User's Guide to the Beamer Class, Version 2.21

`http://latex-beamer.sourceforge.net`

Till Tantau

`tantau@users.sourceforge.net`

July 6, 2004

Contents

1 Introduction

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`.

1.1 Getting Started with the Beamer Class and LaTeX/pdfLaTeX

To use the BEAMER class together with `latex` or `pdflatex`, 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 Getting Started with the Beamer Class and LyX

Once installed (see Section 2), using the BEAMER class together with LyX is quite easy: You open a new file and choose `beamer` as the document class. It is often even easier to choose "New from template" and to pick a template from the directory `beamer/lyx/templates`.

To reproduce the example from the previous subsection in LyX, proceed as follows:

- The command `\usepackage{beamerthemesplit}` must be added to the preamble. You can edit the preamble using Layout ⟩ Document ⟩ Preamble.
- Typeset the author and date the usual way, using the styles Author and Date. The title page will then be created automatically.
- To insert the sections and subsections, use the usual Section and Subsection styles.
- To insert the frame containing the table of contents, insert a line of style BeginFrame. Since this frame has no title, do not write anything on the line with style BeginFrame. Next, insert a line of style Standard and use Insert ⟩ Insert TOC to insert the table of contents. Optionally, end the frame using a line of style EndFrame (the following Section style automatically closes the frame).
- To create the last frame, start a new frame using the style BeginFrame. Write the frame title on the line having this style.
- Use the Itemize style to create the itemized text.
- Add the overlay specifications (the texts like `<1->`) to the items by entering TeX-mode (press on the little TeX icon) and writing `<1->`. This TeX text should be placed right at the beginning of the item.
- You must end this frame using the style EndFrame (sadly, the end of the document and also the beginning of the appendix do not automatically end the last frame – whereas the start of a frame, section, part, or subsection does).

Now use View ⟩ PDF to view the resulting presentation. On a slow machine, this may take a while. See Section 3.5.7 for ways of speeding up the compilation.

1.3 How to Read this User's Guide

This user's 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.

You will sometimes find one of the words BEAMER, ARTICLE, or LYX in blue in some description of a command or environment. The first indicates that the description applies only to "normal beamer operation in LaTeX." The word ARTICLE describes some behaviour that is special for the `article` mode. The word LYX describes behaviour that is special when you use LyX to prepare your presentation.

2 Installation

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 Versions

This documentation is part of version 2.21 of the BEAMER class. BEAMER needs a reasonably recent version of LATEX and of several standard packages to run. It also needs the following versions of rather special packages:

1. `pgf` version 0.62 and
2. `xcolor` version 2.00.

This version of BEAMER should also work with more recent versions of `pgf` and `xcolor`, but I cannot guarantee this. In particular, `xcolor` is developped by Uwe Kern, not by me, and BEAMER messes around with the `xcolor` internals in a not-so-friendly way. So whenever Uwe Kern changes `xcolor` internals (to which he is perfectly entitled), things can break down. We are working on removing `beamer`'s dependency on internals of `xcolor`, but that is still work-in-progress.

2.2 Installing Prebundled Packages

I do not create or manage prebundled packages of BEAMER, but, fortunately, nice other people do. I cannot give detailed instructions on how to install these packages, since I do not manage them, but I *can* tell you were to find them. You install them the "usual way" you install packages. If anyone has any hints and additional information on this, please email me.

For Debian, you need the packages at

```
http://packages.debian.org/latex-beamer
http://packages.debian.org/pgf
http://packages.debian.org/latex-xcolor
```

For MiKTEX, you need the packages called `latex-beamer`, `pgf` and `xcolor`.

2.3 Installation in a texmf Tree

If, for whatever reason, you do not wish to use a prebundled package, the "right" way to install BEAMER is to put it in a so-called `texmf` tree. In the following, I explain how to do this.

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 the PGF package, which can be found at the same place. Finally, you need the XCOLOR package, which can also be found at that place (although the version on CTAN might be newer).

LYX For usage with LYX, version 1.3.3 of LYX is known to work. I have not tried other versions; they might also work.

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). You now need to put these files 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 usually located in your home directory at `~/texmf/` or `~/Library/texmf/`.

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`,
- `texmf/tex/latex/pgf`, and
- `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.

LYX For usage of the BEAMER class with LYX, you have to do all of the above. Then you also have to make LYX aware of the file `beamer/lyx/layouts/beamer.layout`. To do so, link (or, not so good in case of later updates, copy) this file to the directory `.lyx/layouts/` in your home directory. Then use LYX's Reconfigure command to make LYX aware of this file.

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.

LYX To test the LYX installation, try creating a new file from the template `beamerpresentation.lyx`, which is located in the directory `beamer/lyx/templates`.

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

BEAMER 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 distribution) 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 8.4.2 for details.

LYX You can either open a new file and then select `beamer` as the document class or you say "New from template" and then use a template from the directory `beamer/lyx/templates`.

3.3 Step Two: Structure Your 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 that 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 that you do not actually intend to talk about, but that might come in handy when questions are asked.

3.4 Step Three: Creating a PDF or PostScript File

BEAMER 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{...}

\frame{} % to enforce entries in the table of contents

\end{document}
```

The empty frame at the end (which should be deleted later) ensures that the sections and subsections are actually part of the table of contents. This frame is necessary since a `\section` or `\subsection` command following the last page of a document have no effect.

LYX Use "View" to check whether the presentation compiles fine. Note that you must put the table of contents inside a frame, but that the title page is created automatically.

3.4.1 Creating PDF

BEAMER 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 auxiliary 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" in the printer dialog 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. If you put several slides on one page and if these slides normally have a white background, it may be useful to write the following in your preamble:

```
\mode<handout>{\beamertemplatesolidbackgroundcolor{black!5}}
```

This will cause the slides of the handout version to have a very light gray background. This makes it easy to discern the slides' border if several slides are put on one page.

LYX Choose "View pdf" to view your presentation.

3.4.2 Creating PostScript

BEAMER 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. The same trick as for the PDF-version can be used to make the borders of slides more pronounced in the handout version.

You can convert a PostScript file to a pdf file using

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

LYX Use "View Postscript" to view the PostScript version.

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 guidelines, but you should be aware of it when you ignore a rule and you should be able to justify it to yourself.

LYX To create a frame, use the style "BeginFrame". The frame title is given on the line of this style. The frame ends automatically with the start of the next frame, with a section or subsection command, and with an empyt line in the sylte "EndFrame". Note that the last frame of your presentation must be ended using "EndFrame" and that the last frame before the appendix must be ended this way.

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 Titles

- 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.
- The title should really *explain* things, not just give a cryptic summary that cannot be understood unless one has understood the whole slide. For example, a title like "The Poset" will have everyone puzzled what this slide might be about. Titles like "Review of the Definition of Partially Ordered Sets (Posets)" or "A Partial Ordering on the Columns of the Genotype Matrix" are *much* more informative.
- Ideally, titles on consecutive frames should "tell a story" all by themselves.
- In English, you should *either always* capitalize all words in a frame title except for words like "a" or "the" (as in a title), *or* you *always* use the normal lowercase letters. Do *not* mix this; stick to one rule. The same is true for block titles. For example, do not use titles like "A short Review of Turing machines." Either use "A Short Review of Turing Machines." or "A short review of Turing machines." (Turing is still spelled with a capital letter since it is a name).
- In English, the title of the whole document should be capitalized, regardless of whether you capitalize anything else.
- In German and other languages that have lots of capitalized words, always use the correct upper-/lowercase letters. Never capitalize anything in addition to what is usually capitalized.

3.5.3 Guidelines on the Body Text

- *Never* use a smaller font size to "fit 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" and 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.

3.5.4 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 7.1.

3.5.5 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.

3.5.6 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.

3.5.7 Ways of Improving Compilation Speed

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, you can do several things to speedup the compilation. First, you can use the `draft` class option.

`\documentclass[draft]{beamer}`

Causes the headlines, footlines, 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.

Second, you can use the following command:

`\includeonlyframes{`⟨*frame label list*⟩`}`

This command behaves a little bit like the `\includeonly` command: Only the frames mentioned in the list are included. All other frames are suppressed. Nevertheless, the section and subsection commands are still executed, so that you still have the correct navigation bars. By labeling the current frame as, say, `current` and then saying `\includeonlyframes{current}`, you can work on a single frame quickly.

The ⟨*frame label list*⟩ is a comma-separated list (without spaces) of the names of frames that have been labeled. To label a frame, you must pass the option `label=`⟨*name*⟩ to the `\frame` command.

Example:

```
\includeonlyframes{example1,example3}

\frame[label=example1]
{This frame will be included. }

\frame[label=example2]
{This frame will not be included. }

\frame{This frame will not be included.}

\againframe{example1} % Will be included
```

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 8.1. 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 `llncs` 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 8.4.

LYX Creating an article version is not really possible in LYX. You can *try*, but I would not advise it.

4 Frames and Overlays

This section explains how you can create frames and overlays. It starts with a description of a general concept, called *overlay specifications*. Nearly all of BEAMER's commands for creating frames and overlays are based on this concept, except for the simple `\pause` command (though it is internally also mapped to this concept).

4.1 The Concept of Overlay Specifications

4.1.1 The General Concept

When creating overlays, how do you specify on which slides of a series of slides a certain text should be shown? (Such a series is called a *frame* in BEAMER.) The approach taken by most presentation classes is to introduce new commands, which get a certain slide number as input and which affect the text on the slide following this command in a certain way. For example, PROSPER's `\FromSlide{2}` command causes all text following this command to be shown only from the second slide on.

The BEAMER class uses a different approach (though the abovementioned command is also available: `\onslide<2->` will have the same effect as `\FromSlide{2}`, expect that `\onslide` transcends environments). The idea is to add *overlay specifications* to certain commands. These specifications are always given in pointed brackets and follow the command "as soon as possible," though in certain cases BEAMER also allows overlay specification to be given a little later. In the simplest case, the specification contains just a number. A command with an overlay specification following it will only have "effect" on the slide(s) mentioned in the specification. What exactly "having an 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.

For a second example, consider the following frame:

```
\frame
{
  \only<1>{This line is inserted only on slide 1.}
  \only<2>{This line is inserted only on slide 2.}
}
```

The command `\only`, which is introduced by BEAMER, normally simply inserts its parameter into the current frame. However, if an overlay-specification is present, it "throws away" its parameter on slides that are not mentioned.

Overlay specifications can only be written behind certain commands, not every command. Which commands you can use and which effects this will have is explained in Section 4.3.2. However, it is quite easy to redefine an existing command such that it becomes "overlay specification aware," see also Section 4.3.2.

The syntax of (basic) overlay specifications is the following: They are comma-separated lists of slides and ranges. Ranges are specified like this: `2-5`, which means slide two through to five. The start or the end 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 selects the slides 1, 2, 3, 6, 7, 8, 10, 12, 13, 14, and 15.

LYX Overlay specifications can also be given in LyX. You must give them in TEX-mode (otherwise the pointed brackets may be "escaped" by LyX, though this will not happen in all versions). For example, to add an overlay specification to an item, simply insert a TEX-mode text like `<3>` as the first thing in that item. Likewise, you can add an overlay specification to environments like `theorem` by giving them in TEX-mode right at the start of the environment.

4.1.2 Mode Specifications

This subsection is only important if you use BEAMER's mode mechanism to create different versions of your presentation. If you are not familiar with BEAMER's modes, please skip this section or read Section 8 first.

In certain cases you may wish to have different overlay specifications to apply to a command in different modes. For example, you might wish a certain text to appear only from the third slide on during your presentation, but in a handout for the audience there should be no second slide and the text should appear already on the second slide. In this case you could write

