results}
\frame{\tableofcontents[current]}
\subsection{Because}
\frame{...}
\subsection{Here}
\frame{...}
```

The following options can be given:

- `part=`⟨*part number*⟩ causes the table of contents of part ⟨*part number*⟩ to be shown, instead of the table of contents of the current part (which is the default). This option can be combined with the other options, although combining it with the `current` option obviously makes no sense.
- `sections={`⟨*overlay specification*⟩`}` causes only the sections mentioned in the ⟨*overlay specification*⟩ to be shown. For example, `sections={<2-4| handout:0>}` causes only the second, third, and fourth section to be shown in the normal version, nothing to be shown in the handout version, and everything to be shown in all other versions. For convenience, if you omit the pointed brackets, the specification is assumed to apply to all versions. Thus `sections={2-4}` causes sections two, three, and four to be shown in all versions.
- `firstsection=`⟨*section number*⟩ specifies which section should be numbered as section "1." This is useful if you have a first section (like an overview section) that should not receive a number. Section numbers are not shown by default. To show them, you must install a different table of contents templates.
- `current` causes all sections but the current to be shown in a semi-transparent way. Also, all subsections but those in the current section are shown in the semi-transparent way.
- `currentsubsection` causes all subsections but the current subsection in the current section to be shown in a semi-transparent way.
- `pausesections` causes a `\pause` command to be issued before each section. This is useful if you wish to show the table of contents in an incremental way.
- `pausesubsections` causes a `\pause` command to be issued before each subsection.
- `hidesubsections` causes the subsections to be omitted. However, if used together with the `current` option, the subsections of the current section are not omitted.
- `shadesubsections` causes the subsections to be shown in a semi-transparent way.

The last two commands are useful if you do not wish to show too many details when presenting the talk outline.

5.2.5 Adding a Bibliography

You can use the bibliography environment and the `\cite` commands of LaTeX in a BEAMER presentation. However, there are a few things to keep in mind:

- It is a bad idea to present a long bibliography in a beamer presentation. Present only very few references.
- Present references only if they are intended as "further reading," for example at the end of a lecture.
- Using the `\cite` commands can be confusing since the audience has little chance of remembering the citations. If you cite the references, always cite them with full author name and year like "[Tantau, 2003]" instead of something like "[2,4]" or "[Tan01,NT02]".
- If you want to be modest, you can abbreviate your name when citing yourself as in "[Nickelsen and T., 2003]" or "[Nickelsen and T, 2003]". However, this can be confusing for the audience since it is often not immediately clear who exactly "T." might be. I recommand using the full name.

Keeping the above warnings in mind, proceed as follows to create the bibliography:

For a beamer presentation, you will typically have to typeset your bibliography items partly "by hand." Nevertheless, you *can* use `bibtex` to create a "first approximation" of the bibliography. Copy the content of the file `main.bbl` into your presentation. If you are not familiar with `bibtex`, you may wish to consult its documentation. It is a powerful tool for creating high-quality citations.

Using `bibtex` or your editor, place your bibliographic references in the environment `thebibliography`. This (standard LaTeX) environment takes one parameter, which should be the longest `\bibitem` label in the following list of bibliographic entries.

`\begin{thebibliography}{`⟨*longest label text*⟩`}`
⟨*environment contents*⟩

\end{thebibliography}

Inserts a bibliography into the current frame. The ⟨*longest label text*⟩ is used to determine the indent of the list. However, several templates for the typesetting of the bibliography (see Section 8.4.6) ignore this parameter since they replace the references by a symbol.

Inside the environment, use a (standard LaTeX) `\bibitem` command for each reference item. Inside each item, use a (standard LaTeX) `\newblock` command to separate the authors's names, the title, the book/journal reference, and any notes. Each of these commands may introduce a new line or color or other formatting, as specified by the template for bibliographies.

The environment must be placed inside a frame. If the bibliography does not fit on one frame, you should split it (create a new frame and a second `thebibliography` environment). Even better, you should reconsider whether it is a good idea to present so many references.

Example:

```
\frame{
  \frametitle{For Further Reading}

  \begin{thebibliography}{Dijkstra, 1982}
  \bibitem[Solomaa, 1973]{Solomaa1973}
    A.~Salomaa.
    \newblock {\em Formal Languages}.
    \newblock Academic Press, 1973.

  \bibitem[Dijkstra, 1982]{Dijkstra1982}
    E.~Dijkstra.
    \newblock Smoothsort, an alternative for sorting in situ.
    \newblock {\em Science of Computer Programming}, 1(3):223--233, 1982.
  \end{thebibliography}
 }
```

\bibitem<⟨*overlay specification*⟩>[⟨*citation text*⟩]{⟨*label name*⟩}

The ⟨*citation text*⟩ is inserted into the text when the item is cited using `\cite{`⟨*label name*⟩`}` in the main presentation text. For a BEAMER presentation, this should usually be as long as possible.

Use `\newblock` commands to separate the authors's names, the title, the book/journal reference, and any notes. If the ⟨*overlay specification*⟩ is present, the entry will only be shown on the specified slides.

Example:

```
\bibitem[Dijkstra, 1982]{Dijkstra1982}
  E.~Dijkstra.
  \newblock Smoothsort, an alternative for sorting in situ.
  \newblock {\em Science of Computer Programming}, 1(3):223--233, 1982.
```

Unlike normal LaTeX, the default template for the bibliography does not repeat the citation text (like "[Dijkstra, 1982]") before each item in the bibliography. Instead, a cute, small article symbol is drawn. The rationale is that the audience will not be able to remember any abbreviated citation texts till the end of the talk. If you really insist on using abbreviations, you can use the command `\beamertemplatetextbibitems` to restore the default bevahior, see also Section 8.4.6.

5.2.6 Adding an Appendix

You can add an appendix to your talk by using the `\appendix` command. You should put frames and perhaps whole subsections into the appendix that you do not intend to show during your presentation, but which might be useful to answer a question. The `\appendix` command essentially just start a new part named `\appendixname`. However, it also sets up certain hyperlinks. Like other parts, the appendix is kept separate of your actual talk.

\appendix

Starts the appendix. All frames, all `\subsection` commands, and all `\section` commands used after this command will not be shown as part of the normal navigation bars.

Example:

```
\begin{document}
\frame{\titlepage}
\section*{Outline}
\frame{\tableofcontents}
\section{Main Text}
\frame{Some text}
\section*{Summary}
\frame{Summary text}

\appendix
\section{\appendixname}
\frame{\tableofcontents}
\subsection{Additional material}
\frame{Details}
\frame{Text omitted in main talk.}
\subsection{Even more additional material}
\frame{More details}
\end{document}
```

5.2.7 Adding Hyperlinks and Buttons

To create an anticipated nonlinear jumps in your talk structure, you can add hyperlinks to your presentation. A hyperlink is a text (usually rendered as a button) that, when you click on it, jumps the presentation to some other slide. Creating such a button is a three-step process:

1. You specify a target using the command `\hypertarget`. In some cases, see below, this step may be skipped.
2. You render the button using `\beamerbutton` or a similar command. This will *render* the button, but clicking it will not yet have any effect.
3. You put the button inside a `\hyperlink` command. Now clicking it will jump to the target of the link.

`\hypertarget<`⟨*overlay specification*⟩`>{`⟨*target name*⟩`}{`⟨*text*⟩`}`

If the ⟨*overlay specification*⟩ is present, the ⟨*text*⟩ is the target for hyper jumps to ⟨*target name*⟩ only on the specified slide. On all other slides, the text is shown normally. Note that you *must* add an overlay specification to the `\hypertarget` command whenever you use it on frames that have multiple slides (otherwise `pdflatex` rightfully complains that you have defined the same target on different slides).

Example:

```
\frame{
  \begin{itemize}
  \item<1-> First item.
  \item<2-> Second item.
  \item<3-> Third item.
  \end{itemize}

  \hyperlink{jumptosecond}{\beamergotobutton{Jump to second slide}}
  \hypertarget<2>{jumptosecond}{}
}
```

The following commands can be used to specify in an abstract way what a button will be used for. How exactly these buttons are rendered is governed by a template, see Section 8.4.10.

`\beamerbutton{`⟨*button text*⟩`}`

Draws a button with the given ⟨*button text*⟩.

Example: `\hyperlink{somewhere}{\beamerbutton{Go somewhere}}`

`\beamergotobutton{`⟨*button text*⟩`}`

Draws a button with the given ⟨*button text*⟩. Before the text, a small symbol (usually a right-pointing arrow) is inserted that indicates that pressing this button will jump to another "area" of the presentation.

Example: `\hyperlink{detour}{\beamergotobutton{Go to detour}}`

\beamerskipbutton{⟨*button text*⟩}

The symbol drawn for this button is usually a double right arrow. Use this button if pressing it will skip over a well-defined part of your talk.

Example:

```
\frame{
  \begin{theorem}
    ...
  \end{theorem}

  \begin{overprint}
  \onslide<1>
    \hfill\hyperlinkframestartnext{\beamerskipbutton{Skip proof}}
  \onslide<2>
    \begin{proof}
      ...
    \end{proof}
  \end{overprint}
}
```

\beamerreturnbutton{⟨*button text*⟩}

The symbol drawn for this button is usually a left pointing arrow. Use this button if pressing it will return from a detour.

Example:

```
\frame{
  \begin{theorem}
    ...
  \end{theorem}

  \hyperlink{detour}{\beamergotobutton{Go to proof details}}
  \hypertarget{returnhere}{}
}
\appendix
\frame{
  \hypertarget{detour}{}
  \begin{proof}
    ...
  \end{proof}
  \hfill\hyperlink{returnhere}{\beamerreturnbutton{Return}}
}
```

To make a button "clickable" you must place it in a command like `\hyperlink`. The command `\hyperlink` is a standard command of the `hyperref` package. The BEAMER class defines a whole bunch of other hyperlink commands that you can also use.

\hyperlink{⟨*target name*⟩}{⟨*link text*⟩}

The ⟨*link text*⟩ is typeset in the usual way. If you click anywhere on this text, you will jump to the slide on which the `\hypertarget` command was used with the parameter ⟨*target name*⟩.

The following commands have a predefined target; otherwise they behave exactly like `\hyperlink`.

\hyperlinkslideprev{⟨*link text*⟩}

Clicking the text jumps one slide back.

\hyperlinkslidenext{⟨*link text*⟩}

Clicking the text jumps one slide forward.

\hyperlinkframestart{⟨*link text*⟩}

Clicking the text jumps to the first slide of the current frame.

`\hyperlinkframeend{`⟨*link text*⟩`}`

Clicking the text jumps to the last slide of the current frame.

`\hyperlinkframestartnext{`⟨*link text*⟩`}`

Clicking the text jumps to the first slide of the next frame.

`\hyperlinkframeendprev{`⟨*link text*⟩`}`

Clicking the text jumps to the last slide of the previous frame.

The previous four command exist also with "`frame`" replaced by "`subsection`" everywhere, and also again with "`frame`" replaced by "`section`".

`\hyperlinkpresentationstart{`⟨*link text*⟩`}`

Clicking the text jumps to the first slide of the presentation.

`\hyperlinkpresentationend{`⟨*link text*⟩`}`

Clicking the text jumps to the last slide of the presentation. This *excludes* the appendix.

`\hyperlinkappendixstart{`⟨*link text*⟩`}`

Clicking the text jumps to the first slide of the appendix. If there is no appendix, this will jump to the last slide of the document.

`\hyperlinkappendixend{`⟨*link text*⟩`}`

Clicking the text jumps to the last slide of the appendix.

`\hyperlinkdocumentstart{`⟨*link text*⟩`}`

Clicking the text jumps to the first slide of the presentation.

`\hyperlinkdocumentend{`⟨*link text*⟩`}`

Clicking the text jumps to the last slide of the presentation or, if an appendix is present, to the last slide of the appendix.

5.3 Navigation Bars and Symbols

Navigation bars and symbols are two independent concepts that can be used to navigate through a presentation. They are created automatically.

5.3.1 Using the Navigation Bars

Most themes that come along with the BEAMER class show some kind of navigation bar during your talk. Although these navigation bars take up quite a bit of space, they are often useful for two reasons:

- They provide the audience with a visual feedback of how much of your talk you have covered and what is yet to come. Without such feedback, an audience will often puzzle whether something you are currently introducing will be explained in more detail later on or not.
- You can click on all parts of the navigation bar. This will directly "jump" you to the part you have clicked on. This is particularly useful to skip certain parts of your talk and during a "question session," when you wish to jump back to a particular frame someone has asked about.

Some navigation bars can be "compressed" using the following option:

`\documentclass[compress]{beamer}`

Tries to make all navigation bars as small as possible. For example, all small frame representations in the navigation bars for a single section are shown alongside each other. Normally, the representations for different subsections are shown in different lines. Furthermore, section and subsection navigations are compressed into one line.

When you click on one of the icons representing a frame in a navigation bar (by default this is icon is a small circle), the following happens:

- If you click on (the icon of) any frame other than the current frame, the presentation will jump to the first slide of the frame you clicked on.
- If you click on the current frame and you are not on the last slide of this frame, you will jump to the last slide of the frame.
- If you click on the current frame and you are on the last slide, you will jump to the first slide of the frame.

By the above rules you can:

- Jump to the beginning of a frame from somewhere else by clicking on it once.
- Jump to the end of a frame from somewhere else by clicking on it twice.
- Skip the rest of the current frame by clicking on it once.

I also tried making a jump to an already-visited frame jump automatically to the last slide of this frame. However, this turned out to be more confusing than helpful. With the current implementation a double-click always brings you to the end of a slide, regardless from where you "come."

By clicking on a section or subsection in the navigation bar, you will jump to that section. Clicking on a section is particularly useful if the section starts with a `\tableofcontents[current]`, since you can use it to jump to the different subsections.

By clicking on the document title in a navigation bar (not all themes show it), you will jump to the first slide of your presentation (usually the title page) *except* if you are already at the first slide. On the first slide, clicking on the document title will jump to the end of the presentation, if there is one. Thus by *double* clicking the document title in a navigation bar, you can jump to the end.

5.3.2 Using the Navigation Symbols

Navigation symbols are small icons that are shown on every slide by default. The following symbols are shown:

1. A slide icon, which is depicted as a single rectangle. To the left and right of this symbol, a left and right arrow are shown.
2. A frame icon, which is depicted as three slide icons "stacked on top of each other". This symbols is framed by arrows.
3. A subsection icon, which is depicted as a highlighted subsection entry in a table of contents. This symbols is framed by arrows.
4. A section icon, which is depicted as a highlighted section entry (together with all subsections) in a table of contents. This symbols is framed by arrows.
5. A presentation icon, which is depicted as a completely highlighted table of contents.
6. An appendix icon, which is depicted as a completely highlighted table of contents consisting of only one section. (This icon is only shown if there is an appendix.
7. Back and forward icons, depicted as circular arrows.
8. A "search" or "find" icon, depicted as a detective's magnifying glass.

Clicking on the left arrow next to an icon always jumps to (the last slide of) the previous slide, frame, subsection, or section. Clicking on the right arrow next to an icon always jump to (the first slide of) the next slide, frame, subsection, or section.

Clicking *on* any of these icons has different effects:

1. If supported by the viewer application, clicking on a slide icon pops up a window that allows you to enter a slide number to which you wish to jump.

2. Clicking on the left side of a frame icon will jump to the first slide of the frame, clicking on the right side will jump to the last slide of the frame (this can be useful for skipping overlays).

3. Clicking on the left side of a subsection icon will jump to the first slide of the subsection, clicking on the right side will jump to the last slide of the subsection.

4. Clicking on the left side of a section icon will jump to the first slide of the section, clicking on the right side will jump to the last slide of the section.

5. Clicking on the left side of the presentation icon will jump to the first slide, clicking on the right side will jump to the last slide of the presentation. However, this does *not* include the appendix.

6. Clicking on the left side of the appendix icon will jump to the first slide of the appendix, clicking on the right side will jump to the last slide of the appendix.

7. If supported by the viewer application, clicking on the back and forward symbols jumps to the previously visited slides.

8. If supported by the viewer application, clicking on the search icon pops up a window that allows you to enter a search string. If found, the viewer application will jump to this string.

You can reduce the number of icons that are shown or their layout by adjusting the navigation symbols template, see Section 8.4.12.

5.4 Command for Creating the Local Structure

Just like your whole presentation, each frame should also be structured. A frame that is solely filled with some long text is very hard to follow. It is your job to structure the contents of each frame such that, ideally, the audience immediately seems which information is important, which information is just a detail, how the presented information is related, and so on.

LaTeX provides different commands for structuring text "locally," for example, via the `itemize` environment. These environments are also available in the beamer class, although their appearance has been slightly changed. Furthermore, the BEAMER class also defines some new commands and environments, see below, that may help you to structure your text.

5.4.1 Itemizations, Enumerations, and Descriptions

There are three predefined environments for creating lists, namely `enumerate`, `itemize`, and `description`. The first two can be nested to depth two, but not further (this would create totally unreadable slides).

The `\item` command is overlay-specification-aware. If an overlay specification is provided, the item will only be shown on the specified slides, see the following example. If the `\item` command is to take an optional argument and an overlay specification, the overlay specification can either come first as in `\item<1>[Cat]` or come last as in `\item[Cat]<1>`.

```
\frame
{
  There are three important points:
  \begin{enumerate}
  \item<1-> A first one,
  \item<2-> a second one with a bunch of subpoints,
    \begin{itemize}
    \item first subpoint. (Only shown from second slide on!).
    \item<3-> second subpoint added on third slide.
    \item<4-> third subpoint added on fourth slide.
    \end{itemize}
  \item<5-> and a third one.
  \end{enumerate}
}
```

`\begin{itemize}`
⟨*environment contents*⟩

\end{itemize}

Used to display a list of items that do not have a special ordering. Inside the environment, use an \item command for each topic. The appearence of the items can be changed using templates, see Section 8.4.

Example:

```
\begin{itemize}
\item This is important.
\item This is also important.
\end{itemize}
```

\begin{enumerate}
⟨*environment contents*⟩
\end{enumerate}

Like itemize, except that the list is ordered.

Example:

```
\begin{enumerate}
\item This is important.
\item This is also important.
\end{enumerate}
```

\begin{description}[⟨*long text*⟩]
⟨*environment contents*⟩
\end{description}

Like itemize, but used to display an list that explains or defines labels. The width of ⟨*long text*⟩ is used to set the indent. Normally, you choose the widest label in the description and copy it here.

Example:

```
\begin{description}
\item[Lion] King of the savanna.
\item[Tiger] King of the jungle.
\end{description}

\begin{description}[longest label]
\item<1->[short] Some text.
\item<2->[longest label] Some text.
\item<3->[long label] Some text.
\end{description}
```

\usedescriptionitemofwidthas{⟨*long text*⟩}

This command overrides the default width of the description label by the width of ⟨*long text*⟩ for the current TEX group. You should only use this command if, for some reason or another, you cannot give the ⟨*long text*⟩ as an argument to the description environment. This happens, for example, if you create a description environment in LYX.

Example:

```
\usedescriptionitemofwidthas{longest label}
\begin{description}
\item<1->[short] Some text.
\item<2->[longest label] Some text.
\item<3->[long label] Some text.
\end{description}
```

5.4.2 Block Environments and Simple Structure Commands

The BEAMER class predefines a number of useful environments and commands. Using these commands makes is easy to change the appearance of a document by changing the theme.

\alert<⟨*overlay specification*⟩>{⟨*hilighted text*⟩}

The given text is hilighted, typically be coloring the text red. If the ⟨*overlay specification*⟩ is present, the command only has an effect on the specified slides.

Example: `This is \alert{important}.`

\structure<⟨*overlay specification*⟩>{⟨*text*⟩}

The given text is marked as part of the structure, typically by coloring it in the `structure` color. If the ⟨*overlay specification*⟩ is present, the command only has an effect on the specified slides.

Example: `\structure{Paragraph Heading.}`

\begin{block}{⟨*block title*⟩}<⟨*overlay specification*⟩>
⟨*environment contents*⟩
\end{block}

Inserts a block, like a definition or a theorem, with the title ⟨*block title*⟩. If the ⟨*overlay specification*⟩ is present, the block is shown only on the specified slides. In the example, the definition is shown only from slide 3 onward.

Example:

```
\begin{block}{Definition}<3->
  A \alert{set} consists of elements.
\end{block}
```

\begin{alertblock}{⟨*block title*⟩}<⟨*overlay specification*⟩>
⟨*environment contents*⟩
\end{alertblock}

Inserts a block whose title is hilighted. If the ⟨*overlay specification*⟩ is present, the block is shown only on the specified slides.

Example:

```
\begin{alertblock}{Wrong Theorem}
  $1=2$.
\end{alertblock}
```

\begin{exampleblock}{⟨*block title*⟩}<⟨*overlay specification*⟩>
⟨*environment contents*⟩
\end{exampleblock}

Inserts a block that is supposed to be an example. If the ⟨*overlay specification*⟩ is present, the block is shown only on the specified slides.

Example:

```
\begin{exampleblock}{Example}
  The set $\{1,2,3,5\}$ has four elements.
\end{exampleblock}
```

Predefined English block environments, that is, block environments with fixed title, are: `Theorem`, `Proof`, `Corollary`, `Fact`, `Example`, and `Examples`. You can also use these environments with a lowercase first letter, the result is the same. The following German block environments are also predefined: `Problem`, `Loesung`, `Definition`, `Satz`, `Beweis`, `Folgerung`, `Lemma`, `Fakt`, `Beispiel`, and `Beispiele`. See the following example for their usage.

```
\frame
{
  \frametitle{A Theorem on Infinite Sets}

  \begin{theorem}<1->
    There exists an infinite set.
  \end{theorem}

  \begin{proof}<2->
    This follows from the axiom of infinity.
  \end{proof}

  \begin{example}<3->
    The set of natural numbers is infinite.
  \end{example}
}
```

5.4.3 Framed Text

In order to draw a frame (a rectangle) around some text, you can use LaTeXs standard command `\fbox`. More frame types are offered by the package `fancybox`, which defines the following commands: `\shadowbox`, `\doublebox`, `\ovalbox`, and `\Ovalbox`. Please consult the LaTeX Companion for details on how to use these commands.

The BEAMER class also defines an environment for creating boxes:

`\begin{beamerboxesrounded}[`⟨*options*⟩`]{`⟨*head*⟩`}`
⟨*environment contents*⟩
`\end{beamerboxesrounded}`

The text inside the environment is framed by a rectangular area with rounded corners. The background of the rectangular area is filled with a certain color, which depends on the current color scheme (see below). If the ⟨*head*⟩ is not empty, ⟨*head*⟩ is drawn in the upper part of the box in a different color, which also depends on the scheme. The following options can be given:

- `scheme=`⟨*name*⟩ causes the color scheme ⟨*name*⟩ to be used. A color scheme must previously be defined using the command `\beamerboxesdeclarecolorscheme`.
- `width=`⟨*dimension*⟩ causes the width of the text inside the box to be the specified ⟨*dimension*⟩. By default, the `\textwidth` is used. Note that the box will protrude 4pt to the left and right.
- `shadow=`⟨*true or false*⟩. If set to `true`, a shadow will be drawn.

A color scheme dictates the background colors used in the head part and in the body of the box. If no ⟨*head*⟩ is given, the head part is completely suppressed.

Example:

```
\begin{beamerboxesrounded}[scheme=alert,shadow=true]{Theorem}
  $A = B$.
\end{beamerboxesrounded}
```

`\beamerboxesdeclarecolorscheme{`⟨*scheme name*⟩`}{`⟨*head color*⟩`}{`⟨*body color*⟩`}`

Declares a color scheme for later use in a `beamerboxesrounded` environment.

Example: `\beamerboxesdeclarecolorscheme{alert}{red}{red!15!averagebackgroundcolor}`

5.4.4 Figures and Tables

You can use the standard LaTeX environments `figure` and `table` much the same way you would normally use them. However, any placement specification will be ignored. Figures and tables are immediately inserted where the environments start. If there are too many of them to fit on the frame, you must manually split them among additional frames.

Example:

```
\frame{
  \begin{figure}
    \pgfuseimage{myfigure}
    \caption{This caption is placed below the figure.}
  \end{figure}

  \begin{figure}
    \caption{This caption is placed above the figure.}
    \pgfuseimage{myotherfigure}
  \end{figure}
}
```

You can adjust how the figure and table captions are typeset by changing the corresponding template, see Section 8.4.14.

5.4.5 Splitting a Frame into Multiple Columns

Three environments are used to create columns on a slide. Columns are especially useful for placing a graphic next to a description/explanation. The main environment for creating columns is called `columns`. Inside this environment, you can place several `column` environments. Each will create a new column.

`\begin{columns}`[⟨*options*⟩]
⟨*environment contents*⟩
`\end{columns}`

A multi-column area. Inside the environment you should place only `column` environments or `\column` commands (see below). The following ⟨*options*⟩ may be given:

- `b` will cause the bottom lines of the columns to be vertically aligned.
- `c` will cause the columns to be centered vertically relative to each other. Default, unless the global option `slidestop` is used.
- `onlytextwidth` is the same as `totalwidth=\textwidth`.
- `t` will cause the first lines of the columns to be aligned. Default if global option `slidestop` is used.
- `totalwidth=`⟨*width*⟩ will cause the columns to occupy not the whole page width, but only ⟨*width*⟩, all told.

Example:

```
\begin{columns}[t]
  \begin{column}{5cm}
    Two\\lines.
  \end{column}
  \begin{column}{5cm}
    One line (but aligned).
  \end{column}
\end{columns}
```

Example:

```
\begin{columns}[t]
  \column{5cm}
    Two\\lines.