```
\only<3| handout:2>{Some text}
```

The vertical bar, which must be followed by a (white) space, separates the two different specifications `3` and `handout:2`. By writing a mode name before a colon, you specify that the following specification only applies to that mode. If no mode is given, as in `3`, the mode `beamer` is automatically added. For this reason, if you write `\only<3>{Text}` and you are in `handout` mode, the text will be shown on all slides since there is no restriction specified for handouts and since the `3` is the same as `beamer:3`.

It is also possible to give an overlay specification that contains only a mode name (or several, separated by vertical bars):

```
\only<article>{This text is shown only in article mode.}
```

An overlay specification that does not contain any slide numbers is called a (pure) *mode specification*. If a mode specification is given, all modes that are not mentioned are automatically suppressed. Thus `<beamer:1->` means "on all slides in `beamer` mode and also on all slides in all other modes, since nothing special is specified for them," whereas `<beamer>` means "on all slides in `beamer` mode and not on any other slide."

Mode specifications can also be used outside frames as in the following examples:

```
\section<presentation>{This section exists only in the presentation modes}
\section<article>{This section exists only in the article mode}
```

You can also mix pure mode specifications and overlay specifications, although this can get confusing:

```
\only<article| beamer:1>{Riddle}
```

This will cause the text `Riddle` to be inserted in `article` mode and on the first slide of a frame in `beamer` mode, but not at all in `handout` or `trans` mode. (Try to find out how `<beamer| beamer:1>` differs from `<beamer>` and from `<beamer:1>`.)

4.1.3 Action Specifications

This subsection also introduces a rather advanced concept. You may also wish to skip it on first reading.

Some overlay-specification-aware commands cannot only handle normal overlay specifications, but also so called *action specifications*. In an action specification, the list of slide numbers and ranges is prefixed by ⟨*action*⟩`@`, where ⟨*action*⟩ is the name of a certain action to be taken on the specified slides:

```
\item<3-| alert@3> Shown from slide 3 on, alerted on slide 3.
```

In the above example, the `\item` command, which allows actions to be specified, will uncover the item text from slide three on and it will, additionally, alert this item exactly on slide 3.

Not all commands can take an action specification. Currently, only `\item` (though not in `article` mode currently), `\action`, the environment `actionenv`, and the block environments (like `block` or `theorem`) handle them.

By default, the following actions are available:

- `alert` alters the item or block.
- `uncover` uncovers the item or block (this is the default, if no action is specified).
- `only` causes the whole item or block to be inserted only on the specified slides.
- `visible` causes the text to become visible only on the specified slides (the difference between `uncover` and `visible` is the same as between `\uncover` and `\visible`).
- `invisible` causes the text to become invisble on the specified slides.

The rest of this section explains how you can add your own actions and make commands action-specification-aware. You may wish to skip it upon first reading.

You can easily add your own actions: An action specification like ⟨*action*⟩`@`⟨*slide numbers*⟩ simply inserts an environment called ⟨*action*⟩`env` around the `\item` or parameter of `\action` with `<`⟨*slide numbers*⟩`>` as overlay specification. Thus, by defining a new overlay-specification-aware environment named ⟨*my action name*⟩`env`, you can add your own action:

```
\newenvironment{checkenv}{\only{\useitemizeitemtemplate{X}}}{}
```

You can then write

```
\item<beamer:check@2> Text.
```

This will change the itemization symbol before `Text.` to `X` on slide 2 in `beamer` mode. The definition of `checkenv` used the fact that `\only` also accepts an overlay-specification given after its argument.

The whole action mechanism is base on the following environment:

`\begin{actionenv}<`⟨*action specification*⟩`>`
⟨*environment contents*⟩
`\end{actionenv}`

This environment extracts all actions from the ⟨*action specification*⟩ for the current mode. For each action of the form ⟨*action*⟩`@`⟨*slide numbers*⟩, it inserts the following text: `\begin{`⟨*action*⟩`env}<`⟨*slide number*⟩`>` and the beginning of the environment and the text `\end{`⟨*action*⟩`env}` at the end. If there are several action specifications, several environments are opened (and closed in the appropriate order). An ⟨*overlay specification*⟩ without an action is promoted to `uncover@`⟨*overlay specification*⟩.

If the so called *default overlay specification* is not empty, it will be used in case no ⟨*action specification*⟩ is given. The default overlay specification is usually just empty, but it may be set either by providing an additional optional argument to the command `\frame` or to the environments `itemize`, `enumerate`, or `description` (see these for details). Also, the default action specification can be set using the command `\beamerdefaultoverlayspecification`, see below.

Example:

```
\frame
{
  \begin{actionenv}<2-| alert@3-4,6>
    This text is shown the same way as the text below.
  \end{actionenv}

  \begin{uncoverenv}<2->
    \begin{alertenv}<3-4,6>
      This text is shown the same way as the text above.
    \end{alertenv}
  \end{uncoverenv}
}
```

`\action<`⟨*action specification*⟩`>{`⟨*text*⟩`}`

This has the same effect as putting ⟨*text*⟩ in an `actionenv`.

Example: `\action<alert@2>{Could also have used \alert<2>{}.}`

`\beamerdefaultoverlayspecification{`⟨*default overlay specification*⟩`}`

Locally sets the default overlay specification to the given value. This overlay specification will be used in every `actionenv` environment and every `\item` that does not have its own overlay specification. The main use of this command is to install an incremental overlay specification like `<+->` or `<+-| alert@+>`, see Section 4.1.4.

Usually, the default overlay specification is installed automatically by the optional arguments to `\frame`, `itemize`, `enumerate`, and `description`. You will only have to use this command if you wish to do funny things.

Example: `\beamerdefaultoverlayspecification{<+->}`

Example: `\beamerdefaultoverlayspecification{}` clears the default overlay specification. (Actually, it installs the default overlay specification `<*>`, which just means "always," but the "portable" way of clearing the default overlay specification is this call.)

4.1.4 Incremental Specifications

This subsection is mostly important for people who have already used overlay specifications a lot and have grown tired of writing things like `<1->`, `<2->`, `<3->`, and so on again and again. You should skip this section on first reading.

Often you want to have overlay specifications that follow a pattern similar to the following:

```
\begin{itemize}
\item<1-> Apple
\item<2-> Peach
\item<3-> Plum
\item<4-> Orange
\end{itemize}
```

The problem starts if you decide to insert a new fruit, say, at the beginning. In this case, you would have to adjust all of the overlay specifications. Also, if you add a `\pause` command before the `itemize`, you would also have to update the overlay specifications.

BEAMER offers a special syntax to make creating lists as the one above more "robust." You can replace it by the following list of *incremental overlay specifications*:

```
\begin{itemize}
\item<+-> Apple
\item<+-> Peach
\item<+-> Plum
\item<+-> Orange
\end{itemize}
```

The effect of the `+`-sign is the following: You can use it in any overlay specification at any point where you would usually use a number. If a `+`-sign is encountered, it is replaced by the current value of the LaTeX counter `beamerpauses`, which is 1 at the beginning of the frame. Then the counter is increased by 1, though it is only increased once for every overlay specification, even if the specification contains multiple `+`-signs (they are replaced by the same number).

In the above example, the first specification is replaced by `<1->`. Then the second is replaced by `<2->` and so forth. We can now easily insert new entries, without having to change anything. We might also write the following:

```
\begin{itemize}
\item<+-| alert@+> Apple
\item<+-| alert@+> Peach
\item<+-| alert@+> Plum
\item<+-| alert@+> Orange
\end{itemize}
```

This will alert the current item when it is uncovered. For example, the first specification `<+-| alert@+>` is replaced by `<1-| alert@1>`, the second is replaced by `<2-| alert@2>`, and so on. Since the `itemize` environment also allows you to specify a default overlay specification, see the documentation of that environment, the above example can be written even more economically as follows:

```
\begin{itemize}[<+-| alert@+>]
\item Apple
\item Peach
\item Plum
\item Orange
\end{itemize}
```

The `\pause` command also updates the counter `beamerpauses`. You can change this counter yourself using the normal LaTeX commands `\setcounter` or `\addtocounter`.

Any occurence of a `+`-sign may be followed by an *offset* in round brackets. This offset will be added to the value of `beamerpauses`. Thus, if `beamerpauses` is 2, then `<+(1)->` expands to `<3->` and `<+(-1)-+>` expands to `<1-2>`.

4.2 Frames

4.2.1 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 is shown on all slides of the frame. If a frame contains commands that have an overlay specification, the frame will contain multiple slides; otherwise it contains only one slide.

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

The ⟨*overlay specification*⟩ dictates which slides of a frame are to be shown. If left out, the number is calculated automatically. The ⟨*frame text*⟩ can be normal LaTeX text, but may not contain `\verb` commands or `verbatim` environments, unless the `containsverbatim` options is given, see also Section 4.2.4.

Example:

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

Example:

```
\frame<beamer>  % frame is only shown in beamer mode
{
  \frametitle{Outline}
  \tabelofcontent[current]
}
```

Normally, the complete ⟨*frame text*⟩ is put on a slide. If the text does not fit on a slide being too high, it will be squeezed as much as possible, a warning issued, and the text just extends unpleasantly over the bottom. You can use the option `allowframebreaks` to cause the ⟨*frame text*⟩ to be split among several slides, though you cannot use overlays then. See the explanation of the `allowframebreaks` option for details.

The ⟨*default overlay specification*⟩ is an optional argument that is "detected" according to the following rule: If the first optional argument in square brackets starts with a `<`, then this argument is a ⟨*default overlay specification*⟩, otherwise it is a normal ⟨*options*⟩ argument. Thus `\frame[<+->][plain]` would be legal, but also `\frame[plain]`.

The effect of the ⟨*default overlay specification*⟩ is the following: Every command or environment *inside the frame* that accepts an action specification, see Section 4.1.3, (this includes the `\item` command, the `actionenv` environment, `\action`, and all block environments) and that is not followed by an overlay specification gets the ⟨*default overlay specification*⟩ as its specification. By providing an incremental specification like `<+->`, see Section 4.1.4, this will essentially cause all blocks and all enumerations to be uncovered piece-wise (blocks internally employ action specifications).

Example: In this frame, the theorem is shown from the first slide on, the proof from the second slide on, with the first two itemize points shown one after the other; the last itemize point is shown together with the first one. In total, this frame will contain four slides.