  \column{5cm}
    One line (but aligned).
\end{columns}
```

To create a column, you can either use the `column` environment or the `\column` command.

`\begin{column}`[⟨*placement*⟩]`{`⟨*column width*⟩`}`
⟨*environment contents*⟩
`\end{column}`

Creates a single column of width ⟨*column width*⟩. The vertical placement of the enclosing `columns` environment can be overruled by specifying a specific ⟨*placement*⟩ (`t` for top, `c` for centered, and `b` for bottom).

Example: The following code has the same effect as the above examples:

```
\begin{columns}
  \begin{column}[t]{5cm}
    Two\\lines.
  \end{column}
  \begin{column}[t]{5cm}
    One line (but aligned).
  \end{column}
\end{columns}
```

`\column[`⟨*placement*⟩`]{`⟨*column width*⟩`}`

Starts a single column. The parameters and options are the same as for the `column` environment. The column automatically ends with the next occurence of `\column` or of a `column` environment or of the end of the current `columns` environment.

Example:

```
\begin{columns}
  \column[t]{5cm}
    Two\\lines.
  \column[t]{5cm}
    One line (but aligned).
\end{columns}
```

6 Graphics, Colors, Animations, and Special Effects

6.1 Graphics

Graphics often convey concepts or ideas much more efficiently than text: A picture can say more than a thousand words. (Although, sometimes a word can say more than a thousand pictures.) In the following, the advantages and disadvantages of different possible ways of creating graphics for beamer presentations are discussed.

6.1.1 Including External Graphic Files

One way of creating graphics for a presentation is to use an external program, like `xfig` or the Gimp. These programs have an option to *export* graphic files in a format that can then be inserted into the presentation.

The main advantage is:

- You can use a powerful program to create a high-quality graphic.

The main disadvantages are:

- You have to worry about many files. Typically there are at least two for each presentation, namely the program's graphic data file and the exported graphic file in a format that can be read by TEX.
- Changing the graphic using the program does not automatically change the graphic in the presentation. Rather, you must reexport the graphic and rerun LATEX.
- It may be difficult to get the line width, fonts, and font sizes right.
- Creating formulas as part of graphics is often difficult or impossible.

You can use all the standard LATEX commands for inserting graphics, like `\includegraphics` (be sure to use the package `graphics`). Also, the `pgf` package offers commands for including graphics. Either will work fine in most situations, so choose whichever you like. Like `\pgfdeclareimage`, `\includegraphics` also includes an image only once in a `.pdf` file, even if it used several times (as a matter of fact, the `graphics` package is even a bit smarter about this than `pgf`). However, currently only `pgf` offers the ability to include images that are partly transparent.

There are few things to note about the format of graphics you can include:

- When using `latex` and `dvips`, you can only include external graphic files ending with the extension `.eps` (Encapsulated PostScript). This is true both for the normal `graphics` package and for `pgf`. When using `pgf`, do *not* add the extension `.eps`. When using `graphics`, do add the extension. If your graphic file has a different format (like a `.jpg` file), you must first convert it to an `.eps` file using some conversion program.
- When using `pdflatex`, you can only include external graphic files ending with one of the extensions `.pdf`, `.jpg`, `.jpeg`, or `.png`. As before, do not add these extension when using `pgf`, but do add them when using `graphics`. If your graphic file has a different format, you have to convert it.

6.1.2 Inlining Graphic Commands

A different way of creating graphics is to insert graphic drawing commands directly into your LaTeX file. There are numerous packages that help you do this. They have various degrees of sophistication. Inlining graphics suffers from none of the disadvantages mentioned above for including external graphic files, but the main disadvantage is that it is often hard to use these packages. In some sense, you "program" your graphics, which requires a bit of practice.

When choosing a graphic package, there are a few things to keep in mind:

- Many packages produce poor quality graphics. This is especially true of the standard `picture` environment of LaTeX.
- Powerful packages that produce high-quality graphics often do not work together with `pdflatex`.
- The most powerful and easiest-to-use package around, namely `pstricks`, does not work together with `pdflatex` and this is a fundamental problem. Due to the fundamental differences between PDF and PostScript, it is not possible to write a "`pdflatex` backend for `pstricks`."

A solution to the above problem (though not necessarily the best) is to use the PGF package. It produces high-quality graphics and works together with `pdflatex`, but also with normal `latex`. It is not as powerful as `pstricks` (as pointed out above, this is because of rather fundamental reasons) and not as easy to use, but it should be sufficient in most cases.

6.2 Color Management

The color management of the BEAMER class relies on the packages `xcolor`, which is an extension of the `color` package, and on `xxcolor`, which in turn is an extension of `xcolor`. Hopefully, in the future `xxcolor` and `xcolor` will merge into one package and perhaps they will someday also merge together with `color`.

6.2.1 Colors of Main Text Elements

By default, the following colors are used in a presentation:

- Normal text is typeset in `black`.
- All "structural" elements, like titles, navigation bars, block titles, and so on, are typeset using the color `structure`. By default, this color is bluish. Using one of the class options `red`, `blackandwhite`, or `brown` changes this. You can also change this color simply be redefining the color `structure`.
- All "alert" text is typeset by mixing in 85% of red. To change this, you can either redefine the color `alert`, or you can change the whole alert template.
- All examples are typeset using 50% of green. To change this, you must change the example templates.

`\documentclass[brown]{beamer}`

Changes the main color of the navigation and title bars to a brownish color.

`\documentclass[red]{beamer}`

Changes the main color of the navigation and title bars to a reddish color.

`\documentclass[blackandwhite]{beamer}`

Changes the main color of the navigation and title bars to monochrome.

6.2.2 Average Background Color

In some situations, for example when creating a transparency effect, it is useful to have access to the current background color. One can then, for example, mix a color with the background color to create a "transparent" color.

Unfortunately, it is not always clear what exactly the background color is. If the background is a shading or a picture, different parts of a slide have different background colors. In these cases, one can at least try to mix-in an *average* background color, called `averagebackgroundcolor`. If a shading or picture is not too colorful, this works fairly well.

To specify the average background color, use the following command:

\beamersetaveragebackground{⟨*color expression*⟩}

Installs the given color as the average background color. See the `xcolor` package for the syntax of color expressions.

Example: `\beamersetaveragebackground{red!10}`

If you use the commands from Section 8.4.4 for installing a background coloring, the average background color is computed automatically for you. When you directly use the command `\usebackgroundtemplate`, you should must set the average background color afterward.

6.2.3 Transparency Effects

By default, *covered* items are not shown during a presentation. Thus if you write `\uncover<2>{Text.}`, the text is not shown on any but the second slide. On the other slide, the text is not simply printed using the background color – it is not shown at all. This effect is most useful if your background does not have a uniform color.

Sometimes however, you might prefer that covered items are not completely covered. Rather, you would like them to be shown already in a very dim or shaded way. This allows your audience to get a feeling for what is yet to come, without getting distracted by it. Also, you might wish text that is covered "once more" still to be visible to some degree.

Ideally, there would be an option to make covered text "transparent." This would mean that when covered text is shown, it would instead be mixed with the background behind it. Unfortunately, `pgf` does not support real transparency yet. Nevertheless, one can come "quite close" to transparent text using the special command

```
\beamersetuncovermixins{#1}{#2}
```

This commands allows you to specify in a quite general way how a covered item should be rendered. You can even specify different ways of rendering the item depending on how long it will take before this item is shown or for how long it has already been covered once more. The transparency effect will automatically apply to all colors, *except* for the colors in images and shadings. For images and shadings there is a workaround, see the documentation of the PGF package.

As a convenience, two commands install a predefined uncovering behavior.

\beamertemplatetransparentcovered

Makes all covered text nearly transparent.

\beamertemplatetransparentcovereddynamic

Makes all covered text nearly transparent, but is a dynamic way. The longer it will take till the text is uncovered, the stronger the transparency.

\beamersetuncovermixins{⟨*not yet list*⟩}{⟨*once more list*⟩}

The ⟨*not yet list*⟩ specifies how to render covered items that have not yet been uncovered. The ⟨*once more list*⟩ specifies how to render covered items that have once more been covered. If you leave one of the specifications empty, the corresponding covered items are completely covered, that is, they are invisible.

Example:

```
\beamersetuncovermixins
  {\mixinon<1>{15!averagebackgroundcolor}
    \mixinon<2>{10!averagebackgroundcolor}
    \mixinon<3>{5!averagebackgroundcolor}
    \mixinon<4->{2!averagebackgroundcolor}}
  {\mixinon<1->{15!averagebackgroundcolor}}
```

The ⟨*not yet list*⟩ and the ⟨*once more list*⟩ can contain any number of the following two commands:

\mixinon<⟨*overlay specification*⟩>{⟨*mix-in specification*⟩}

The ⟨*overlay specification*⟩ specifies on which slides the ⟨*mix-in specification*⟩ should be applied to all colors. Unlike other overlay specifications, this ⟨*overlay specification*⟩ is a "relative" overlay specification. For example, the specification "3" here means "things that will be uncovered three slides ahead,"

respectively "things that have once more been covered for three slides." More precisely, if an item is uncovered for more than one slide and then covered once more, only the "first moment of uncovering" is used for the calculation of how long the item has been covered once more.

Mix-in specifications are a concept introduced by the `xcolor` package. The ⟨*mix-in specification*⟩ specifies how colors should be altered by adding another color to them. The specification consists of two parts, separated by an exclamation mark. The first part is a number between 0 and 100, where 0 means "do not mix in the text color at all" and 100 means "use only the text color". The second part is the color that should be mixed in. This second part may be omitted (along with the exclamation mark), in which case "white" is used as mix-in color. Any color that has been defined using the `\definecolor` command is permissible as a mix-in color.

The mix-in specifications is added to the PGF alternate extension for shadings and images (see the PGF documentation). Nested uses of mix-in accumulate.

Example:

```
\beamersetuncovermixins{\mixinon<1>{15!blue}{\mixinon<1->{15!white}}
\pgfdeclareimage{book}{book}
\pgfdeclareimage{book.!15!averagebackgroundcolor}{filenameforbooknearlyblue}
\pgfdeclareimage{book.!15!white}{filenameforbooknearlywhite}
```

For all items that become uncovered on the next slide or that have just been covered on the previous slide (depending on whether this command is used as part of the first or second parameter of the command `\beamersetuncovermixins`), use only 15% of the actual color and 85% of the average background color.

`\opaqueness`<⟨*overlay specification*⟩>{⟨*percentage of opaqueness*⟩}

Text that is covered on the specified slides (once more, relative to the current slide), is rendered with the specified opaqueness, where 100 is fully opaque and 0 is fully transparent. Currently, since real transparency is not yet implemented, this command does nearly the same as `\mixin`<⟨*overlay specification*⟩>{⟨*percentage of opaqueness*⟩`!averagebackgroundcolor}`. However, there are two differences: first, at some future point this command might result in real transparency; second, the alternate `pgf` extension is different.

The alternate PGF extension used inside an opaque area is ⟨*percentage of opaqueness*⟩`opaque`. In case of nested calls, only the innermost opaqueness specification is used.

Example:

```
\beamersetuncovermixins{\opaqueness<1->{15}{\opaqueness<1->{15}}
\pgfdeclareimage{book}{book}
\pgfdeclareimage{book.15opaque}{filenameforbooknearlytransparent}
```

Makes everything that is uncovered in two slides only 15 percent opaque.

`\invisibleon`<⟨*overlay specification*⟩>

Text that is covered on the specified slides (once more, relative to the current slide), is not shown at all.

Example: `\invisibleon<2->`

6.3 Animations

A word of warning first: Animations can be very distracting. No matter how cute a rotating, flying theorem seems to look and no matter how badly you feel your audience needs some action to keep it happy, most people in the audience will typically feel you are making fun of them.

6.3.1 Using an External Viewer

If you have created an animation using some external program (like a renderer), you can use the capabilities of the presentation program (like the Acrobat Reader) to show the animation. Unfortunately, currently there is no portable way of doing this and even the Acrobat Reader does not support this feature on all platforms.

6.3.2 Animations Created by Showing Slides in Rapid Succession

You can create an animation in a portable way by using the overlay commands of the BEAMER package to create a series of slides that, when shown in rapid succession, present an animation. This is a flexible approach, but such animations will typically be rather static since it will take some time to advance from one slide to the next. This approach is mostly useful for animations where you want to explain each "picture" of the animation. When you advance slides "by hand," that is, by pressing a forward button, it typically takes at least a second for the next slide to show.

More "lively" animations can be created by relying on a capability of the viewer program. Some programs support showing slides only for a certain number of seconds during a presentation (for the Acrobat Reader this works only in full-screen mode). By setting the number of seconds to zero, you can create a rapid succession of slides.

To facilitate the creation of animations using this feature, commands can be used: `\animate` and `\animatevalue`.

`\animate<`⟨*overlay specification*⟩`>`

The slides specified by ⟨*overlay specification*⟩ will be shown only as shortly as possible.

Example:

```
\frame{
  \frametitle{A Five Slide Animation}
  \animate<2-4>

  The first slide is shown normally. When the second slide is shown
  (presumably after pressing a forward key), the second, third, and
  fourth slides ``flash by.'' At the end, the content of the fifth
  slide is shown.