```
\frame[<+->]
{
  \begin{theorem}
    $A = B$.
  \end{theorem}
  \begin{proof}
    \begin{itemize}
    \item Clearly, $A = C$.
    \item As shown earlier,  $C = B$.
    \item<3-> Thus $A = B$.
    \end{itemize}
  \end{proof}
}
```

The following ⟨*options*⟩ may be given:

- `allowdisplaybreaks=`⟨*break desirability*⟩ causes the AMSTEX command `\allowdisplaybreaks` `[`⟨*break desirability*⟩`]` to be issued for the current frame. The ⟨*break desirability*⟩ can be a value between 0 (meaning formulas may never be broken) and 4 (the default, meaning that formulas can be broken anywhere without any penalty). The option is just a convenience and makes sense only together with the `allowsframebreaks` option.
- `allowframebreaks=`⟨*fraction*⟩. When this option is given, the frame will be automatically broken up into several frames, if is does not fit on a single slide. In detail, when this option is given, the following things happen:
  1. The option `containsverbatim` is automatically selected, as a side-effect. Thus, frames with this option set may contain verbatim text.
  2. Consequently, overlays are not supported.
  3. Any notes for the frame created using the `\note` command will be inserted after the first page of the frame.
  4. Any footnotes for the frame will be inserted at the last page of the frame.
  5. If there is a frame title, each of the pages will have this frame title, with a special note added indicating which page of the frame that page is. By default, this special note is a Roman number. However, this can be changed by changing the continutation template, see Section 9.4.8.

  If a frame needs to be broken into several pages, the material on all but the last page fills only 95% of each page by default. Thus, there will be some space left at the top and/or bottom, depending on the vertical placement option for the frame. This yields a better visual result than a 100% filling, which typically looks crowded. However, you can change this percentage using the optional argument ⟨*fraction*⟩, where 1 means 100% and 0.5 means 50%. This percentage includes the frame title. Thus, in order to split a frame "roughly in half," you should give 0.6 as ⟨*fraction*⟩.

  Most of the fine details of normal TEX page breaking also apply to this option. For example, when you wish equations to be broken automatically, be sure to use the `\allowdisplaybreaks` command. You can insert `\break`, `\nobreak`, and `\penalty` commands to control where breaks should occur. The commands `\pagebreak` and `\nopagebreak` also work, including their options. Since you typically do not want page breaks for the frame to apply also to the `article` mode, you can add a mode specification like `<presentation>` to make these commands apply only to the presentation modes. The command `\framebreak` is a shorthand for `\pagebreak<presentation>` and `\noframebreak` is a shorthand for `\nopagebreak<presentation>`.

  The use of this option *evil.* In a (good) presentation you prepare each slide carefully and think twice before putting something on a certain slide rather than on some different slide. Using the `allowframebreaks` option invites the creation of horrible, endless presentations that resemble more a "paper projected on the wall" than a presentation. Nevertheless, the option does have its uses. Most noticably, it can be convenient for automatically splitting bibliographies or long equations.

  Example:

  ```
  \frame[allowframebreaks]
  {
    \frametitle{References}

    \begin{thebibliography}{XX}
  ```

```
  \bibitem...
  \bibitem...
    ...
  \bibitem...
  \end{thebibliography}
}
```

Example:

```
\frame[allowframebreaks,allowdisplaybreaks]
{
  \frametitle{A Long Equation}

  \begin{align}
    \zeta(2) &= 1 + 1/4 + 1/9 + \cdots \\
    &= ... \\
    ...
    &= \pi^2/6.
  \end{align}
}
```

- `b`, `c`, `t` will cause the frame to be vertically aligned at the bottom/center/top. This overrides the global placement policy, which is governed by the class options `slidestop` and `slidescentered`.
- `containsverbatim` tells BEAMER that the frame contains verbatim commands. In this case, only one slide of the frame is typeset (unless all slides are suppressed by the ⟨*overlay specification*⟩). If you wish to use verbatim text in a frame with several slides, a more roundabout approach is necessary, see Section 4.2.4. This option cannot be used together with the `label` option.
- `label=`⟨*name*⟩ causes the frame's contents to be stored under the name ⟨*name*⟩ for later resumption using the command `\againframe`. 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 `\againframe`.

  Furthermore, on each slide of the frame a label with the name ⟨*name*⟩`<`⟨*slide number*⟩`>` is created. On the *first* slide, furthermore, a label with the name ⟨*name*⟩ is created (so the labels ⟨*name*⟩ and ⟨*name*⟩`<1>` point to the same slide). Note that labels in general, and these labels in particular, can be used as targets for hyperlinks.
- `plain` causes the headlines, footlines, and sidebars to be suppressed. This is useful for creating single frames with different head- and footlines or for creating frames showing big pictures that completely fill the frame.

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

```
\frame[plain]{%
  \begin{centering}%
    \pgfimage[height=\paperheight]{somebigimagefile}%
    \par%
  \end{centering}%
}
```

  Example: A title page, in which the head- and footlines 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}
```

- shrink=⟨*minimum shrink percentage*⟩. This option will cause the text of the frame to be shrunken if it is too large to fit on the frame. BEAMER will first normally typeset the whole frame. Then it has a look at vertical size of the frame text (excluding the frame title). If this vertical size is larger than the text height minus the frame title height, BEAMER computes a shrink factor and scales down the frame text by this factor such that the frame text then fills the frame completely. Using this option will automatically cause the `squeeze` option to be used, also.

  Since the shrinking takes place only after everything has been typeset, shrunken frame text will not fill the frame completely horizontally. For this reason, you can specify a ⟨*minimum shrink percentage*⟩ like 20. If this percentage is specified, the frame will be shrunk by *at least* by this percentage. Since BEAMER knows this, it can increase the horizontal width proportionally such that the shrunken text once more fills the entire frame. If, however, the percentage is not enough, the text will be shrunken as needed and you will be punished by a warning message.

  The best way to use this option is to identify frames that are overly full, but in which all text absolutely has to be fit on a single frame. Then start specifying first `shrink=5`, then `shrink=10`, and so on, until no warning is issued any more (or just ignore the warning when things look satisfactory).

  Using this option is *very evil.* It will result in changes of the font size from slide to slide, which is a typographic nightmare. Its usage can always be avoided by restructuring and simplifying frames, which will result in a better presentation.

  Example:

  ```
  \frame[shrink=5]
  {
    Some evil endless slide that is 5\% too large.
  }
  ```

- squeeze causes all vertical spaces in the text to be squeezed together as much as possible. Currently, this just causes the vertical space in enumerations or itemizations to be reduced to zero.

  Using this option is not good, but also not evil.

LYX Use the style "BeginFrame" to start a frame and the style "EndFrame" to end it. A frame is automatically ended by the start of a new frame and by the start of a new section or subsection (but not by the end of the document!).

LYX You can pass options and an overlay specification to a frame by giving these in TEX-mode as the first thing in the frame title. (Some magic is performed to extract them in LYX mode from there.)

LYX The style "BeginPlainFrame" is included as a convenience. It passes the `plain` option to the frame. To pass further options to a plain frame, you should use the normal "BeginFrame" style and specify all options (include `plain`).

LYX In LYX, you can insert verbatim text directly even in overlayed frames. The reason is that LYX uses a different internal mechanism for typesetting verbatim text, that is easier to handle for BEAMER.

ARTICLE In `article` mode, the `\frame` command does not create any visual reference to the original frame (no frame is drawn). Rather, the frame text is inserted into the normal text. To change this, you can modify the frame template, see Section 9.4.4. To suppress a frame in `article` mode, you can, for example, specify `<presentation>` as overlay specification.

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 simply have not been able to come up with an idea of how to implement it in the presence of multiple overlays.

4.2.2 Components of a Frame

Each frame consists of several components:

1. a headline,
2. a footline,
3. a left sidebar,
4. a right sidebar,
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 9.4.9 for the head- and footlines and Section 9.4.10 for the sidebars. 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}}
```

ARTICLE This command has no effect.

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

`\frametitle<`⟨*overlay specification*⟩`>[`⟨*short frame title*⟩`]{`⟨*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. The ⟨*short frame title*⟩ is normally not shown, but its available via the `\insertshortframetitle` command. The ⟨*overlay specification*⟩ is mostly useful for suppressing the frame title in `article` mode.

Example:

```
\frame{
  \frametitle{A Frame Title is Important.}
  \framesubtitle{Subtitles are not so important.}

  Frame contents.
}
```

If you are using the `allowframebreaks` option with the current frame, a continuation text (like "(cont.)" or something similar, depending on the continuation template) is automatically added to the ⟨*frame title text*⟩ at the end.

BEAMER The frame title is not typeset immediately when the command `\frametitle` is encountered. Rather, the argument of the command is stored internally and the frame title is only typeset when the complete frame has been read. This gives you access to both the ⟨*frame title text*⟩ and to the ⟨*subframe title text*⟩ that is possibly introduced using the `\framesubtitle` command.

ARTICLE By default, this command creates a new paragraph in `article` mode, entitled ⟨*frame title text*⟩. Using the ⟨*overlay specification*⟩ makes it easy to suppress the a frame title once in a while. If you generally wish to suppress *all* frame titles in `article` mode, say `\useframetitletemplate{}`.

LYX The frame title is the text that follows on the line of the "BeginFrame" style.

`\framesubtitle<`*⟨overlay specification⟩*`>{`*⟨frame subtitle text⟩*`}`

If present, a subtitle will be shown in a smaller font below the main title. Like the `\frametitle` command, this command can be given anywhere in the frame, since the frame title is actually typeset only when everything else has already been typeset.

Example:

```
\frame{
  \frametitle<presentation>{Frame Title Should Be in Uppercase.}
  \framesubtitle{Subtitles can be in lowercase if they are full sentences.}

  Frame contents.
}
```

ARTICLE By default, the subtitle is not shown in any way in `article` mode.

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." You can override this for individual frames using the `c` or `b` option.

`\documentclass[slidescentered]{beamer}`

Place text of slides at the (vertical) center of the slides. This is the default. You can override this for individual frames using the `t` or `b` option.

4.2.3 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. Another useful specification is `<beamer>`, which causes the frame to be shown normally in `beamer` mode, but to be suppressed in all other versions.

4.2.4 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 verbatim text using the `containsverbatim` commans:

```
\frame[containsverbatim]
{
  \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}

}
```

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[`⟨*options*⟩`]{`⟨*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.

The following ⟨*options*⟩ may be given:

- `colored` declares that the verbatim text will have its "own" colors. Normally, the verbatim text is typeset using the current color, which allows you to use commands like `\alert` to make verbatim text red on certain slides. However, if the verbatim text has, say, a special background color or different parts of it a colored differently (the `lstlisting` environment does this), then you do *not* want the verbatim text to inherit its color from the "outside." In this case, you should give the `colored` option.

Example:

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

```
\defverbatim[colored]\algorithmB{
\begin{lstlisting}[language={C++},backgroundcolor=\color{yellow}]
int main (void)
{
  cout << "Hello world." << endl;
  return 0;
}
\end{lstlisting}
}

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

\frame
{
  Same algorithm typeset using the lstlisting environment:
  \algorithmB
}
```

4.3 Creating Overlays

4.3.1 The Pause Commands

The `pause` command offers an easy, but not very flexible way of creating frames that are uncovered piecewise. If you say `\pause` somewhere in a frame, only the text on the frame 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. You can also use `\pause` inside environments; its effect will last after the environment. However, taking this to extremes and use `\pause` deeply within a nested environment may not have the desired result.

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.

The effect of `\pause` lasts till the next `\pause`, `\onslide`, or the end of the frame.

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

  Shown from fourth slide on.

  \begin{itemize}
  \unpause
  \item
    Shown from first slide on.
  \pause
  \item
    Shown from fifth slide on.
  \end{itemize}
}
```

`\pause[⟨number⟩]`

This command causes the text following it to be shown only from the next slide on, or, if the optional ⟨*number*⟩ is given, from the slide with the number ⟨*number*⟩. If the optional ⟨*number*⟩ is given, the counter `beamerpauses` is set to this number. This command uses the `\onslide` command, internally. This command does *not* work inside `amsmath` environments like `align`, since these do really wicked things.

Example:

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

ARTICLE This command is ignored.

LYX Use the "Pause" style with an empty line to insert a pause.

To "unpause" some text, that is, to temporarily suspend pausing, use the command `\onslide`, see below.

4.3.2 Commands with Overlay Specifications

A much more powerful and flexible way of specifying overlays uses overlay specifications, see Section 4.1 for an introduction to this concept. In this subsection, the basic commands that take overlay specifications are described.

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 should directly follow the command as in the following example (but there are exceptions to this rule):

```
\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:

`\onslide<⟨overlay specification⟩>`

All text following this command will only be shown (uncovered) on the specified slides. On non-specified slides, the text still occupies space. If no slides are specified, the following text is always shown. You need not call this command in the same TeX group, its effect transcends block groups. However, this command has a *different* effect inside an `overprint` environment, see the description of `overprint`.

Example:

```
\frame
{
  Shown on first slide.
  \onslide<2-3>
  Shown on second and third slide.
  \begin{itemize}
  \item
    Still shown on the second and third slide.
  \onslide<4->
  \item
    Shown from slide 4 on.
  \end{itemize}
  Shown from slide 4 on.
  \onslide
  Shown on all slides.
}
```

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

If either ⟨*overlay specification*⟩ is present (though only one may be 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.}`

Since the overlay specification may also be given after the text, you can often use `\only` to make other commands overlay-specification-aware in a simple manner:

Example:

```
\newcommand{\myblue}{\only{\color{blue}}}
\frame
{
  \myblue<2> This text is blue only on slide 2.
}
```

`\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 6.3.

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

ARTICLE This command has the same effect as `\only`.

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

This command does almost the same as `\uncover`. The only difference is that if the text is not shown, it is never shown in a transparent way, but rather it is not shown at all. Thus for this command the transparency settings have no effect.

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

ARTICLE This command has the same effect as `\only`.

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

This command is the opposite of `\visible`.

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

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

Only one ⟨*overlay specification*⟩ may be given. 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.}`

Once more, giving the overlay specification at the end is useful when the command is used inside other commands.

Example: Here is the definition of `\uncover`:

```
\newcommand{\uncover}{\alt{\@firstofone}{\makeinvisible}}
```

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

BEAMER Only one ⟨*alert specification*⟩ may be given. The effect of ⟨*alert specification*⟩ is described in Section 4.1.3.

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

ARTICLE The ⟨*action specification*⟩ is currently completely ignored.

LYX The ⟨*action specification*⟩ must be given in TEX-mode and it must be given at the very start of the item.

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 and *except* if you wish to use that lable as a hyperjump target. Then you need to specifiy a slide.

Labels can be used as target of hyperjumps. A convenient way of labelling a frame is to use the `label=`⟨*name*⟩ option of the `\frame` command. However, this will cause the whole frame to be kept in memory till the end of the compilation, which may pose a problem.

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}}
}
```

4.3.3 Environments with Overlay Specifications

Environments can also be equipped with overlay specifications. For most of the predefined environments, see Section 5.3.3, adding an overlay specification 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.

For each of the basic commands `\only`, `\alt`, `\visible`, `\uncover`, and `\invisible` there exists "environment versions" `onlyenv`, `altenv`, `visibleenv`, `uncoverenv`, and `invisibleenv`. Except for `altenv` and `onlyenv`, these environments do the same as the commands.

\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. The difference to `\only` is, that the text is actually typeset inside a box that is then thrown away, whereas `\only` immediately throws away its contents. If the text is not "typesettable," the `onlyenv` may produce an error where `\only` would not.

Example:

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

\begin{altenv}<⟨*overlay specification*⟩>{⟨*begin text*⟩}{⟨*end text*⟩}{⟨*alternate begin text*⟩}{⟨*alternate end text*⟩}<⟨*overlay specification*⟩>
⟨*environment contents*⟩
\end{altenv}

Only one ⟨*overlay specification*⟩ may be given. On the specified slides, ⟨*begin text*⟩ will be inserted at the beginning of the environment and ⟨*end text*⟩ will be inserted at the end. On all other slides, ⟨*alternate begin text*⟩ and ⟨*alternate end text*⟩ will be used.

Example:

```
\frame
{
  This
  \begin{altenv}<2>{(}{)}{[}{]}
    word
  \end{uncoverenv}
  is in round brackets on slide 2 and in square brackets on slide 1.
}
```

4.3.4 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}
```

LYX Use the style “OverlayArea” to insert an overlay area.

`\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}
```

LYX Use the style “Overprint” to insert an `overprint` environment. You have to use TEX-mode to insert the `\onslide` commands.

4.4 Making Commands and Environments Overlay-Specification-Aware

This subsection explains how to define new commands that are 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.

BEAMER extends the syntax of LATEX's standard command `\newcommand`:

`\newcommand<>{⟨command name⟩}[⟨argument number⟩][⟨default optional value⟩]{⟨text⟩}`

Declares the new command named ⟨*command name*⟩. The ⟨*text*⟩ should contain the body of this command and it may contain occurences of parameters like `#`⟨*number*⟩. Here ⟨*number*⟩ may be between 1 and ⟨*argument number*⟩ + 1. The additionally allowed argument is the overlay specification.

When ⟨*command name*⟩ is used, it will scan as many as ⟨*argument number*⟩ arguments. While scanning them, it will look for an overlay specification, which may be given between any two arguments, before the first argument, or after the last argument. If it finds an overlay specification like `<3>`, it will call ⟨*text*⟩ with arguments 1 to ⟨*argument number*⟩ set to the normal arguments and the argument number ⟨*argument number*⟩ + 1 set to `<3>` (including the pointed brackets). If no overlay specification is found, the extra argument is empty.

If the ⟨*default optional value*⟩ is provided, the first argument of ⟨*command name*⟩ is optional. If no optional argument is specified in square brackets, the ⟨*default optional value*⟩ is used.

Example: The following command will typeset its argument in red on the specified slides:

```
\newcommand<>{\makered}[1]{{\color#2{red}#1}}
```

Example: Here is BEAMER's definition of `\emph`:

```
\newcommand<>{\emph}[1]{{\only#2{\itshape}#1}}
```

Example: Here is BEAMER's definition of `\transdissolve` (the command `\beamer@dotrans` mainly passes its argument to `hyperref`):

```
\newcommand<>{\transdissolve}[1][]{\only#2{\beamer@dotrans[#1]{Dissolve}}}
```

`\renewcommand<>`{⟨*existing command name*⟩}[⟨*argument number*⟩][⟨*default optional value*⟩]{⟨*text*⟩}

Redeclares a command that already exists in the same way as `\newcommand<>`. Inside ⟨*text*⟩, you can still access to original definitions using the command `\beameroriginal`, see the example.

Example: This command is used in BEAMER to make `\hyperlink` overlay-specification-aware:

```
\renewcommand<>{\hyperlink}[2]{\only#3{\beameroriginal{\hyperlink}{#1}{#2}}}
```

`\newenvironment<>`{⟨*environment name*⟩}[⟨*argument number*⟩][⟨*default optional value*⟩]
{⟨*begin text*⟩}{⟨*end text*⟩}

Declares a new environment that is overlay-specification-aware. If this environment is encountered, the same algorithm as for `\newcommand<>` is used to parse the arguments and the overlay specification.

Note that, as always, the ⟨*end text*⟩ may not contain any arguments like `#1`. In particular, you do not have access to the overlay specification. In this case, it is usually a good idea to use `altenv` environment in the ⟨*begin text*⟩.

Example: Declare your own action block:

```
\newenvironment<>{myboldblock}[1]{%
  \begin{actionenv}#2%
    \textbf{#1}
    \par}
  {\par%
  \end{actionenv}}

\frame
{
  \begin{myboldblock}<2>
    This theorem is shown only on the second slide.
  \end{myboldblock}
}
```

Example: Text in the following environment is normally bold and italic on non-specified slides:

```
\newenvironment<>{boldornormal}
  {\begin{altenv}#1
    {\begin{bfseries}}{\end{bfseries}}
    {}{}}
  {\end{altenv}}
```

Incidentally, since `altenv` also accepts its argument at the end, the same effect could have been achieved using just

```
\newenvironment{boldornormal}
  {\begin{altenv}
    {\begin{bfseries}}{\end{bfseries}}
    {}{}}
  {\end{altenv}}
```

`\renewenvironment<>`{⟨*existing environment name*⟩}[⟨*argument number*⟩][⟨*default optional value*⟩]
{⟨*begin text*⟩}{⟨*end text*⟩}

Redefines an existing environment. The original environment is still available under the name `original`⟨*existing environment name*⟩.

Example:

```
\renewenvironment<>{verse}
{\begin{actionenv}#1\begin{originalverse}}
{\end{originalverse}\end{actionenv}}
```

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 Creating the Static Global Structure

This section lists the commands that are used for structuring a presentation "globally" using commands like `\section` or `\part`. These commands are use to create a *static* structure, meaning that the resulting presenation is normally presented one slide after the other in the order the slides occur. Section 5.2 explains which commands can be used to create the *interactive* structure. For the interactive structure, you must interact with the presenation program, typically by clicking on hyperlinks, to advance the presentation.

5.1.1 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}`

LYX If you use the "Title" style in your presentation, a title page is automatically inserted.

For compatibility with other classes, in `article` mode the following command is also provided:

`\maketitle`

BEAMER If used inside a frame, it has the same effect as `\titlepage`. If used outside a frame, it has the same effect as `\frame{\titlepage}`; in other words, a frame is added if necessary.

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 headlines and footlines. 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}
```

ARTICLE The short form is ignored in `article` mode.

`\subtitle[⟨short subtitle⟩]{⟨subtitle⟩}`

The ⟨*subshort tile*⟩ is not used by default, but is available via the insert `\insertshortsubtitle`. The subtitle is shown below the title in a smaller font.

Example:

```
\title{The Beamer Class}
\subtitle{An easily paced introduction with many examples.}
```

ARTICLE This command causes the subtitle to be appended to the title with a linebreak and a `\normalsize` command issued before it. This may or may not be what you would like to happen.

`\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}}`

ARTICLE The short form is ignored in `article` mode.

`\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}
```

ARTICLE The short form is ignored in `article` mode. The long form is also ignored, except if the document class (like `llncs`) defines it.

`\date[⟨short date⟩]{⟨date⟩}`

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

ARTICLE The short form is ignored in `article` mode.

`\titlegraphic{⟨text⟩}`

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

Example: `\titlegraphic{\pgfuseimage{titlegraphic}}`

ARTICLE The command is ignored in `article` mode.

`\subject{⟨text⟩}`

Enters the ⟨*text*⟩ as the subject text in the PDF document info. It currently has no other effect.

By default, the `\title` and `\author` commands will also insert their arguments into a resulting PDF-file in the document information fields. This may cause problems if you use complicated things like boxes as arguments to these commands. In this case, you might wish to switch off the automatic generation of these entries using the following class option:

`\documentclass[usepdftitle=false]{beamer}`

Suppresses the automatic generation of title and author entries in the PDF document information.

5.1.2 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 (actually, I do not really know why; comments would be appreciated).

`\section<`⟨*mode specification*⟩`>[`⟨*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. If a ⟨*mode specification*⟩ is given, the command only has an effect for the specified modes.

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

ARTICLE The ⟨*mode specification*⟩ allows you to provide an alternate section command in `article` mode. This is necessary for example if the ⟨*short section name*⟩ is unsuitable for the table of contents:

Example:

```
\section<presentation>[Results]{Results on the Main Problem}
\section<article>{Results on the Main Problem}
```

`\section<`⟨*mode specification*⟩`>*{`⟨*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}`

Example: `\section<beamer>*{Outline}`

`\subsection<`⟨*mode specification*⟩`>[`⟨*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<`⟨*mode specification*⟩`>*{`⟨*subsection name*⟩`}`

Starts a subsection without an entry in the table of contents. No heading is created, but the ⟨*subsection name*⟩ is shown in the navigation bar, *except* if ⟨*subsection name*⟩ is empty. In this case, neither a table of contents entry nor a navigation bar entry is created, *but* any frames in this "empty" subsection are shown in the navigation bar.