  ... code for creating an animation with five slides ...
}
```

`\animatevalue<`⟨*start slide*⟩`-`⟨*end slide*⟩`>{`⟨*name*⟩`}{`⟨*start value*⟩`}{`⟨*end value*⟩`}`

The ⟨*name*⟩ must be the name of a counter or a dimension. It will be varied between two values. For the slides in the specified range, the counter or dimension is set to an interpolated value that depends on the current slide number. On slides before the ⟨*start slide*⟩, the counter or dimension is set to ⟨*start value*⟩; on the slides after the ⟨*end slide*⟩ it is set to ⟨*end value*⟩.

Example:

```
\newcount\opaqueness
\frame{
  \animate<2-10>
  \animatevalue<1-10>{\opaqueness}{100}{0}
  \begin{colormixin}{\the\opaqueness!averagebackgroundcolor}
    \frametitle{Fadeout Frame}

    This text (and all other frame content) will fade out when the
    second slide is shown. This even works with
    {\color{green!90!black}colored} \alert{text}.
  \end{colormixin}
}

\newcount\opaqueness
\newdimen\offset
\frame{
  \frametitle{Flying Theorems (You Really Shouldn't!)}

  \animate<2-14>

  \animatevalue<1-15>{\opaqueness}{100}{0}
  \animatevalue<1-15>{\offset}{0cm}{-5cm}
  \begin{colormixin}{\the\opaqueness!averagebackgroundcolor}
  \hskip\offset
```

```
      \begin{minipage}{\textwidth}
        \begin{theorem}
          This theorem flies out.
        \end{theorem}
      \end{minipage}
    \end{colormixin}

    \animatevalue<1-15>{\opaqueness}{0}{100}
    \animatevalue<1-15>{\offset}{-5cm}{0cm}
    \begin{colormixin}{\the\opaqueness!averagebackgroundcolor}
    \hskip\offset
      \begin{minipage}{\textwidth}
        \begin{theorem}
          This theorem flies in.
        \end{theorem}
      \end{minipage}
    \end{colormixin}
  }
```

6.4 Slide Transitions

PDF in general, and the Acrobat Reader in particular, offer a standardized way of defining *slide transitions*. Such a transition is a visual effect that is used to show the slide. For example, instead of just showing the slide immediately, whatever was shown before might slowly "dissolve" and be replaced by the slide's content.

Slide transitions should be used with great care. Most of the time, they only distract. However, they can be useful in some situations: For example, you might show a young boy on a slide and might wish to dissolve this slide into slide showing a grown man instead. In this case, the dissolving gives the audience visual feedback that the young boy "slowly becomes" the man.

There are a number of commands that can be used to specify what effect should be used when the current slide is presented. Consider the following example:

```
\frame{
  \pgfuseimage{youngboy}
}
\frame{
  \transdissolve
  \pgfuseimage{man}
}
```

The command `\transdissolve` causes the slide of the second frame to be shown in a "dissolved way." Note that the dissolving is a property of the second frame, not of the first one. We could have placed the command anywhere on the frame.

The transition commands are overlay-specification-aware. We could collapse the two frames into one frame like this:

```
\frame{
  \only<1>{\pgfuseimage{youngboy}}
  \only<2>{\pgfuseimage{man}}
  \transdissolve<2>
}
```

This states that on the first slide the young boy should be shown, on the second slide the old man should be shown, and when the second slide is shown, it should be shown in a "dissolved way."

In the following, the different commands for creating transitional effects are listed. All of them take an optional argument that may contain a list of ⟨*key*⟩=⟨*value*⟩ pairs. The following options are possible:

- `duration=`⟨*seconds*⟩. Specifies the number of ⟨*seconds*⟩ the transition effect needs. Default is one second, but often a shorter one (like 0.2 seconds) is more appropriate. Viewer applications, especially Acrobat, may interprete this option in slightly strange ways.
- `direction=`⟨*degree*⟩. For "directed" effects, this option specifies the effect's direction. Allowed values are `0`, `90`, `180`, `270`, and for the glitter effect also `315`.

`\transblindshorizontal<`*⟨overlay specification⟩*`>[`*⟨options⟩*`]`

Show the slide as if horizontal blinds where pulled away.

Example: `\transblindshorizontal`

`\transblindsvertical<`*⟨overlay specification⟩*`>[`*⟨options⟩*`]`

Show the slide as if vertical blinds where pulled away.

Example: `\transblindsvertical<2,3>`

`\transboxin<`*⟨overlay specification⟩*`>[`*⟨options⟩*`]`

Show the slide by moving to the center from all four sides.

Example: `\transboxin<1>`

`\transboxout<`*⟨overlay specification⟩*`>[`*⟨options⟩*`]`

Show the slide by showing more and more of a rectangular area that is centered on the slide center.

Example: `\transboxout`

`\transdissolve<`*⟨overlay specification⟩*`>[`*⟨options⟩*`]`

Show the slide by slowly dissolving what was shown before.

Example: `\transdissolve[duration=0.2]`

`\transglitter<`*⟨overlay specification⟩*`>[`*⟨options⟩*`]`

Show the slide with a glitter effect that sweeps in the specified direction.

Example: `\transglitter<2-3>[direction=90]`

`\transsplitverticalin<`*⟨overlay specification⟩*`>[`*⟨options⟩*`]`

Show the slide by sweeping two vertical lines from the sides inward.

Example: `\transsplitverticalin`

`\transsplitverticalout<`*⟨overlay specification⟩*`>[`*⟨options⟩*`]`

Show the slide by sweeping two vertical lines from the center outward.

Example: `\transsplitverticalout`

`\transsplithorizontalin<`*⟨overlay specification⟩*`>[`*⟨options⟩*`]`

Show the slide by sweeping two horizontal lines from the sides inward.

Example: `\transsplithorizontalin`

`\transsplithorizontalout<`*⟨overlay specification⟩*`>[`*⟨options⟩*`]`

Show the slide by sweeping two horizontal lines from the center outward.

Example: `\transsplithorizontalout`

`\transwipe<`*⟨overlay specification⟩*`>[`*⟨options⟩*`]`

Show the slide by sweeping a single line in the specified direction, thereby "wiping out" the previous contents.

Example: `\transwipe[direction=90]`

You can also specify how *long* a given slide should be shown, using the following overlay-specification-aware command:

`\transduration<`*⟨overlay specification⟩*`>{`*⟨number of seconds⟩*`}`

In full screen mode, show the slide for *⟨number of seconds⟩*. In zero is specified, the slide is shown as short as possible. This can be used to create interesting pseudo-animations.

Example: `\transduration<2>{1}`

7 Managing Non-Presentation Versions and Material

The BEAMER package offers different ways of creating special versions of your talk and adding material that are not shown during the presentation. You can create a *handout* version of the presentation that can be distributed to the audience. You can also create a version that is more suitable for a presentation using an overhead projector. You can add notes for yourself that help you remember what to say for specific slides. Finally, you can have a completely independent "article" version of your presentation coexist in your main file. All special versions are created by specifying different class options and rerunning TEX on the main file.

7.1 Creating Handouts

A *handout* is a version of a presentation that is printed on paper and handed out to the audience before or after the talk. (See Section 3.4.2 for how to place numerous frames on one pages, which is very useful for handouts.) For the handout you typically want to produce as few slides as possible per frame. In particular, you do not want to print a new slide for each slide of a frame. Rather, only the "last" slide should be printed.

In order to create a handout, specify the class option `handout`. If you do not specify anything else, this will cause all overlay specifications to be suppressed. For most cases this will create exactly the desired result.

`\documentclass[handout]{beamer}`

Create a version that uses the `handout` overlay specifications.

In some cases, you may want a more complex behaviour. For example, if you use many `\only` commands to draw an animation. In this case, suppressing all overlay specifications is not such a good idea, since this will cause all steps of the animation to be shown at the same time. In some cases this is not desirable. Also, it might be desirable to suppress some `\alert` commands that apply only to specific slides in the handout.

For a fine-grained control of what is shown on a handout, you can use *alternate overlay specifications*. They specify which slides of a frame should be shown for a special version, for example for the handout version. An alternate overlay specification is written alongside the normal overlay specification inside the pointed brackets. It is separated from the normal specification by a vertical bar and a space. The version to which the alternate specification applies is written first, followed by a colon. Here is an example:

```
\only<1-3,5-9| handout:2-3,5>{Text}
```

This specification says: "Normally, insert the text on slides 1–3 and 5–9. For the handout version, insert the text only on slides 2, 3, and 5." If no alternate overlay specification is given for handouts, the default is "always." This causes the desirable effect that if you do not specify anything, the overlay specification is effectively suppressed for the handout.

An especially useful specification is the following:

```
\only<3| handout:0>{Not shown on handout.}
```

Since there is no zeroth slide, the text is not shown. Likewise, `\alert<3| handout:0>{Text}` will not alert the text on a handout.

You can also use an alternate overlay specification for the optional argument of the frame command as in the following example.

```
\frame<1-| handout:0>{Text...}
```

This causes the frame to be suppressed in the handout version. Also, you can restrict the presentation such that only specific slides of the frame are shown on the handout:

```
\frame<1-| handout:4-5>{Text...}
```

It is also possible to give only an alternate overlay specification. For example, `\alert<handout:0>{...}` causes the text to be always hilighted during the presentation, but never on the handout version. Likewise, `\frame<handout:0>{...}` causes the frame to be suppressed for the handout.

Finally, note that it is possible to give more than one alternate overlay specification and in any order. For example, the following specification states that the text should be inserted on the first three slides in the presentation, in the first two slides of the transparency version, and not at all in the handout.

```
\only<trans:1-2| 1-3| handout:0>{Text}
```

If you wish to give the same specification in all versions, you can do so by specifying `all:` as the version. For example,

```
\frame<all:1-2>
{
  blah...
}
```

ensures that the frame has two slides in all versions.

7.2 Creating Transparencies

The main aim of the BEAMER class is to create presentations for beamers. However, it is often useful to print transparencies as backup, in case the hardware fails. A transparencies version of a talk often has less slides than the main version, since it takes more time to switch slides, but it may have more slides than the handout version. For example, while in a handout an animation might be condensed to a single slide, you might wish to print several slides for the transparency version.

You can use the same mechanism as for creating handouts: Specify `trans` as a class option and add alternate transparency specifications for the `trans` version as needed. An elaborated example of different overlay specifications for the presentation, the handout, and the transparencies can be found in the file `beamerexample1.tex`.

`\documentclass[trans]{beamer}`

Create a version that uses the `trans` overlay specifications.

7.3 Adding Notes

You can add notes to your slides using the command `\note`. A note is a reminder to yourself of what you should say or should keep in mind when presenting a frame. The `\note` command should be given after the frame to which the note applies. Here is a typical example.

```
\frame{
  \begin{itemize}
  \item<1-> Eggs
  \item<2-> Plants
  \item<3-> Animals
  \end{itemize}
}
\note{Tell joke about eggs.}
```

The note command will create a new page that contains your text plus some information that should make it easier to match the note to the frame while talking.

Since you normally do not wish the notes to be part of your presentation, you must explicitly specify the class option `notes` to include notes. If this option is not specified, notes are suppressed. If you specify `notesonly` instead of `notes`, only notes will be included and all normal frames are parsed, but not displayed. This is useful for printing the notes.

By default, you can fit only little on each note (they are only intended to be reminders after all). Using the class option `compressnotes` will allow you to squeeze much more on each note card.

`\documentclass[notes]{beamer}`

Include notes in the output file. Normally, notes are not included.

`\documentclass[notesonly]{beamer}`

Include only the notes in the output file. Useful for printing them.

`\documentclass[compressnotes]{beamer}`

Squeezes as much text as possible on each note card.

`\note{`⟨*note text*⟩`}`

Creates a note page. Should be given right after a frame.

Example: `\note{Talk no more than 1 minute.}`

\noteitems{⟨*list of item commands*⟩}

Just like the \note command, except that an itemize environment is setup inside the note.

Example:

```
\frame{Bla bla...}
\noteitems{
\item Stress the importance.
\item Use no more than 2 minutes.
}
```

7.4 Creating an Article Version

In the following, the "article version" of your presentation refers to a normal TEX text typeset using, for example, the document class article or perhaps lncs or a similar document class. This version of the presentation will typically follow different typesetting rules and may even have a different structure. Nevertheless, you may wish to have this version coexist with your presentation in one file and you may wish to share some part of it (like a figure or a formula) with your presentation.

7.4.1 Article, Common, and Presentation Mode

The class option class=⟨*class name*⟩, where ⟨*class name*⟩ is the name of another document class like article or report, causes the beamer class to transfer control almost immediately to the class named ⟨*class name*⟩. None of the normal commands defined by the beamer class will be defined, except for the three commands listed in the following. All class options passed to the beamer class will be passed on to the class ⟨*class name*⟩, *except*, naturally, for the option class=⟨*class name*⟩ itself.

\documentclass[class=⟨*another class name*⟩,⟨*options for another class*⟩]{beamer}

Transfer control to document class ⟨*another class name*⟩ with the options ⟨*options for another class*⟩. See Section 7.4 for details.

Example:

```
\documentclass[class=article,a4paper]{beamer}
```

This will cause the rest of the text to be typeset using the article class with the only class option being a4paper.

You can use three commands to specify which part of your text belong to the article version, which belongs to the actual presentation, and which belongs to both. These command switch between three different modes: article mode, presentation mode, and common mode. While TEX scans text in the article mode, this text is read normally when an article is requested, but thrown away if a presentation is requested. In presentation mode, the behavior is the other way round. In common mode, the text is always inserted.

Right after the documentclass command and right after the \begin{document} (provided it is the sole entry on a line with no comments following and no leading spaces), TEX is always in common mode. If you do not wish this to be the case, simply append a comment to the line.

If you use \input or \include or a related command to include another file, make sure that when TEX reaches the end of this file, it is in common mode.

\article

All text following this command will only be present in the article version. For the presentation version, this text will be completely ignored. This command must be the only command in a line and it must start the line.

\presentation

All text following this command will only be present in the presentation version. For the article version, the text will be completely ignored. This command must be the only command in a line and it must start the line.

\common

All text following this command will be present in both the article and the presentation version. This command must be the only command in a line and it must start the line.

```
\documentclass[class=article,a4paper]{beamer}
\documentclass[red]{beamer}

\article
\usepackage{fullpage}

\common
\usepackage[english]{babel}
\usepackage{pgf}

\presentation
\usepackage{beamerthemesplit}

\begin{document}

\pgfdeclareimage[height=1cm]{myimage}{filename}

\presentation
  \frame{

\common
    \begin{figure}
      \pgfuseimage{myimage}
    \end{figure}

\presentation
    }
\end{document}
```

The above commands cannot be used inside macros (they are implemented similarly to `verbatim` environments, only that the contents is sometimes thrown away instead of rendered). However, there is one exception: Inside a `\frame`, these commands can be used, provided they "balance" inside the frame and provided you switch back to presentation mode by the end of the frame (as in the above example). If you have problems with these commands inside a frame, try using a `\def` command outside the frame as in the following example:

```
\begin{document}
...
\common
  \def\myfigure{
    \begin{figure}
      \pgfimage{filename}
    \end{figure}}

\article
  \myfigure

\presentation
  \frame{\myfigure}
\end{document}
```

7.4.2 Workflow

The following workflow steps are optional, but they can simplify the creation of the article version.

- In the main file `main.tex`, delete the first line, which sets the document class.
- Create a file named, say, `main.beamer.tex` with the following content:

  ```
  \documentclass{beamer}
  \input{main.tex}
  ```

- Create an extra file named, say, `main.article.tex` with the following content:

```
\documentclass[class=article]{beamer}
\setjobnamebeamerversion{main.beamer}
\input{main.tex}
```

- You can now run `pdflatex` or `latex` on the two files `main.beamer.tex` and `main.article.tex`.

The command `\setjobnamebeamerversion` tells the article version where to find the presentation version. This is necessary if you wish to include slides from the presentattion version in an article as figures.

`\setjobnamebeamerversion{`⟨*filename without extension*⟩`}`

Tells the beamer class where to find the presentation version of the current file.

An example of this workflow approach can be found in the `examples` subdirectory for files starting with `beamerexample2`.

7.4.3 Including Slides from the Presentation Version in the Article Version

In order to include a slide from your presentation in your article version, you must do two things: First, you must place a normal LaTeX label on the slide using the `\label` command. Since this command is overlay-specification-aware, you can also select specific slides of a frame. Also, by adding the option `label=`⟨*name*⟩ to a frame, a label ⟨*name*⟩`<`⟨*slide number*⟩`>` is automatically added to each slide of the frame.

Once you have labeled a slide, you can use the following command in your article version to insert the slide into it:

`\includeslide[`⟨*options*⟩`]{`⟨*label name*⟩`}`

This command calls `\pgfimage` with the given ⟨*options*⟩ for the file specified by

`\setjobnamebeamerversion`⟨*filename*⟩

Furthermore, the option `page=`⟨*page of label name*⟩ is passed to `\pgfimage`, where the ⟨*page of label name*⟩ is read internally from the file ⟨*filename*⟩`.snm`.

Example:

```
\article
  \begin{figure}
    \begin{center}
      \includeslide[height=5cm]{slide1}
    \end{center}
    \caption{The first slide (height 5cm). Note the partly covered second item.}
  \end{figure}
  \begin{figure}
    \begin{center}
      \includeslide{slide2}
    \end{center}
    \caption{The second slide (original size). Now the second item is also shown.}
  \end{figure}
```

The exact effect of passing the option `page=`⟨*page of label name*⟩ to the command `\pgfimage` is explained in the documentation of `pgf`. In essence, the following happens:

- For old version of `pdflatex` and for any version of `latex` together with `dvips`, the `pgf` package will look for a file named

  ⟨*filename*⟩`.page`⟨*page of label name*⟩`.`⟨*extension*⟩

  For each page of your `.pdf` or `.ps` file that is to be included in this way, you must create such a file by hand. For example, if the PostScript file of your presentation version is named `main.beamer.ps` and you wish to include the slides with page numbers 2 and 3, you must create (single page) files `main.beamer.page2.ps` and `main.beamer.page3.ps` "by hand" (or using some script). If these files cannot be found, `pgf` will complain.

- For new versions of `pdflatex`, `pdflatex` also looks for the files according to the above naming scheme. However, if it fails to find them (because you have not produced them), it uses a special mechanism to directly extract the desired page from the presentation file `main.beamer.pdf`.

8 Customization

8.1 Fonts

8.1.1 Serif Fonts and Sans-Serif Fonts

By default, the beamer class uses the Computer Modern sans-serif fonts for typesetting a presentation. The Computer Modern font family is the original font family designed by Donald Knuth himself for the TEX program. A sans-serif font is a font in which the letters do not have serifs (from Frensh *sans*, which means "without"). Serifs are the little hooks at the ending of the strokes that make up a letter. The font you are currently reading is a serif font. By comparison, this text is in a sans-serif font.

The choice Computer Modern sans-serif had the following reasons:

- The Computer Modern family has a very large number of symbols available that go well together.
- Sans-serif fonts are (generally considered to be) easier to read when used in a presentation. In low resolution rendering, serifs decrease the legibility of a font.

While these reasons are pretty good, you still might wish to change the font:

- The Computer Modern fonts are a bit boring if you have seen them too often. Using another font (but not Times!) can give a fresh look.
- Other fonts, especially Times, are sometime rendered better since they seem to have better internal hinting.
- A presentation typeset in a serif font creates a conservative impression, which might be exactly what you wish to create.

You must decide whether the text should be typeset in sans-serif or in serif. To choose this, use either the class option `sans` or `serif`. By default, `sans` is selected, so you do not need to specify this.

`\documentclass[sans]{beamer}`

Use a sans-serif font during the presentation. (Default.)

`\documentclass[serif]{beamer}`

Use a serif font during the presentation.

8.1.2 Fonts in Mathematical Text

By default, if a sans-serif font is used for the main text, mathematical formulas are also typeset using sans-serif letters. In most cases, this is visually the pleasing and easily readable way of typesetting mathematical formulas. However, in mathematical texts the font used to render, say, a variable is sometimes used to differentiate between different meanings of this variable. In such case, it may be necessary to typeset mathematical text using serif letters. Also, if you have a lot of mathematical text, the audience may be quicker to "parse" it, if it typeset in the way people usually read mathematical text: in a serif font.

You can use the two options `mathsans` and `mathserif` to override the overall sans-serif/serif choice for math text. However, using the option `mathsans` in a `serif` environment makes little sense in my opinion.

`\documentclass[mathsans]{beamer}`

Override the math font to be a sans-serif font.

`\documentclass[mathserif]{beamer}`

Override the math font to be a serif font.

The command `\mathrm` will always produce upright (not slanted), serif text and the command `\mathsf` will always produce upright, sans-serif text. The command `\mathbf` will produce upright, bold-face, sans-serif or serif text, depending on whether `mathsans` or `mathserif` is used.

To produce an upright, sans-serif or serif text, depending on whether `mathsans` or `mathserif` is used, you can use for instance the command `\operatorname` from the `amsmath` package. Using this command instead of `\mathrm` or `\mathsf` directly will automatically adjust upright mathematical text if you switch from sans-serif to serif or back.

8.1.3 Font Families

Independently of the serif/sans-serif choice, you can switch the document font. To do so, you should use one of the prepared packages of LaTeX's font mechanism. For example, to change to Times/Helvetica, simply add

```
\usepackage{times}
```

in your preamble. Note that if you do not specify `serif` as a class option, Helvetica (not Times) will be selected as the text font.

There may be many other fonts available on your installation. Typically, at least some of the following packages should be available: `avant`, `bookman`, `chancery`, `charter`, `euler`, `helvet`, `mathtime`, `mathptm`, `newcent`, `palatino`, `pifont`, `times`, `utopia`.

If you use `times` together with the `serif` option, you may wish to include also the package `mathptm`. If you use the `mathtime` package (you have to buy some of the fonts), you also need to specify the `serif` option.

8.1.4 Font Sizes

The default sizes of the fonts are chosen in a way that makes it difficult to fit "too much" onto a slide. Also, it will ensure that your slides are readable even under bad conditions like a large room and a small only a small projection area. However, you may wish to enlarge or shrink the fonts a bit if you know this to be more appropriate in your presentation environment.

The default font size is 11pt. This may seem surprisingly small, but the actual size of each frame is just 128mm times 96mm and the viewer application enlarges the font. By specifying a default font size smaller than 11pt you can put more onto each slide, by specifying a larger font size you can fit on less.

To specify the the font size, you can use the following class options:

`\documentclass[8pt]{beamer}`

This is way too small. Requires that the package `extsize` is installed.

`\documentclass[9pt]{beamer}`

This is also too small. Requires that the package `extsize` is installed.

`\documentclass[10pt]{beamer}`

If you really need to fit much onto each frame, use this option. Works without `extsize`.

`\documentclass[smaller]{beamer}`

Same as the `10pt` option.

`\documentclass[11pt]{beamer}`

The default font size. You need not specify this option.

`\documentclass[12pt]{beamer}`

Makes all fonts a little bigger, which makes the text more readable. The downside is that less fits onto each frame.

`\documentclass[bigger]{beamer}`

Same as the `12pt` option.

`\documentclass[14pt]{beamer}`

Makes all fonts somewhat bigger. Requires `extsize` to be installed.

`\documentclass[17pt]{beamer}`

This is about the default size of PowerPoint. Requires `extsize` to be installed.

`\documentclass[20pt]{beamer}`

This is really huge. Requires `extsize` to be installed.

8.1.5 Font Encodings

The same font can come in different encodings, which are (very roughly spoken) the ways the characters of a text are mapped to glyphs (the actual shape of a particular character in a particular font at a particular size). In TEX two encodings are often used: the T1 encoding and the OT1 endocding (old T1 encoding).

Conceptually, the newer T1 encoding is preferable over the old OT1 encoding. For example, hyphenation of words containing umlauts (like the famous German work Fräulein) will work only if you use the T1 encoding. Unfotunately, only the bitmapped version of the Computer Modern fonts are available in this encoding in a standard installation. For this reason, using the T1 encoding will produce PDF files that render very poorly.

Most standard PostScript fonts are available in T1 encoding. For example, you can use Times in the T1 encoding. The package `lmodern` makes the standard Computer Modern fonts available in the T1 encoding. Furthermore, if you use `lmodern` several extra fonts become available (like a sans-serif boldface math) and extra symbols (like proper guillemots).

8.2 Margin Sizes

The "paper size" of a beamer presentation is fixed to 128mm times 96mm. The aspect ratio of this size is 4:3, which is exactly what most beamers offer these days. It is the job of the presentation program (like `acroread`) to display the slides at full screen size. The main advantage of using a small "paper size" is that you can use all your normal fonts at their natural sizes. In particular, inserting a graphic with 11pt labels will result in reasonably sized labels during the presentation.

You should refrain from changing the "paper size." However, you *can* change the size of the left and right margins, which default to 1cm. To change them, you should use the following two commands:

`\beamersetleftmargin{`⟨*left margin dimension*⟩`}`

Sets a new left margin. This excludes the left side bar. Thus, it is the distance between the right edge of the left side bar and the left edge of the text. This command can only be used in the preamble (before the `document` environment is used).

Example: `\beamersetleftmargin{1cm}`

`\beamersetrightmargin{`⟨*left margin dimension*⟩`}`

Like `\beamersetleftmargin`, only for the right margin.

For more information on side bars, see Section 8.4.9.

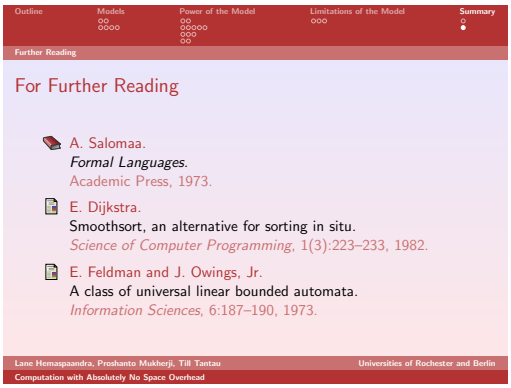
8.3 Themes

Just like LATEX in general, the BEAMER class tries to separate the contents of a text from the way it is typeset (displayed). There are two ways in which you can change how a presentation is typeset: you can specify a different theme and you can specify different templates. A theme is a predefined collection of templates.

There exist a number of different predefined themes that can be used together with the BEAMER class. Feel free to add further themes. Themes are used by including an appropriate LATEX style file, using the standard `\usepackage` command.

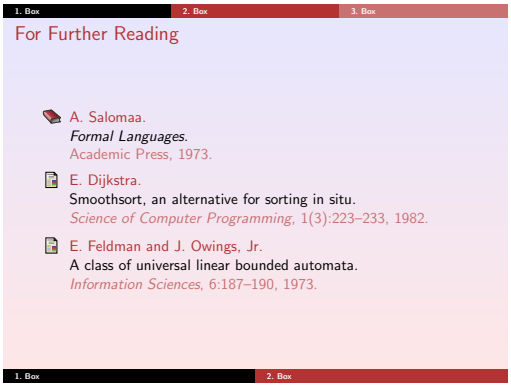
`\usepackage{beamerthemebars}`

Example:



`\usepackage[headheight=⟨head height⟩,footheight=⟨foot height⟩]{beamerthemeboxes}`

Example:



Example:

```
\usepackage[headheight=12pt,footheight=12pt]{beamerthemeboxes}
```

For this theme, you can specify an arbitrary number of templates for the boxes in the head line and in the foot line. You can add a template for another box by using the following commands.

`\addheadboxtemplate{⟨background color command⟩}{⟨box template⟩}`

Each time this command is invoked, a new box is added to the head line, with the first added box being shown on the left. All boxes will have the same size.

Example:

```
\addheadboxtemplate{\color{black}}{\color{white}\tiny\quad 1. Box}
\addheadboxtemplate{\color{structure}}{\color{white}\tiny\quad 2. Box}
\addheadboxtemplate{\color{structure!50}}{\color{white}\tiny\quad 3. Box}
```

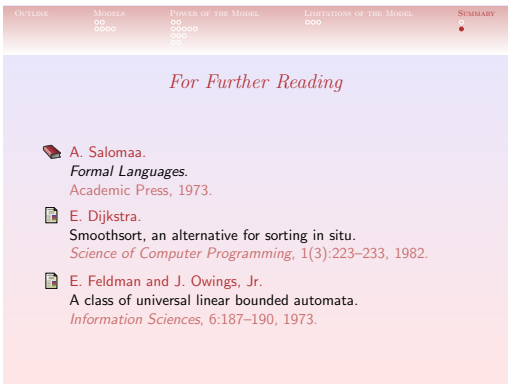
`\addfootboxtemplate{⟨background color command⟩}{⟨box template⟩}`

Example:

```
\addfootboxtemplate{\color{black}}{\color{white}\tiny\quad 1. Box}
\addfootboxtemplate{\color{structure}}{\color{white}\tiny\quad 2. Box}
```

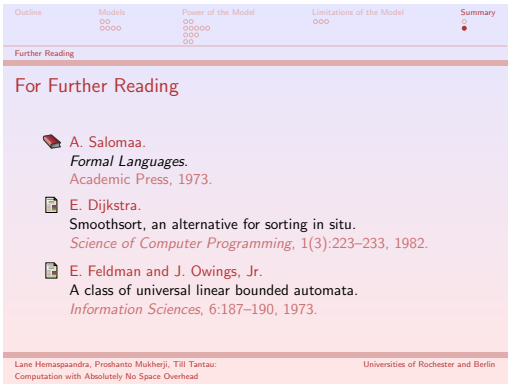
`\usepackage{beamerthemeclassic}`

Example:



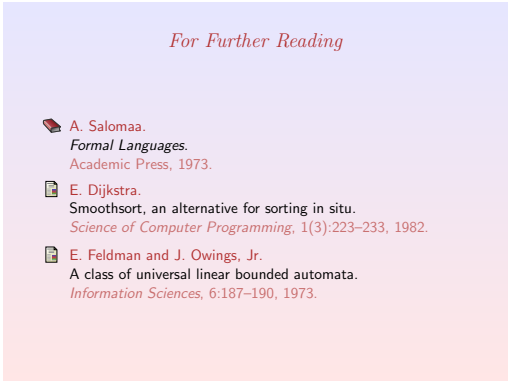
`\usepackage{beamerthemelined}`

Example:



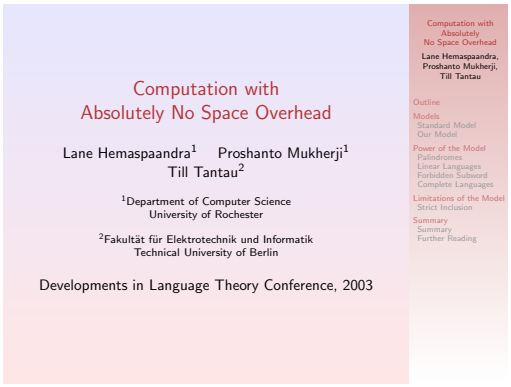
`\usepackage{beamerthemeplain}`

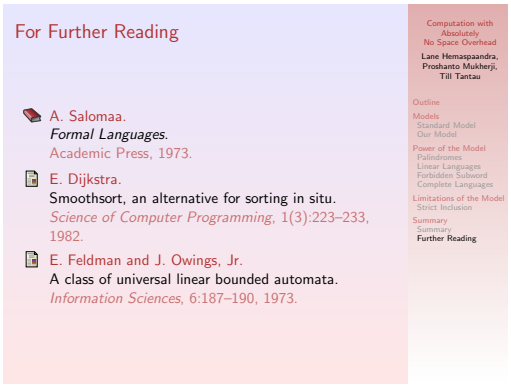
Example:



`\usepackage[width=`⟨*sidebar width*⟩`]{beamerthemesidebar}`

Example:





Example:

`\usepackage[width=3cm]{beamerthemesidebar}`

`\usepackage[width=`⟨*sidebar width*⟩`]{beamerthemesidebartab}`

Example:

`\usepackage[width=`⟨*sidebar width*⟩`]{beamerthemesidebardark}`

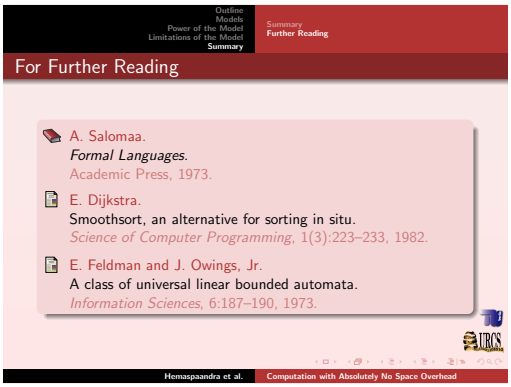
Example:

`\usepackage[width=`⟨*sidebar width*⟩`]{beamerthemesidebartabdark}`

Example:

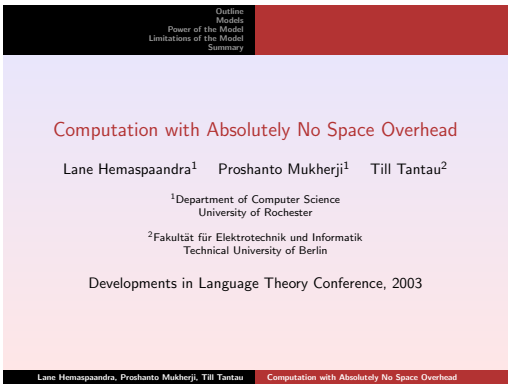
`\usepackage{beamerthemeshadow}`

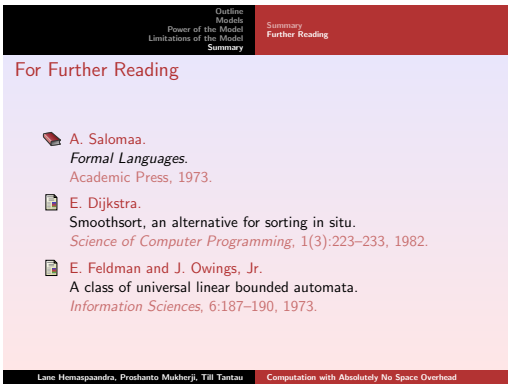
Example:



`\usepackage{beamerthemesplit}`

Example:

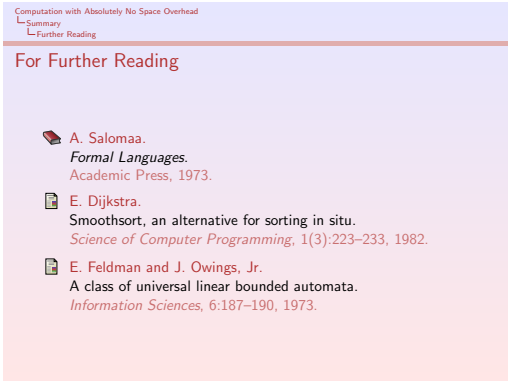




The theme `beamerthemesplitcondensed` is no longer supported. Use `beamerthemesplit` with the `compress` class option instead.

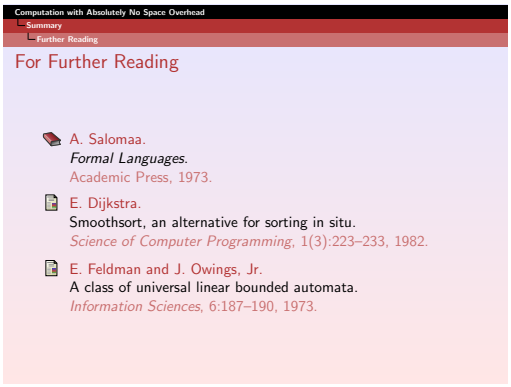
`\usepackage{beamerthemetree}`

Example:



```
\usepackage{beamerthemetreebars}
```

Example:



8.4 Templates

8.4.1 Introduction to Templates

If you only wish to modify a small part of how your presentation is rendered, you do not need to create a whole new theme. Instead, you can modify an appropriate template.

A template specifies how a part of a presentation is typeset. For example, the frame title template dictates where the frame title is put, which font is used, and so on.

As the name suggests, you specify a template by writing the exact LaTeX code you would also use when typesetting a single frame title by hand. Only, instead of the actual title, you use the command `\insertframetitle`.

For example, suppose we would like to have the frame title typeset in red, centered, and boldface. If we were to typeset a single frame title by hand, it might be done like this:

```
\frame
{
  \begin{centering}
    \color{red}
    \textbf{The Title of This Frame.}
    \par
  \end{centering}

  Blah, blah.
}
```

In order to typeset the frame title in this way on all slides, we can change the frame title template as follows:

```
\useframetitletemplate{
  \begin{centering}
    \color{red}
    \textbf{\insertframetitle}
    \par
  \end{centering}
}
```

We can then use the following code to get the desired effect:

```
\frame
{
  \frametitle{The Title of This Frame.}

  Blah, blah.
}
```

When rendering the frame, the BEAMER class will use the code of the frame title template to typeset the frame title and it will replace every occurrence of `\insertframetitle` by the current frame title.

In the following subsections all commands for changing templates are listed, like the above-mentioned command `\useframetitletemplate`. Inside these commands, you should use the `\insertxxxx` commands, which are listed following the template changing commands. Although the `\insertxxxx` commands are listed alongside the templates for which they make the most sense, you can (usually) also use them in all other templates.

Some of the below subsections start with commands for using *predefined* templates. Calling one of them will change a template in a predefined way. Using them, you can use, for example, your favorite theme together with a predefined background.

Here are a few hints that might be helpful when you wish to redefine a template:

- Usually, you might wish to copy code from an existing template. The code often takes care of some things that you may not yet have thought about.
- When copying code from another template and when inserting this code in the preamble of your document (not in another style file), you may have to "switch on" the at-character (`@`). To do so, add the command `\makeatletter` before the `\usexxxtemplate` command and the command `\makeatother` afterward.
- Most templates having to do with the frame components (head lines, side bars, etc.) can only be changed in the preamble. Other templates can be changed during the document.
- The height of the head line and foot line templates is calculated automatically. This is done by typesetting the templates and then "having a look" at their heights. This recalculation of the heights takes place several times, but at least twice: once directly after a call to `\useheadtemplate`, respectively `\usefoottemplate`, and once before the `\begin{document}`. Because of this, your templates must be "typesetable" inside the preamble. In particular, any images you use must already be declared.
- The left and right margins of the head and foot line templates are the same as of the normal text. In order to start the head line and foot line at the page margin, you must insert a negative horizontal skip using `\hskip-\Gm@lmargin`. You may wish to add a `\hskip-\Gm@rmargin` at the end to avoid having TEX complain about overfull boxes.
- Getting the boxes right inside any template is often a bit of a hassle. You may wish to consult the TEX book for the glorious details on "Making Boxes." If your headline is simple, you might also try putting everything into a `pgfpicture` environment, which makes the placement easier.

8.4.2 Title Page

Predefined Templates

`\beamertemplatelargetitlepage`

Causes the title page to be typeset with a large font for the title.

`\beamertemplateboldtitlepage`

Causes the title page to be typeset with a bold font for the title.

Template Installation Commands

`\usetitlepagetemplate{`⟨*title page template*⟩`}`

Example:

```
\usetitlepagetemplate{
  \vbox{}
  \vfill
  \begin{centering}
    \Large\structure{\inserttitle}
    \vskip1em\par
    \normalsize\insertauthor\vskip1em\par
    {\scriptsize\insertinstitute\par}\par\vskip1em
    \insertdate\par\vskip1.5em
    \inserttitlegraphic
  \end{centering}
  \vfill
}
```

If you wish to suppress the head and foot line in the title page, use `\frame[plain]{\titlepage}`.

Inserts for this Template

`\insertauthor`

Inserts the author names into a template.

`\insertdate`

Inserts the date into a template.

`\insertinstitute`

Inserts the institute into a template.

`\inserttitle`

Inserts a version of the document title into a template that is useful for the title page.

`\inserttitlegraphic`

Inserts the title graphic into a template.

8.4.3 Part Page

Predefined Templates

`\beamertemplatelargepartpage`

Causes the part pages to be typeset with a large font for the part name.

`\beamertemplateboldpartpage`

Causes the part pages to be typeset with a bold font for the part name.

Template Installation Commands

`\usepartpagetemplate{`⟨*part page template*⟩`}`

Example:

```
\usepartpagetemplate{
  \begin{centering}
    \Large\structure{\partname~\insertromanpartnumber}
    \vskip1em\par
    \insertpart\par
  \end{centering}
  }
```

Inserts for this Template

`\insertpart`

Inserts the current part name.

`\insertpartnumber`

Inserts the current part number as an Arabic number into a template.

`\insertpartromannumber`

Inserts the current part number as a Roman number into a template.

8.4.4 Background

Predefined Templates

\beamertemplateshadingbackground{⟨*color expression page bottom*⟩}{⟨*color expression page top*⟩}

Installs a vertically shaded background such that the specified bottom color changes smoothly to the specified top color. **Use with care: Background shadings are often distracting!** However, a very light shading with warm colors can make a presentation more lively.

Example:

```
\beamertemplateshadingbackground{red!10}{blue!10}
 Bottom is light red, top is light blue
```

\beamertemplategridbackground

Installs a light grid as background.

Template Installation Commands

\usebackgroundtemplate{⟨*background template*⟩}

Installs a new background template. Call \beamersetaveragebackground after you have called this macro, see Section 6.2.2 for details.

Example:

```
\usebackgroundtemplate{%
  \color{red}%
  \vrule  height\paperheight width\paperwidth%
}
```

8.4.5 Table of Contents

Predefined Templates

\beamertemplateplaintoc

Installs a simple table of contents template with indented subsections.

Example: \beamertemplateplaintoc

\beamertemplateballsectiontoc

Installs a table of contents template in which small balls are shown before each section and subsection.

Example: \beamertemplatenumberedballsectiontoc

\beamertemplatenumberedsectiontoc

Installs a table of contents template in which the sections are numbered.

Example: \beamertemplatenumberedsectiontoc

\beamertemplatenumberedcirclesectiontoc

Installs a table of contents template in which the sections are numbered and the numbers are drawn on a small circle.

Example: \beamertemplatenumberedcirclesectiontoc

\beamertemplatenumberedballsectiontoc

Installs a table of contents template in which the sections are numbered and the numbers are drawn on a small ball.

Example: \beamertemplatenumberedballsectiontoc

\beamertemplatenumberedsubsectiontoc

Installs a table of contents template in which the subsections are numbered.

Example: \beamertemplatenumberedsubsectiontoc

Template Installation Commands

\usetemplatetocsection[⟨*mix-in specification*⟩]{⟨*template*⟩}{⟨*grayed template*⟩}

Installs a ⟨*template*⟩ for rendering sections in the table of contents. If the ⟨*mix-in specification*⟩ is present, the ⟨*grayed template*⟩ may not be present and the grayed sections names are obtained by mixing in the ⟨*mix-in specification*⟩. If ⟨*mix-in specification*⟩ is not present, ⟨*grayed template*⟩ must be present and is used to render grayed section names.

Example:

```
\usetemplatetocsection
{\color{structure}\inserttocsection}
{\color{structure!50}\inserttocsection}

\usetemplatetocsection[50!averagebackgroundcolor]
{\color{structure}\inserttocsection}
```

\usetemplatetocsubsection[⟨*mix-in specification*⟩]{⟨*template*⟩}{⟨*grayed template*⟩}

See \usetemplatetocsection.

Example:

```
\usetemplatetocsubsection
{\leavevmode\leftskip=1.5em\color{black}\inserttocsubsection\par}
{\leavevmode\leftskip=1.5em\color{black!50!white}\inserttocsubsection\par}

\usetemplatetocsection[50!averagebackgroundcolor]
{\leavevmode\leftskip=1.5em\color{black}\inserttocsubsection\par}
```

Inserts for this Template

\inserttocsection

Inserts the table of contents version of the current section name into a template.

\inserttocsectionnumber

Inserts the number of the current section (in the table of contents) into a template.

\inserttocsubsection

Inserts the table of contents version of the current subsection name into a template.

\inserttocsubsectionnumber

Inserts the number of the current subsection (in the table of contents) into a template.

8.4.6 Bibliography

Predefined Templates

\beamertemplatetextbibitems

Shows the citation text in front of references in a bibliography instead of a small symbol.

\beamertemplatearrowbibitems

Changes the symbol before references in a bibliography to a small arrow.

\beamertemplatebookbibitems

Changes the symbol before references in a bibliography to a small book icon.

\beamertemplatearticlebibitems

Changes the symbol before references in a bibliography to a small article icon. (Default)

Template Installation Commands

\usebibitemtemplate{⟨*citation template*⟩}

Installs a template for the citation text before the entry. (The "label" of the item.)

Example: `\usebibitemtemplate{\color{structure}\insertbiblabel}`

\usebibliographyblocktemplate{⟨*template 1*⟩}{⟨*template 2*⟩}{⟨*template 3*⟩}{⟨*template 4*⟩}

The text ⟨*template 1*⟩ is inserted before the first block of the entry (the first block is all text before the first occurrence of a `\newblock` command). The text ⟨*template 2*⟩ is inserted before the second block (the text between the first and second occurrence of `\newblock`). Likewise for ⟨*template 3*⟩ and ⟨*template 4*⟩.

The templates are inserted *before* the blocks and you do not have access to the blocks themselves via insert commands. In the following example, the first `\par` commands ensure that the author, the title, and the journal are put on different lines. The color commands cause the author (first block) to be typeset using the theme color, the second block (title of the paper) to be typeset in black, and all other lines to be typeset in a washed-out version of the theme color.

Example:

```
\usebibliographyblocktemplate
{\color{structure}}
{\par\color{black}}
{\par\color{structure!75}}
{\par\color{structure!75}}
```

Inserts for these Templates

\insertbiblabel

Inserts the current citation label into a template.

8.4.7 Frame Titles

Predefined Templates

\beamertemplateboldcenterframetitle

Typesets the frame title using a bold face and centers it.

\beamertemplatelargeframetitle

Typesets the frame title using a large face and flushes it left.

Template Installation Commands

\useframetitletemplate{⟨*frame title template*⟩}

Example:

```
\useframetitletemplate{%
  \begin{centering}
    \structure{\textbf{\insertframetitle}}
    \par
  \end{centering}
}
```

Inserts for this Template

\insertframetitle

Inserts the current frame title into a template.

8.4.8 Head Lines and Foot Lines

Predefined Templates

\beamertemplateheadempty

Makes the head line empty.

\beamertemplatefootempty

Makes the foot line empty.

\beamertemplatefootpagenumber

Shows only the page number in the foot line.

Template Installation Commands

\usefoottemplate{⟨*foot line template*⟩}

The final height of the foot line is calculated by invoking this template just before the beginning of the document and by setting the foot line height to the height of the template.

Example:

```
\usefoottemplate{\hfil\tiny{\color{black!50}\insertpagenumber}}
```

or

```
\usefoottemplate{%
  \vbox{%
    \tinycolouredline{structure!75}%
      {\color{white}\textbf{\insertshortauthor\hfill\insertshortinstitute}}%
    \tinycolouredline{structure}%
      {\color{white}\textbf{\insertshorttitle}\hfill}%
    }}
```

\useheadtemplate{⟨*head line template*⟩}

See \usefoottemplate.

Example:

```
\useheadtemplate{%
  \vbox{%
  \vskip3pt%
  \beamerline{\insertnavigation{\paperwidth}}%
  \vskip1.5pt%
  \insertvrule{0.4pt}{structure!50}}%
}
```

Inserts for these Templates

\insertframenumber

Inserts the number of the current frame (not slide) into a template.

\inserttotalframenumber

Inserts the total number of the frames (not slides) into a template. The number is only correct on the second run of TEX on your document.

\insertlogo

Inserts the logo(s) into a template.

\insertnavigation{⟨*width*⟩}

Inserts a horizontal navigation bar of the given ⟨*width*⟩ into a template. The bar lists the sections and below them mini frames for each frame in that section.

\insertpagenumber

Inserts the current page number into a template.

\insertsection

Inserts the current section into a template.

\insertsectionnavigation{⟨*width*⟩}

Inserts a vertical navigation bar containing all sections, with the current section hilighted.

\insertsectionnavigationhorizontal{⟨*width*⟩}{⟨*left insert*⟩}{⟨*right insert*⟩}

Inserts a horizontal navigation bar containing all sections, with the current section hilighted. The ⟨*left insert*⟩ will be inserted to the left of the sections, the {⟨*right insert*⟩} to the right. By inserting a triple fill (a `filll`) you can flush to bar to the left or right.

Example:

```
\insertsectionnavigationhorizontal{.5\textwidth}{\hskip0pt plus1filll}{}
```

\insertshortauthor

Inserts the short version of the author into a template.

\insertshortdate

Inserts the short version of the date into a template.

\insertshortinstitute

Inserts the short version of the institute into a template.

\insertshortpart

Inserts the short version of the part name into a template.

\insertshorttitle

Inserts the short version of the document title into a template.

\insertsubsection

Inserts the current subsection into a template.

\insertsubsectionnavigation{⟨*width*⟩}

Inserts a vertical navigation bar containing all subsections of the current section, with the current subsection hilighted.

\insertsubsectionnavigationhorizontal{⟨*width*⟩}{⟨*left insert*⟩}{⟨*right insert*⟩}

See `\insertsectionnavigationhorizontal`.

\insertverticalnavigation{⟨*width*⟩}

Inserts a vertical navigation bar of the given ⟨*width*⟩ into a template. The bar shows a little table of contents. The individual lines are typeset using the templates `\usesectionsidetemplate` and `\usesubsectionsidetemplate`.

\insertvrule{⟨*color expression*⟩}{⟨*thickness*⟩}

Inserts a rule of the given color and ⟨*thickness*⟩ into a template.

8.4.9 Side Bars

Side bars are vertical areas that stretch from the lower end of the head line to the top of the foot line. There can be a side bar at the left and one at the right (or even both). Side bars can show a table of contents, but they could also be added for purely aesthetic reasons.

When you install a side bar template, you must explicitly specify the horizontal size of the side bar. The vertical size is determined automatically. Each side bar can have its own background, which can be setup using special side background templates.

Adding a sidebar of a certain size, say 1cm, will make the main text 1cm narrower. The distance between the inner side of a side bar and the outer side of the text, as specified by the command `\beamersetleftmargin` and its counterpart for the right margin, is not changed when a side bar is installed.

Internally, the sidebars are typeset by showing them as part of the headline. The BEAMER class keeps track of six dimensions, three for each side: the variables `\beamer@leftsidebar` and `\beamer@rightsidebar` store the (horizontal) sizes of the side bars, the variables `\beamer@leftmargin` and `\beamer@rightmargin` store the distance between sidebar and text, and the macros `\Gm@lmargin` and `\Gm@rmargin` store the distance from the edge of the paper to the edge of the text. Thus the sum `\beamer@leftsidebar` and `\beamer@leftmargin` is exactly `\Gm@lmargin`. Thus, if you wish to put some text right next to the left side bar, you might write `\hskip-\beamer@leftmargin` to get there.

In the following, only the commands for the left side bars are listed. Each of these commands also exists for the right side bar, with "left" replaced by "right" everywhere.

`\useleftsidebartemplate{`⟨*horizontal size*⟩`}{`⟨*template*⟩`}`

When the side bar is typeset, the ⟨*template*⟩ is invoked inside a `\vbox` of the height of the side bar. Thus, the below example will produce a side bar of half a centimeter width, in which the word "top" is printed just below the head line and "bottom" is printed just above the foot line.

Example:

```
\useleftsidebartemplate{1cm}{
  top
  \vfill
  bottom
}
```

`\useleftsidebarbackgroundtemplate{`⟨*template*⟩`}`

The template is shown behind whatever is shown in the left side bar.

Example:

```
\useleftsidebarbackgroundtemplate
  {\color{red}\vrule height\paperheight width\beamer@leftsidebar}
```

`\useleftsidebarcolortemplate{`⟨*color expression*⟩`}`

Uses the given color as background for the side bar.

Example:

```
\useleftsidebarcolortemplate{\color{red}}
\useleftsidebarcolortemplate{\color[rgb]{1,0,0.5}}
```

`\useleftsidebarverticalshadingtemplate{`⟨*bottom color expression*⟩`}{`⟨*top color expression*⟩`}`

Installs a smooth vertical transition between the given colors as background for the side bar.

Example:

```
\useleftsidebarverticalshadingtemplate{white}{red}
```

`\useleftsidebarhorizontalshadingtemplate{`⟨*left end color expression*⟩`}{`⟨*right end color expression*⟩`}`

Installs a smooth horizontal transition between the given colors as background for the side bar.

Example:

```
\useleftsidebarhorizontalshadingtemplate{white}{red}
```

\usesectionsidetemplate{⟨*current section template*⟩}{⟨*other section template*⟩}

Both parameters should be \hboxes. The templates are used to typeset a section name inside a side navigation bar.

Example:

```
\usesectionsidetemplate
{\setbox\tempbox=\hbox{\color{black}\tiny{\kern3pt\insertsectionhead}}%
  \ht\tempbox=8pt%
  \dp\tempbox=2pt%
  \wd\tempbox=\beamer@sidebarwidth%
  \box\tempbox}
{\setbox\tempbox=\hbox{\color{structure!75}\tiny{\kern3pt\insertsectionhead}}%
  \ht\tempbox=8pt%
  \dp\tempbox=2pt%
  \wd\tempbox=\beamer@sidebarwidth%
  \box\tempbox}
```

\usesubsectionsidetemplate{⟨*current subsection template*⟩}{⟨*other subsection template*⟩}

See \usesectionsidetemplate.

Example:

```
\usesectionsidetemplate
{\setbox\tempbox=\hbox{\color{black}\tiny{\kern3pt\insertsectionhead}}%
  \ht\tempbox=8pt%
  \dp\tempbox=2pt%
  \wd\tempbox=\beamer@sidebarwidth%
  \box\tempbox}
{\setbox\tempbox=\hbox{\color{structure!75}\tiny{\kern3pt\insertsectionhead}}%
  \ht\tempbox=8pt%
  \dp\tempbox=2pt%
  \wd\tempbox=\beamer@sidebarwidth%
  \box\tempbox}
```

8.4.10 Buttons

Predefined Templates

\beamertemplateoutlinebuttons

Renders buttons as rectangles with rounded left and right border. Only the border (outline) is painted.

\beamertemplatesolidbuttons

Renders buttons as filled rectangles with rounded left and right border.

Template Installation Commands

\usebuttontemplate{⟨*button template*⟩}

Installs a new button template. This template is invoked whenever a button should be rendered.

Example:

```
\usebuttontemplate{\color{structure}\insertbuttontext}
```

Inserts

Inside the button template, the button text can be accessed via the following command:

\insertbuttontext

Inserts the text of the current button into a template. When called by button creation commands, like \beamerskipbutton, the symbol will be part of this text.

The button creation commands automatically add the following three inserts to the text to be rendered by \insertbuttontext:

\insertgotosymbol

Inserts a small right-pointing arrow.

\insertskipsymbol

Inserts a double right-pointing arrow.

\insertreturnsymbol

Inserts a small left-pointing arrow.

You can redefine these commands to change these symbols.

8.4.11 Navigation Bars

Predefined Templates

\beamertemplatecircleminiframe

Changes the symbols in a navigation bar used to represent a frame to a small circle.

\beamertemplatecircleminiframeinverted

Changes the symbols in a navigation bar used to represent a frame to a small circle, but with the colors inverted. Use this if the navigation bar is shown on a dark background.

\beamertemplatesphereminiframe

Changes the symbols in a navigation bar used to represent a frame to a small spheres.

\beamertemplatesphereminiframeinverted

Changes the symbols in a navigation bar used to represent a frame to a small spheres, but with the colors inverted. Use this if the navigation bar is shown on a `structure` background.

\beamertemplateboxminiframe

Changes the symbols in a navigation bar used to represent a frame to a small box.

\beamertemplateticksminiframe

Changes the symbols in a navigation bar used to represent a frame to a small vertical bar of varying length.

Template Installation Commands

\usesectionheadtemplate{⟨*current section template*⟩}{⟨*other section template*⟩}

The templates are used to render the section names in a navigation bar.

Example:

```
\usesectionheadtemplate
{\color{structure}\tiny\insertsectionhead}
{\color{structure!50}\tiny\insertsectionhead}
```

\usesubsectionheadtemplate{⟨*current subsection template*⟩}{⟨*other subsection template*⟩}

See `\usesectionheadtemplate`.

Example:

```
\usesubsectionheadtemplate
{\color{structure}\tiny\insertsubsectionhead}
{\color{structure!50}\tiny\insertsubsectionhead}
```

\useminislidetemplate{⟨*template current frame icon*⟩}{⟨*template current subsection frame icon*⟩}
{⟨*template other frame icon*⟩}{⟨*horizontal offset*⟩}{⟨*vertical offset*⟩}

The templates are used to draw frame icons in navigation bars. The offsets describe the offset between icons.

Example:

```
\useminislidetemplate
  {
    \color{structure}%
    \hskip-0.4pt\vrule height\boxsize width1.2pt%
  }
  {%
    \color{structure}%
    \vrule height\boxsize width0.4pt%
  }
  {%
    \color{structure!50}%
    \vrule height\boxsize width0.4pt%
  }
  {.1cm}
  {.05cm}
```

8.4.12 Navigation Symbols

Predefined Templates

\beamertemplatenavigationsymbolsempty

Suppresses all navigation symbols.

\beamertemplatenavigationsymbolsframe

Shows only the frame symbol as navigation symbol.

\beamertemplatenavigationsymbolsvertical

Organizes the navigation symbols vertically.

\beamertemplatenavigationsymbolshorizontal

Organizes the navigation symbols horizontally.

Template Installation Commands

\usenavigationsymbolstemplate{⟨*symbols template*⟩}

Installs a new symbols template. This template is invoked by themes at the place where the navigation symbols should be shown.

Example:

```
\usenavigationsymbolstemplate{\vbox{%
  \hbox{\insertslidenavigationsymbols}
  \hbox{\insertframenavigationsymbols}
  \hbox{\insertsubsectionnavigationsymbols}
  \hbox{\insertsectionnavigationsymbols}
  \hbox{\insertdocnavigationsymbols}
  \hbox{\insertbackfindforwardnavigationsymbols}}}
```

Inserts for this Template

The following inserts are useful for the navigation symbols template:

\insertslidenavigationsymbols

Inserts the slide navigation symbol, see Section 5.3.2.

\insertframenavigationsymbols

Inserts the frame navigation symbol, see Section 5.3.2.

\insertsubsectionnavigationsymbols

Inserts the subsection navigation symbol, see Section 5.3.2.

\insertsectionnavigationsymbols

Inserts the section navigation symbol, see Section 5.3.2.

\insertdocnavigationsymbols

Inserts the presentation navigation symbol and (if necessary) the appendix navigation symbol, see Section 5.3.2.

\insertbackfindforwardnavigationsymbols

Inserts a back, a find, and a forward navigation symbol, see Section 5.3.2.

8.4.13 Footnotes

Template Installation Commands

\usefootnotetemplate{⟨*footnote template*⟩}

Example:

```
\usefootnotetemplate{
  \parindent 1em
  \noindent
  \hbox to 1.8em{\hfil\insertfootnotemark}\insertfootnotetext}
```

Inserts for these Templates

\insertfootnotemark

Inserts the current footnote mark (like a raised number) into a template.

\insertfootnotetext

Inserts the current footnote text into a template.

8.4.14 Captions

Predefined Templates

\beamertemplatecaptionwithnumber

Changes the caption template such that the number of the table or figure is also shown.

\beamertemplatecaptionownline

Changes the caption template such that the word “Table” or “Figure” has its own line.

Template Installation Commands

\usecaptiontemplate{⟨*caption template*⟩}

Example:

```
\usecaptiontemplate{
  \small
  \structure{\insertcaptionname~\insertcaptionnumber:}
  \insertcaption
}
```

Inserts for these Templates

\insertcaption

Inserts the text of the current caption into a template.

\insertcaptionname

Inserts the name of the current caption into a template. This word is either “Table” or “Figure” or, if the `babel` package is used, some translation thereof.

\insertcaptionnumber

Inserts the number of the current figure or table into a template.

8.4.15 Lists (Itemizations, Enumerations, Descriptions)

Predefined Templates

\beamertemplatedotitem

Changes the symbols shown in an `itemize` environment to dots.

\beamertemplateballitem

Changes the symbols shown in an `itemize` environment to small plastic balls.

Template Installation Commands

\useenumerateitemtemplate{⟨*template*⟩}

The ⟨*template*⟩ is used to render the default item in the top level of an enumeration.

Example: `\useenumerateitemtemplate{\insertenumlabel}`

\useitemizeitemtemplate{⟨*template*⟩}

The ⟨*template*⟩ is used to render the default item in the top level of an itemize list.

Example: `\useitemizeitemtemplate{\pgfuseimage{mybullet}}`

\usesubitemizeitemtemplate{⟨*template*⟩}

The ⟨*template*⟩ is used to render the default item in the second level of an itemize list.

Example: `\usesubitemizeitemtemplate{\pgfuseimage{mysubbullet}}`

\useitemizetemplate{⟨*begin text*⟩}{⟨*end text*⟩}

The ⟨*begin text*⟩ is inserted at the beginning of a top-level itemize list, the ⟨*end text*⟩ at its end.

Example: `\useitemizetemplate{}{}`

\usesubitemizetemplate{⟨*begin text*⟩}{⟨*end text*⟩}

The ⟨*begin text*⟩ is inserted at the beginning of a second-level itemize list, the ⟨*end text*⟩ at its end.

Example: `\usesubitemizetemplate{\begin{small}}{\end{small}}`

\useenumerateitemtemplate{⟨*template*⟩}

The ⟨*template*⟩ is used to render the default item in the top-level of an enumeration.

Example: `\useenumerateitemtemplate{\insertenumlabel}`

\usesubenumerateitemtemplate{⟨*template*⟩}

The ⟨*template*⟩ is used to render the default item in the second level of an enumeration.

Example: `\usesubenumerateitemtemplate{\insertenumlabel-\insertsubenumlabel}`

\useenumeratetemplate{⟨*begin text*⟩}{⟨*end text*⟩}

The ⟨*begin text*⟩ is inserted at the beginning of a top-level enumeration, the ⟨*end text*⟩ at its end.

Example: `\useenumeratetemplate{}{}`

\usesubenumeratetemplate{⟨*begin text*⟩}{⟨*end text*⟩}

The ⟨*begin text*⟩ is inserted at the beginning of a second-level enumeration, the ⟨*end text*⟩ at its end.

Example: `\usesubenumeratetemplate{\begin{small}}{\end{small}}`

\usedescriptiontemplate{⟨*description template*⟩}{⟨*default width*⟩}

The ⟨*default width*⟩ is used as width of the default item, if no other width is specified; the width `\labelsep` is automatically added to this parameter.

Example: `\usedescriptionitemtemplate{\color{structure}\insertdescriptionitem}{2cm}`

Inserts for these Templates

\insertdescriptionitem

Inserts the current item of a description environment into a template.

\insertenumlabel

Inserts the current number of the top-level enumeration (as an Arabic number) into a template.

\insertsubenumlabel

Inserts the current number of the second-level enumeration (as an Arabic number) into a template.

8.4.16 Hilighting Commands

Template Installation Commands

\usealerttemplate{⟨*alert template*⟩}

Example: `\usealerttemplate{{\color{red}\insertalert}}`

\usestructuretemplate{⟨*structure template*⟩}

Example: `\usestructuretemplate{{\color{structure}\insertstructure}}`

Inserts for these Templates

\insertalert

Inserts the current alerted text into a template.

\insertstructure

Inserts the current structure text into a template.

8.4.17 Block Environments

Predefined Templates

\beamertemplateboldblocks

Block titles are printed in bold.

\beamertemplatelargeblocks

Block titles are printed slightly larger.

\beamertemplateroundedblocks

Changes the block templates such that they are printed on a rectangular area with rounded corners.

\beamertemplateshadowblocks

Changes the block templates such that they are printed on a rectangular area with rounded corners and a shadow.

Template Installation Commands

\useblocktemplate{⟨*block beginning template*⟩}{⟨*block end template*⟩}

Example:

```
\useblocktemplate
  {%
   \medskip%
    {\color{blockstructure}\textbf{\insertblockname}}%
    \par%
  }
  {\medskip}
```

\usealertblocktemplate{⟨*block beginning template*⟩}{⟨*block end template*⟩}

Example:

```
\usealertblocktemplate
  {%
    \medskip
    {\alert{\textbf{\insertblockname}}}%
  \par}
  {\medskip}
```

\useexampleblocktemplate{⟨*block beginning template*⟩}{⟨*block end template*⟩}

Example:

```
\useexampleblocktemplate
  {%
    \medskip
    \begingroup\color{darkgreen}{\textbf{\insertblockname}}
    \par}
  {%
     \endgroup
     \medskip
  }
```

Inserts for these Templates

\insertblockname

Inserts the name of the current block into a template.

9 License: The GNU Public License, Version 2

The BEAMER class is distributed under the GNU public license, version 2. In detail, this means the following (the following text is copyrighted by the Free Software Foundation):

9.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

9.2 Terms and Conditions For Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based

on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or

otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

9.3 No Warranty

10. Because the program is licensed free of charge, there is no warranty for the program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

11. In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.