Example:

```
\section{Summary}

  \frame{This frame is not shown in the navigation bar}

  \subsection*{}

  \frame{This frame is shown in the navigation bar, but no subsection
    entry is shown.}

  \subsection*{A subsection}

  \frame{Normal frame, shown in navigation bar. The subsection name is
    also shown in the navigation bar, but not in the table of contents.}
```

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<beamer>
  {
    \frametitle{Outline}
    \tableofcontents[current]
  }
}
```

ARTICLE This command has no effect in `article` mode.

LYX You have to insert this command using a TEX-mode text.

\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<beamer>
  {
    \frametitle{Outline}
    \tableofcontents[current,currentsubsection]
  }
}
```

5.1.3 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<⟨*mode specification*⟩>[⟨*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 9.4.3.

Example: `\frame{\partpage}`

`\AtBeginPart{`⟨*text*⟩`}`

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

Example:

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

5.1.4 Splitting a Course Into Lectures

When using BEAMER with the `article` mode, you may wish to have the lecture notes of a whole course reside in one file. In this case, only a few frames are actually part of any particular lecture.

The `\lecture` command makes it easy to select only a certain set of frames from a file to be presented. This command takes (among other things) a label name. If you say `\includeonlylecture` with this label name, then only the frames following the `\lecture` command are shown. The frames following other `\lecture` commands are suppressed.

By default, the `\lecture` command has no other effect. It does not create any frames or introduce entries in the table of contents. However, you can use `\AtBeginLecture` to have BEAMER insert, say, a title page at the beginning of (each) lecture.

`\lecture[`⟨*short lecture name*⟩`]{`⟨*lecture name*⟩`}{`⟨*lecture label*⟩`}`

Starts a lecture. The ⟨*lecture name*⟩ will be available via the `\insertlecture` command. The ⟨*short lecture name*⟩ is available via the `\insertshortlecture` command.

Example:

```
\begin{document}
\lecture{Vector Spaces}{week 1}

\section{Introduction}
...
```

```
\section{Summary}

\lecture{Scalar Products}{week 2}

\section{Introduction}
...
\section{Summary}

\end{document}
```

ARTICLE This command has no effect in `article` mode.

`\includeonlylecture`⟨*lecture label*⟩

Causes all `\frame`, `\section`, `\subsection`, and `\part` commands following a `\lecture` command to be suppressed, except if the lecture's label matches the ⟨*lecture label*⟩. Frames before any `\lecture` commands are always included. This command should be given in the preamble.

Example: `\includeonlylecture{week 1}`

ARTICLE This command has no effect in `article` mode.

`\AtBeginLecture{`⟨*text*⟩`}`

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

Example:

```
\AtBeginLecture{\frame{\Large Today's Lecture: \insertlecture}}
```

ARTICLE This command has no effect in `article` mode.

5.1.5 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 9.4.6.

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.

ARTICLE The options are ignored in `article` mode.

LYX You can give options to the `\tableofcontents` command by inserting a TeX-mode text with the options in square brackets directly after the table of contents.

5.1.6 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 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 recommend 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 9.4.7) 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) or use the `allowframebreaks` option. 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 behavior, see also Section 9.4.7.

5.1.7 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 starts 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<⟨*mode specification*⟩>

Starts the appendix in the specified modes. 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 Creating the Interactive Global Structure

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 present a complicated picture and you have to "zoom out" different parts to explain details.
- 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 5.2.4.

Concerning the first three 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 `\againframe` command to "continue" frames that you previously started somewhere, but where certain details have been suppressed. You can use the `\againframe` command at a much later point, for example only in the appendix to show additional slides there.

- You can use the `\framezoom` command to create links to zoomed out parts of a complicated slide.

5.2.1 Adding Hyperlinks and Buttons

To create 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` or (easier) the command `\label`. 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}{}
}
```

ARTICLE You must say `\usepackage{hyperref}` in your preamble to use this command in `article` mode.

The `\label` command creates a hypertarget as a side-effect and the `label=`⟨*name*⟩ option of the `\frame` command creates a label named ⟨*name*⟩`<`⟨*slide number*⟩`>` for each slide of the frame as a side-effect. Thus the above example could be written more easily as:

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

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

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 9.4.11.

`\beamerbutton{`⟨*button text*⟩`}`

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

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

ARTICLE This command (and the following) just insert their argument in `article` mode.

\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<1>[label=mytheorem]
{
  \begin{theorem}
    ...
  \end{theorem}

  \begin{overprint}
  \onslide<1>
    \hfill\hyperlink{mytheorem<2>}{\beamergotobutton{Go to proof details}}
  \onslide<2>
    \begin{proof}
      ...
    \end{proof}
    \hfill\hyperlink{mytheorem<1>}{\beamerreturnbutton{Return}}
  \end{overprint}
}
\appendix
\againframe<2>{mytheorem}
```

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<⟨*overlay specification*⟩>{⟨*target name*⟩}{⟨*link text*⟩}<⟨*overlay specification*⟩>

Only one ⟨*overlay specification*⟩ may be given. 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*⟩. If an ⟨*overlay specification*⟩ is present, the hyperlink (including the ⟨*link text*⟩) is completely suppressed on the non-specified slides.

The following commands have a predefined target; otherwise they behave exactly like `\hyperlink`. In particular, they all also accept an overlay specification and they also accept it at the end, rather than at the beginning.

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

Clicking the text jumps one slide back.

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

Clicking the text jumps one slide forward.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`\hyperlinkdocumentend<`⟨*overlay specification*⟩`>{`⟨*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.

5.2.2 Repeating a Frame at a Later Point

Sometimes you may wish some slides of a frame to be shown in your main talk, but wish some "supplementary" slides of the frame to be shown only in the the appendix. In this case, the `\againframe` commands is useful.

\againframe<⟨*overlay specification*⟩>[<⟨*default overlay specification*⟩>][⟨*options*⟩]{⟨*name*⟩}

BEAMER Resumes a frame that was previously created using `\frame` with the option `label=`⟨*name*⟩. You must have used this option, just placing a label inside a frame "by hand" is not enough. 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*⟩, ⟨*default overlay specification*⟩ (if present), 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.
}

\againframe<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 fourth 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}

  \begin{overprint}
  \onslide<1>
    \hyperlink{Cantor<2>}{\beamergotobutton{Proof details}}

  \onslide<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}}
  \end{overprint}
}

...
\appendix

\againframe<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.

ARTICLE This command is ignored in `article` mode.

LYX Use the style "AgainFrame" to insert an `\againframe` command. The ⟨*label name*⟩ is the text on following the style name and is *not* put in TEX-mode. However, an overlay specification must be given in TEX-mode and it must precede the label name.

5.2.3 Adding Anticipated Zooming

When you present a graphic during a presentation, you should explain everything that is shown in the graphic. When graphics are imported from other context, they typically contain far too much information and should be radically simplified.

Sometimes however, the complexity of a graphic is intensional and you are willing to spend much time explaining the graphic in great detail. In this case, you will often run into the problem that fine details of the graphic are hard to discern for the audience. One way to solve this problem is to use the command `\framezoom`. This command allows you to specify that clicking on a certain area of a frame should zoom out this area. You can then explain the details. Clicking on the zoomed out picture will take you back to the original one.

`\framezoom`<⟨*button overlay specification*⟩><⟨*zoomed overlay specification*⟩>[⟨*options*⟩]
(⟨*upper left x*⟩,⟨*upper left y*⟩)(⟨*zoom area width*⟩,⟨*zoom area depth*⟩)

This command should be given somewhere at the beginning of a frame. When given, two different things will happen, depending on whether the ⟨*button overlay specification*⟩ applies to the current slide of the frame or whether the ⟨*zoomed overlay specification*⟩ applies. These overlay specifications should not overlap.

If the ⟨*button overlay specification*⟩ applies, a clickable are is created inside the frame. The size of this area is given by ⟨*zoom area width*⟩ and ⟨*zoom area depth*⟩, which are two normal TEX dimensions (like `1cm` or `20pt`). The upper left corner of this area is given by ⟨*upper left x*⟩ and ⟨*upper left y*⟩, which are also TEX dimensions. They are measures *relative to the place where the first normal text of a frame would go*. Thus, the location `(0pt,0pt)` is at the beginning of the normal text (which excludes the headline and also the frame title).

By default, the button is clickable, but it will not be indicated in any special way. You can draw a border around the button by using the following ⟨*option*⟩:

- `border=`⟨*width in pixels*⟩ will draw a border around the specified button area. The default width is 1 pixel. The color of this button is the `linkbordercolor` of `hyperref`. BEAMER sets this color to a 50% gray by default. To change this, you can use the command `\hypersetup{linkbordercolor={`⟨*red*⟩ ⟨*green*⟩ ⟨*blue*⟩`}}`, where ⟨*red*⟩, ⟨*green*⟩, and ⟨*blue*⟩ are values between 0 and 1.

When you press the button created in this way, the viewer application will hyperjump to the first of the frames specified by the ⟨*zoomed overlay specification*⟩. For the slides to which this overlay specification applies, the following happens:

The exact same area as the one specified before is "zoomed out" to fill the whole normal text area of the frame. Everything else, including the sidebars, the headlines and footlines, and even the frame title retain their normal size. The zooming is performed in such a way that the whole specified area is completely shown. The aspect ratio is kept correct and the zoomed area will possibly show more than just the specified area if the aspect ratio of this area and the aspect ratio of the available text area do not agree.

Behind the whole text area (which contains the zoomed area) a big invisible "Back" button is put. Thus clicking anywhere on the text area will jump back to the original (unzoomed) picture.

You can specify several zoom areas for a single frame. In this case, you should specify different ⟨*zoomed overlay specification*⟩, but you can specify the same ⟨*button overlay specification*⟩. You cannot nest zoomings in the sense that you cannot have a zoom button on a slide that is in some ⟨*zoomed overlay specification*⟩. However, you can have overlapping and even nested ⟨*button overlay specification*⟩. When clicking on an area that belongs to several buttons, the one given last will "win" (it should hence be the smallest one).

If you do not wish to have the frame title shown on a zoomed slide, you can add an overlay specification to the `\frametitle` command that simply suppresses the title for the slide. Also, by using the `plain` option, you can have the zoomed slide fill the whole page.

Example: A simple case

```
\frame
{
  \frametitle{A Complicated Picture}

  \framezoom<1><2>(0cm,0cm)(2cm,1.5cm)
  \framezoom<1><3>(1cm,3cm)(2cm,1.5cm)
  \framezoom<1><4>(3cm,2cm)(3cm,2cm)

  \pgfimage[height=8cm]{complicatedimagefilename}
}
```

Example: A more complicate case in which the zoomed parts completely fill the frames.

```
\frame<1>[label=zooms]
{
  \frametitle<1>{A Complicated Picture}

  \framezoom<1><2>[border](0cm,0cm)(2cm,1.5cm)
  \framezoom<1><3>[border](1cm,3cm)(2cm,1.5cm)
  \framezoom<1><4>[border](3cm,2cm)(3cm,2cm)

  \pgfimage[height=8cm]{complicatedimagefilename}
}
\againframe<2->[plain]{zooms}
```

5.2.4 Using the Navigation Bars

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

5.2.5 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 symbol 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 symbol 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 9.4.13.

5.3 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.

5.3.1 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}[<`⟨*default overlay specification*⟩`>]`
⟨*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 appearance of the items can be changed using templates, see Section 9.4.

If the optional parameter ⟨*default overlay specification*⟩ is given, in every occurrence of an `\item` command that does not have an overlay specification attached to it, the ⟨*default overlay specification*⟩ is used. By setting this specification to be an incremental overlay specification, see Section 4.1.4, you can implement, for example, a step-wise uncovering of the items. The ⟨*default overlay specification*⟩ is inherited by subenvironments. Naturally, in a subenvironment you can reset it locally by setting it to `<1->`.

Example:

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

Example:

```
\begin{itemize}[<+->]
\item This is shown from the first slide on.
\item This is shown from the second slide on.
\item This is shown from the third slide on.
\item<1-> This is shown from the first slide on.
\item This is shown from the fourth slide on.
\end{itemize}
```

Example:

```
\begin{itemize}[<+-| alert@+>]
\item This is shown from the first slide on and alerted on the first slide.
\item This is shown from the second slide on and alerted on the second slide.
\item This is shown from the third slide on and alerted on the third slide.
\end{itemize}
```

Example:

```
\newenvironment{mystepwiseitemize}{\begin{itemize}[<+-| alert@+>]}{\end{itemize}}
```

LYX Unfortunately, currently you cannot specify optional arguments with the `itemize` environment. You can, however, use the command `\beamerdefaultoverlayspecification` before the environment to get the desired effect.

`\begin{enumerate}[<`⟨*default overlay specification*⟩`>] [`⟨*mini template*⟩`]`
⟨*environment contents*⟩
`\end{enumerate}`

Used to display a list of items that are ordered. Inside the environment, use an `\item` command for each topic. By default, before each item increasing Arabic numbers followed by a dot are printed (as in “1.” and “2.”). This can be changed by specifying a different template, see Section 9.4.16.

The first optional argument ⟨*default overlay specification*⟩ has exactly the same effect as for the `itemize` environment. It is “detected” by the opening `<`-sign in the ⟨*default overlay specification*⟩. Thus, if there is only one optional argument and if this argument does not start with `<`, then it is considered to be a ⟨*mini template*⟩.

The syntax of the ⟨*mini template*⟩ is the same as the syntax of mini templates in the `enumerate` package (you do not need to include the `enumerate` package, this is done automatically). Roughly spoken, the text of the ⟨*mini template*⟩ is printed before each item, but any occurrence of a `1` in the mini template is replaced by the current item number, an occurrence of the letter `A` is replaced by the ith letter of the alphabet (in uppercase) for the ith item, and the letters `a`, `i`, and `I` are replaced by the corresponding lowercase letters, lowercase Roman letters, and uppercase Roman letters, respectively. So the mini template `(i)` would yield the items (i), (ii), (iii), (iv), and so on. The mini template `A.)` would yield the items A.), B.), C.), D.) and so on. For more details on the possible mini templates, see the documentation of the `enumerate` package. Note that there is also a template that governs the appearance of the mini template (for example, to change its color), see Section 9.4.16.

Example:

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

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

\begin{enumerate}[(i)]
\item First Roman point.
\item Second Roman point.
\end{enumerate}

\begin{enumerate}[<+->][(i)]
\item First Roman point.
\item Second Roman point, uncovered on second slide.
\end{enumerate}
```

ARTICLE To use the ⟨*mini template*⟩, you have to include the package `enumerate`.

LYX The same constraints as for `itemize` apply.

`\begin{description}[<`⟨*default overlay specification*⟩`>][`⟨*long text*⟩`]`
⟨*environment contents*⟩
`\end{description}`

Like `itemize`, but used to display a 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.

As for `enumerate`, the ⟨*default overlay specification*⟩ is detected by an opening `<`. The effect is the same as for `enumerate` and `itemize`.

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}
```

Example: The following has the same effect as the previous example:

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

LYX Since you cannot specify the optional argument in LYX, if you wish to specify the width, you must use the command `\usedescriptionitemofwidthas`, which you must insert in TEX-mode shortly before the environment.

`\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}
```

5.3.2 Hilighting

The BEAMER class predefines commands and environments for hilighting text. 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}.`

ARTICLE Alerted text is typeset as emphasized text. This can be changed by specifying another template.

LYX You need to use TEX-mode to insert this command.

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

Environment version of the `\alert` command.

`\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.}`

ARTICLE Structure text is typeset as bold text. This can be changed by specifying another template.

LYX You need to use TEX-mode to insert this command.

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

Environment version of the `\structure` command.

5.3.3 Block Environments

The BEAMER class predefines an environment for typesetting a "block" of text that has a heading. The appearence of the block is governed by a template.

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

Only one ⟨*action specification*⟩ may be given. Inserts a block, like a definition or a theorem, with the title ⟨*block title*⟩. If the ⟨*action specification*⟩ is present, the given actions are taken on the specified slides, see Section 4.1.3. In the example, the definition is shown only from slide 3 onward.

Example:

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

ARTICLE The block name is typeset in bold.

LYX The argument of the block must (currently) be given in TEX-mode. More precisely, there must be an opening brace in TEX-mode and a closing brace in TEX-mode around it. The text in between can also be typeset using LYX. I hope to get rid of this some day.

`\begin{alertblock}<`⟨*action specification*⟩`>{`⟨*block title*⟩`}<`⟨*action specification*⟩`>`
⟨*environment contents*⟩

\end{alertblock}

Inserts a block whose title is hilighted. Behaves like the `block` environment otherwise.

Example:

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

ARTICLE The block name is typeset in bold and is emphasized.

LYX Same applies as for `block`.

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

Inserts a block that is supposed to be an example. Behaves like the `block` environment otherwise.

Example: In the following example, the block is completely suppressed on the first slide (it does not even occupy any space).

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

ARTICLE The block name is typeset in italics.

LYX Same applies as for `block`.

LYX Overlay specifications must be given right at the beginning of the environments and in TEX-mode.

5.3.4 Theorem Environments

The BEAMER class predefines several environments, like `theorem` or `definition` or `proof`, that you can use to typeset things like, well, theorems, definitions, or proofs. The complete list is the following: `theorem`, `corollary`, `definition`, `definitions`, `fact`, `example`, and `examples`. The following German block environments are also predefined: `Problem`, `Loesung`, `Definition`, `Satz`, `Beweis`, `Folgerung`, `Lemma`, `Fakt`, `Beispiel`, and `Beispiele`.

Here is a typical example on how to use them:

```
\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->[Natural Numbers]
    The set of natural numbers is infinite.
  \end{example}
}
```

In the following, only the English versions are discussed. The German ones behave analogously.

\begin{theorem}<⟨*action specification*⟩>[⟨*additional text*⟩]<⟨*action specification*⟩>
⟨*environment contents*⟩

\end{theorem}

Inserts a theorem. Only one ⟨*action specification*⟩ may be given. If present, the ⟨*additional text*⟩ is shown behind the word "Theorem" in rounded brackets (although this can be changed by the template).

The appearance of the theorem is governed by templates, see Section 9.4.19 for details on how to change these. Every theorem is put into a `block` environment, thus the templates for blocks also apply.

The theorem style (a concept from `amsthm`) used for this environment is `plain`. In this style, the body of a theorem should be typeset in italics. The head of the theorem should be typeset in a bold font, but this is usually overruled by the templates.

If the option `envcountsect` is given either as class option in one of the `presentation` modes or as an option to the package `beamerbasearticle` in `article` mode, then the numbering of the theorems is local to each section with the section number prefixing the theorem number; otherwise they are numbered consecutively throughout the presentation or article. I recommend using this option in `article` mode.

By default, no theorem numbers are shown in the `presentation` modes.

Example:

```
\begin{theorem}[Kummer, 1992]
  If $\#^_A^n$ is $n$-enumerable, then $A$ is recursive.
\end{theorem}

\begin{theorem}<2->[Tantau, 2002]
  If $\#_A^2$ is $2$-fa-enumerable, then $A$ is regular.
\end{theorem}
```

LYX If present, the optional argument and the action specification must be given in TEX-mode at the beginning of the environment.

The environments `corollary`, `fact`, and `lemma` behave exactly the same way.

\documentclass[envcountsect]{beamer}

Causes theorems, definitions, and so on to be numbered locally to each section. Thus, the first theorem of the second section would be Theorem 2.1 (assuming that there are no definitions, lemmas, or corollaries earlier in the section).

\begin{defintion}<⟨*action specification*⟩>[⟨*additional text*⟩]<⟨*action specification*⟩>
⟨*environment contents*⟩
\end{defintion}

Behaves like the `theorem` environment, except that the theorem style `definition` is used. In this style, the body of a theorem is typeset in an upright font.

The environment `definitions` behaves exactly the same way.

\begin{example}<⟨*action specification*⟩>[⟨*additional text*⟩]<⟨*action specification*⟩>
⟨*environment contents*⟩
\end{example}

Behaves like the `theorem` environment, except that the theorem style `example` is used. A side-effect of using this theorem style is that the ⟨*environment contents*⟩ is put in an `exampleblock` instead of a `block`.

The environment `examples` behaves exactly the same way.

Some remarks on numbered theorems:

BEAMER The default template for typesetting theorems suppresses the theorem number, even if this number is "available" for typesetting (which it is by default in all predefined environments; but if you define your own environment using `\newtheorem*` no number will be available). I would like to discourage using numbered theorems in presentations. The audience has no chance of remembering these numbers. *Never* say things like "now, by Theorem 2.5 that I showed you earlier, we have ..." It would be much better to refer to, say, Kummer's Theorem instead of Theorem 2.5. If Theorem 2.5 is some obscure theorem that does not have its own name (like Kummer's Theorem or Main Theorem or Second Main Theorem or Key Lemma), then the audience will have forgotten about it anyway by the time you refer to it again.

In my opinion, the only situtation in which numbered theorems make sense in a presentation is in a lecture, in which the students can read lecture notes in parallel to the lecture where the theorems are numbered in exactly the same way.

ARTICLE In `article` mode, theorems are automatically numbered. By specifying the class option `envcountsect`, theorems will be numbered locally to each section, which is usually a good idea, except for very short articles.

The predefined environments number everything consecutively. Thus if there are one theorem, one lemma, and one definition, you would have Theorem 1, Lemma 2, and Definition 3. Some people prefer all three to be numbered 1. I would *strongly* like to discourage this. The problem is that this makes it virtually impossible to find anything since Theorem 2 might come after Definition 10 or the other way round. Papers and, worse, books that have a Theorem 1 and a Definition 1 are a pain. Do not inflict pain on other people.

`\begin{proof}`<⟨*action specification*⟩>[⟨*proof name*⟩]<⟨*action specification*⟩>
⟨*environment contents*⟩
`\end{proof}`

Typesets a proof. If the optional ⟨*proof name*⟩ is given, it completely replaces the word "Proof." This is different from normal theorems, where the optional argument is shown in brackets.

At the end of the theorem, a `\qed` symbol is shown, except if you say `\qedhere` earlier in the proof (this is exactly as in `amsthm`). The default `\qed` symbol is an open rectangle. To completely suppress the symbol, write `\def\qedsymbol{}` in your preamble. To get an closed rectangle, say

```
\useqedsymboltemplate{\color{beamerstructure}\vrule width1.5ex height1.5ex depth0pt}
```

If you use `babel` and a different language, the text "Proof" is replaced by whatever is appropriate in the selected language.

Example:

```
\begin{proof}<2->[Sketch of proof]
  Suppose ...
\end{proof}
```

You can define new environments using the following command:

`\newtheorem*{`⟨*environment name*⟩`}[`⟨*numbered same as*⟩`]{`⟨*head text*⟩`}[`⟨*number within*⟩`]`

This command is used exactly the same way as in the `amsthm` package (as a matter of fact, it is the command from that package), see its documentation. The only difference is that environments declared using this command are overlay-specification-aware in BEAMER and that, when typeset, are typeset according to BEAMER's templates.

ARTICLE Environments declared using this command are also overlay-specification-aware in `article` mode.

Example: `\newtheorem{observation}[theorem]{Observation}`

You can also use `amsthm`'s command `\newtheoremstyle` to define new theorem styles. Note that the default template for theorems will ignore any head font setting, but will honor the body font setting.

If you wish to define the environments like `theorem` differently (for example, have it numbered within each subsection), you can use the following class option to disable the definition of the predefined environments:

`\documentclass[notheorems]{beamer}`

Switches off the definition of default blocks like `theorem`, but still loads `amsthm` and makes theorems overlay-specificiation-aware.

The option is also available as a package option for `beamerbasearticle` and has the same effect.

ARTICLE In the `article` version, the package `amsthm` sometimes clashes with the document class. In this case you can use the following option, which is once more available as a class option for BEAMER and as a package option for `beamerbasearticle`, to switch off the loading of `amsthm` altogether.

`\documentclass[noamsthm]{beamer}`

Does not load `amsthm` and also not `amsmath`. Environments like `theorem` or `proof` will not be available.

5.3.5 Framed Text

In order to draw a frame (a rectangle) around some text, you can use LaTeXs standard command `\fbox` and also `\frame` (inside a BEAMER frame, the `\frame` command changes its meaning to the normal LaTeX `\frame` command). 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}
```

ARTICLE This environment is not available in `article` mode.

`\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}`

ARTICLE This command is not available in `article` mode.

5.3.6 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 or use the `allowframebreaks` option.
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 9.4.15.

5.3.7 Splitting a Frame into Multiple Columns

The BEAMER class offers several commands and environments for splitting (perhaps only part of) a frame into multiple columns. These commands have nothing to do with LaTeX's commands for creating columns. 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 either place several `column` environments, each of which creates a new column, or use the `\column` command to create new columns.

`\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}
```

ARTICLE This environment is ignored in `article` mode.

LYX Use “Columns” or “ColumnsTopAligned” to create a `columns` environment. To pass options, insert them in TeX-mode right at the beginning of the environment in square brackets.

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}
```

ARTICLE This command is ignored in `article` mode.

LYX The "Column" styles insert the command version, see below.

`\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 occurrence 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}
```

ARTICLE This command is ignored in `article` mode.

LYX In a "Column" style, the width of the column must be given as normal text, not in TEX-mode.

5.3.8 Positioning Text and Graphics Absolutely

Normally, BEAMER uses TEX's normal typesetting mechanism to position text and graphics on the page. In certain situation you may instead wish a certain text or graphic to appear at a page position that is specified *absolutely*. This means that the position is specified relative to the upper left corner of the slide.

The package `textpos` provides several commands for positioning text absolutely and it works together with BEAMER. When using this package, you will typically have to specify the options `overlay` and perhaps `absolute`. For details on how to use the package, please see its documentation.

5.3.9 Verse, Quotations, Quotes

LATEX defines three environments for typesetting quotations and verses: `verse`, `quotation`, and `quote`. These environments are also available in the BEAMER class, where they are overlay-specification-aware. If an overlay specification is given, the verse or quotation is shown only on the specified slides and is covered otherwise. The difference between a `quotation` and a `quote` is that the first has paragraph indentation, whereas the second does not.

Unlike the standard LATEX environments, in BEAMER these environments do not only change the left and right margins, but also the font: A verse is typeset using an italic serif font, quotations and quotes are typeset using an italic font (whether serif or sans-serif depends on the standard document font). To change this, you can adjust the templates for these environments.

5.3.10 Footnotes

First a word of warning: Using footnotes is usually not a good idea. They disrupt the flow of reading.

You can use the usual `\footnote` command. It has been augmented to take an additional option, for placing footnotes at the frame bottom instead of at the bottom of the current minipage.

`\footnote[`⟨*options*⟩`]{`⟨*text*⟩`}`

Inserts a footnote into the current frame. Footnotes will always be shown at the bottom of the current frame; they will never be "moved" to other frames. As usual, one can give a number as ⟨*options*⟩, which will cause the footnote to use that number. The BEAMER class adds one additional option:

- `frame` causes the footnote to be shown at the bottom of the frame. This is normally the default behavior anyway, but in minipages and certain blocks it makes a difference. In a minipage, the footnote is usually shown as part of the minipage rather than as part of the frame.

Example: `\footnote{On a fast machine.}`

Example: `\footnote[frame,2]{Not proved.}`

You can change the way footnotes are typeset by changing the footnote templates, see Section 9.4.14

6 Color Management

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

Since the `color` package and the `xcolor` package are loaded already by the BEAMER class, in order to pass options to these classes, you need to use the class options `color={`⟨*options for color*⟩`}` or `xcolor={`⟨*options for xcolor*⟩`}` to pass options to these classes.

The whole color management will hopefully be much improved in the future.

6.1 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 `beamerstructure`. 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 `beamerstructure`.
- All "alert" text is typeset by setting the default color and the structure color to 85% of red. To change this, you can either redefine the color `beameralert`, 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 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 9.4.5 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.

6.3 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, several commands install a predefined uncovering behavior.

`\beamertemplatetransparentcovered`

Makes all covered text quite transparent.

`\beamertemplatetransparentcoveredmedium`

Makes all covered text even more transparent.

`\beamertemplatetransparentcoveredhigh`

Makes all covered text highly transparent.

`\beamertemplatetransparentcoveredhigh`

Makes all covered text extremely transparent, but not totally.

`\beamertemplatetransparentcovereddynamic`

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

`\beamertemplatetransparentcovereddynamicmedium`

Like the previous command, only it the "range" of dynamics is smaller.

`\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
  {\opaqueness<1>{15}\opaqueness<2>{10}\opaqueness<3>{5}\opaqueness<4->{2}}
  {\opaqueness<1->{15}}
```

The ⟨*not yet list*⟩ and the ⟨*once more list*⟩ can contain any number of `\opaqueness` commands.

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

The ⟨*overlay specification*⟩ specifies on which slides covered text should have which ⟨*percentage of opaqueness*⟩. 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.

An opaqueness of 100 is fully opaque and 0 is fully transparent. Currently, since real transparency is not yet implemented, this command causes all colors to get a mixing of ⟨*percentage of opaqueness*⟩ of the current `averagebackgroundcolor`. At some future point this command might result in real transparency.

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.

7 Graphics, Animations, Sounds, and Slide Transitions

7.1 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.

7.1.1 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) or `.ps` (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.

  You can *also* include PostScript output from MetaPost when using the `graphics` package, since it implements some filters to convert such PostScript output to PDF on-the-fly. This is a nifty feature since it allows you to use MetaPost output for both PDF and PostScript. The extension of the file must be `.mps` for this to work.

Note that, frustratingly, most graphics format cannot be read by *both* `pdflatex` *and* `dvips`, except for `.mps`.

LYX You can use the usual "Insert Graphic" command to insert a graphic.

The commands `\includegraphics`, `\pgfuseimage`, and `\pgfimage` are overlay-specification-aware in BEAMER. If the overlay specification does not apply, the command has no effect. This is useful for creating a simple animation where each picture of the animation resides in a different file:

```
\frame{
  \includegraphics<1>[height=2cm]{step1.pdf}
  \includegraphics<2>[height=2cm]{step2.pdf}
  \includegraphics<3>[height=2cm]{step3.pdf}
}
```

7.1.2 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` back-end 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.

LYX Inlined graphics must currently by inserted in a large TeX-mode box. This is not very convenient.

7.2 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.

7.2.1 Including External Animation Files

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.

To include an animation in a presentation, you can use, for example, the package `multimedia.sty` which is part of the BEAMER package. You have to include this package explicitly. Despite being distributed as part of the BEAMER distribution, this package is perfectly self-sufficient and can be used independently of BEAMER.

`\usepackage{multimedia}`

A stand-alone package that implements several commands for including external animation and sound files in a PDF document. The package can be used together with both `dvips` plus `ps2pdf` and `pdflatex`, though the special sound support is available only in `pdflatex`.

When including this package, you must also include the `hyperref` package. Since you will typically want to include `hyperref` only at the very end of the preamble, `multimedia` will not include `hyperref` itself. However, `multimedia` can be included both before and after `hyperref`. Since BEAMER includes `hyperref` automatically, you need not worry about this when creating a presentation using BEAMER.

For including an animation in a PDF file, you can use the command `\movie`, which is explained below. Depending on the used options, this command will either setup the PDF file such that the viewer application (like the Acrobat Reader) itself will try to play the movie or that an external program will be called. The latter approach, though much less flexible, must be taken if the viewer application is unable to display the movie itself.

`\movie[`⟨*options*⟩`]{`⟨*poster text*⟩`}{`⟨*movie filename*⟩`}`

This command will insert the movie with the filename ⟨*movie filename*⟩ into the PDF file. The movie file must reside at some place where the viewer application will be able to find it, which is typically only the directory in which the final PDF file resides. The movie file will *not* be embedded into the PDF file in the sense that the actual movie data is part of the `main.pdf` file. The movie file must hence be copied and passed along with the PDF file. (Nevertheless, one often says that the movie is "embedded" in the document, but that just means that one can click on the movie when viewing the document and the movie will start to play.)

The movie will use a rectangular area whose size is determined either by the `width=` and `height=` options or by the size of the ⟨*poster text*⟩. The ⟨*poster text*⟩ can be any TEX text; for example, it might be a `\pgfuseimage` command or an `\includegraphics` command or a `pgfpicture` environment or just plain text. The ⟨*poster text*⟩ is typeset in a box, the box is inserted into the normal text, and the movie rectangle is put exactly over this box. Thus, if the ⟨*poster text*⟩ is an image from the movie, this image will be shown until the movie is started, when it will be exactly replaced by the movie itself. However, there is also a different, sometimes better, way of creating a poster image, namely by using the `poster` option as explained later on.

The aspect ratio of the movie will *not* be corrected automatically if the dimension of the ⟨*poster text*⟩ box does not have the same aspect ratio. Most movies have an aspect ratio of 4:3 or 16:9.

Despite the name, a movie may consist only of sound with no images. In this case, the ⟨*posert text*⟩ might be a symbol representing the sound. There is also a different, dedicated command for including sounds in a PDF file, see the `\sound` command in Section 7.3.

Unless further options are given, the movie will start only when the user clicks on it. Whether the viewer application can actually display the movie depends on the application and the version. For example, the Acrobat Reader up to version 5 does not seem to be able to display any movies or sounds on Linux. On the other hand, the Acrobat Reader Version 6 on MacOS is able to display anything that QuickTime can display, which is just about everything. Embedding movies in a PDF document is provided for by the PDF standard and is not a pecularity of the Acrobat Reader. In particular, one might expect other viewers like `xpdf` to support embedded movies in the future.

Example: `\movie{\pgfuseimage{myposterimage}}{mymovie.avi}`

Example: `\movie[width=3cm,height=2cm,poster]{}{mymovie.mpg}`

If your viewer application is not able to render your movie, but some external application is, you must use the `externalviewer` option. This will ask the viewer application to launch an application for showing the movie instead of displaying it itself. Since this application is started in a new window, this not nearly as nice as having the movie be displayed directly by the viewer (unless you use evil trickery to suppress the frame of the viewer application). Which application is choosen is left to the discreetion of the viewer application, which tries to make its choice according to the extension of the ⟨*movie filename*⟩ and according to some mapping table for mapping extensions to viewer applications. How this mapping table can be modified depends on the viewer application, please see the release notes of your viewer.

The following ⟨*options*⟩ may be given:

- `autostart`. Causes the movie to start playing immediately when the page is shown. At most one movie can be started in this way. The viewer application will typically be able to show only at most one movie at the same time anyway. When the page is no longer shown, the movie immediately stops. This can be a problem if you use the `\movie` command to include a sound that should be played on after the page has been closed. In this case, the `\sound` command must be used.
- `borderwidth=`⟨*TEX dimension*⟩. Causes a border of thickness ⟨*TEX dimension*⟩ to be drawn around the movie. Some versions of the Acrobat Reader seem to have a bug and do not display this border if is smaller than 0.5bp (about 0.51pt).
- `depth=`⟨*TEX dimension*⟩. Overrides the depth of the ⟨*poster text*⟩ box and sets it to the given dimension.
- `duration=`⟨*time*⟩`s`. Specifies in seconds how long the movie should be shown. The ⟨*time*⟩ may be a fractional value and must be followed by the letter `s`. For example, `duration=1.5s` will show the movie for one and a half seconds. In conjunction with the `start` option, you can "cut out" a part of a movie for display.
- `externalviewer`. As explained above, this causes an external application to be launched for displaying the movie in a separate window. Most options, like `duration` or `loop`, have no effect since they are not passed along to the viewer application.
- `height=`⟨*TEX dimension*⟩. Overrides the height of the ⟨*poster text*⟩ box and sets it to the given dimension.
- `label=`⟨*movie label*⟩. Assigns a label to the movie such that it can later be referenced by the command `\hyperlinkmovie`, which can be used to stop the movie or to show a different part of it. The ⟨*movie label*⟩ is not a normal label. It should not be too fancy, since it is inserted literally into the PDF code. In particular, it should not contain closing parantheses.
- `loop`. Causes the movie to start again when the end has been reached. Normally, the movie just stops at the end.
- `once`. Causes the movie to just stop at the end. This is the default.
- `palindrome`. Causes the movie to start playing backwards when the end has been reached, and to start playing forward once more when the beginning is reached, and so on.
- `poster`. Asks the viewer application to show the first image of the movie when the movie is not playing. Normally, nothing is shown when the movie is not playing (and thus the box containing the ⟨*poster text*⟩ is shown). For a movie that does not have any images (but sound) or for movies with an uninformative first image this option is not so useful.
- `repeat` is the same as `loop`.
- `showcontrols=`⟨*true or false*⟩. Causes a control bar to be displayed below the movie while it is playing. Instead of `showcontrols=true` you can also just say `showcontrols`. By default, no control bar is shown.

- `start=`⟨*time*⟩`s`. Causes the first ⟨*time*⟩ seconds of the movie to be skipped. For example, `start=10s,duration=5s` will show seconds 10 to 15 of the movie, when you play the movie.
- `width=`⟨*TEXdimension*⟩ works like the `height` option, only for the width of the poster box.

Example: The following example creates a "background sound" for the slide.

```
\movie[autostart]{}{test.wav}
```

Example: A movie with two extra buttons for showing different parts of the movie.

```
\movie[label=cells,width=4cm,height=3cm,poster,showcontrols,duration=5s]{}{cells.avi}

\hyperlinkmovie[start=5s,duration=7s]{cells}{\beamerbutton{Show the middle stage}}

\hyperlinkmovie[start=12s,duration=5s]{cells}{\beamerbutton{Show the late stage}}
```

A movie can serve as the destination of a special kind of hyperlink, namely a hyperlink introduced using the following command:

`\hyperlinkmovie[`⟨*options*⟩`]{`⟨*movie label*⟩`}{`⟨*text*⟩`}`

Causes the ⟨*text*⟩ to become a movie hyperlink. When you click on the ⟨*text*⟩, the movie with the label ⟨*movie label*⟩ will start to play (or stop or pause or resume, depending on the ⟨*options*⟩). The movie must be on the same page as the hyperlink.

The following ⟨*options*⟩ may be given, many of which are the same as for the `\movie` command; if a different option is given for the link than for the movie itself, the option for the link takes precedence:

- `duration=`⟨*time*⟩`s`. As for `\movie`, this causes the movie to be played only for the given number of seconds.
- `loop` and `repeat`. As for `\movie`, this causes the movie to loop.
- `once`. As for `\movie`, this causes the movie to played only once.
- `palindrome`. As for `\movie`, this causes the movie to be played forth and back.
- `pause`. Causes the playback of the movie to be paused, if the movie was currently playing. If not, nothing happens.
- `play`. Causes the movie to be played from whatever start position is specified. If the movie is already playing, it will be stopped and restarted at the starting position. This is the default.
- `resume`. Resumes playback of the movie, if it has previously been paused. If has not been paused, but not started or is already playing, nothing happens.
- `showcontrols=`⟨*true or false*⟩. As for `\movie`, this causes a control bar to be shown or not shown during playback.
- `start=`⟨*time*⟩`s`. As for `\movie`, this causes the given number of seconds to be skipped at the beginning of the movie if `play` is used to start the movie.
- `stop`. Causes the playback of the movie to be stopped.

7.2.2 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, the following 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 ...
}
```

ARTICLE This command is ignored in `article` mode.

\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}
    }
```

ARTICLE This command is ignored in `article` mode.

7.3 Sounds

You can include sounds in a presentation. Such sound can be played when you open a slide or when a certain button is clicked. The commands for including sounds are defined in the package `multimedia`, which is introduced in Section 7.2.1.

As was already pointed out in Section 7.2.1, a sound can be included in a PDF presentation by treating it as a movie and using the `\movie` command. While this is perfectly sufficient in most cases, there are two cases where this approach is not satisfactory:

1. When a page is closed, any playing movie is immediately stopped. Thus, you cannot use the `\movie` command to create sounds that persist for a longer time.
2. You cannot play two movies at the same time.

The PDF specification introduces special sound objects, which are treated quite differently from movie objects. You can create a sound object using the command `\sound`, which is somewhat similar to `\movie`. There also exists a `\hyperlinksound` command, which is similar to `\hyperlinkmovie`. While it is conceptually better to use `\sound` for sounds, there are a number of this to consider before using it:

- Several sounds *can* be played at the same time. In particular, it is possible to play a general sound in parallel to a (hopefully silent) movie.
- A sound playback *can* persist after the current page is closed (though it need not).
- The data of a sound file *can* be completely embedded in a PDF file, obliberating the need to "carry around" other files.
- The sound objects do *not* work together with `dvips` and `ps2pdf`. They only work with `pdflatex`.
- There is much less control over what part of a sound should be played. In particular, no control bar is shown and you can specify neither the start time nor the duration.
- A bug in some versions of the Acrobat Reader makes it necessary to provide very exact details on the encoding of the sound file. You have to provide the sampling rate, the number of channels (mono or stereo), the number of bits per sample, and the sample encoding method (raw, signed, Alaw or μlaw). If you do not know this data or provide it incorrectly, the sound will be played incorrectly.
- It seems that you can only include uncompressed sound data, which can easily become huge. This is not required by the specification, but I have been unable to make the Acrobat Reader play any compressed data. Data formats that *do* work are `.aif` and `.au`.

`\sound[`⟨*options*⟩`]{`⟨*sound poster text*⟩`}{`⟨*sound filename*⟩`}`

This command will insert the sound with the filename ⟨*sound filename*⟩ into the PDF file. As for `\movie`, the file must be accesible when the sound is to be played. Unlike `\movie`, you can however use the option `inlinesound` to actually embed the sound data in the PDF file.

Also as for a movie, the ⟨*sound poster text*⟩ will be be put in a box that, when clicked on, will start playing the movie. However, you might also leave this box empty and only use the `autostart` option. Once playback of a sound has started, it can only be stopped by starting the playback of a different sound or by use of the `\hyperlinkmute` command.

The supported sound formats depend on the viewer application. My version of the Acrobat Reader supports `.aif` and `.au`. I also need to specify information like the sampling rate, even though this information could be extracted from the sound file and even though the PDF standard specifies that the viewer application should do so. In this regard, my version of the Acrobat Reader seems to be non-standard-conforming.

This command only works together with `pdflatex`. If you use `dvips`, the poster is still shown, but clicking it has no effect and no sound is embedded in any way.

Example: `\sound[autostart,samplingrate=22050]{}{applause.au}`

The following ⟨*options*⟩ may be given:

- `autostart`. Causes the sound to start playing immediately when the page is shown.
- `automute`. Causes all sounds to be muted when the current page is left.
- `bitspersample=`⟨*8 or 16*⟩. Specifies the number of bits per sample in the sound file. If this number is 16, this option need not be specified.
- `channels=`⟨*1 or 2*⟩. Specifies whether the sound is mono or stereo. If the sound is mono, this option need not be specified.
- `depth=`⟨*TEX dimension*⟩. Overrides the depth of the ⟨*sound poster text*⟩ box and sets it to the given dimension.
- `encoding=`⟨*method*⟩. Specifies the encoding method, which may be `Raw`, `Signed`, `muLaw`, or `ALaw`. If the method is `muLaw`, this option need not be specified.
- `height=`⟨*TEX dimension*⟩. Overrides the height of the ⟨*sound poster text*⟩ box and sets it to the given dimension.
- `inlinesound` causes the sound data to be stored directly in the PDF-file.
- `label=`⟨*sound label*⟩. Assigns a label to the sound such that it can later be referenced by the command `\hyperlinksound`, which can be used to start a sound. The ⟨*sound label*⟩ is not a normal label.
- `loop` or `repeat`. Causes the sound to start again when the end has been reached.
- `mixsound=`⟨*true or false*⟩. If set to `true`, the sound is played in addition to any sound that is already playing. If set to `false` all other sounds (though not sound from movies) are stopped before the the sound is played. The default is `false`.
- `samplingrate=`⟨*number*⟩. Specifies the number of samples per second in the sound file. If this number is 44100, this option need not be specified.
- `width=`⟨*TEX dimension*⟩ works like the `height` option, only for the width of the poster box.

Example: The following example creates a "background sound" for the slide, assuming that `applause.au` is encoded correctly (44100 samples per second, mono, μlaw encoded, 16 bits per sample).

```
\sound[autostart]{}{applause.au}
```

Just like movies, sounds can also serve as destinations of special sound hyperlinks.

`\hyperlinksound[`⟨*options*⟩`]{`⟨*sound label*⟩`}{`⟨*text*⟩`}`

Causes the ⟨*text*⟩ to become a sound hyperlink. When you click on the ⟨*text*⟩, the sound with the label ⟨*sound label*⟩ will start to play.

The following ⟨*options*⟩ may be given:

- `loop` or `repeat`. Causes the sound to start again when the end has been reached.
- `mixsound=`⟨*true or false*⟩. If set to `true`, the sound is played in addition to any sound that is already playing. If set to `false` all other sounds (though not sound from movies) is stopped before the the sound is played. The default is `false`.

Since there is no direct way of stopping the playback of a sound, the following command is useful:

`\hyperlinkmute{`⟨*text*⟩`}`

Causes the ⟨*text*⟩ to become a hyperlink that, when clicked, stops the playback of all sounds.

7.4 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 interpret 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`.

ARTICLE All of these commands are ignored in `article` mode.

LYX You must insert these commands using TEX-mode.

`\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}`

8 Managing Non-Presentation Versions and Material

The BEAMER package offers different ways of creating special versions of your talk and adding material that is 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.

8.1 Creating Handouts

A *handout* is a version of a presentation in which the slides are 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 page, 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 *mode specifications.* They specify which slides of a frame should be shown for a special version, for example for the handout version. As explained in Section 4.1, a mode 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. Here is an example:

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

This specification says: "Normally (in `beamer` mode), 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 special mode 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 a mode specification for the overlay specification 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.

8.2 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.

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

8.3 Adding Notes

A *note* is a small piece of paper that is intended as a reminder to yourself of what you should say or should keep in mind when presenting a slide.

8.3.1 Specifying Note Contents

To add a note to a slide or a frame, you should use the `\note` command. This command can be used both inside and outside frames, but it has quite different behaviors then: Inside frames, `\note` commands accumulate and append a single note page after the current slide; outside frames each `\note` directly inserts a single note page with the given parameter as contents. Using the `\note` command inside frames is usually preferably over using them outside, since only commands issued inside frames profit from the class option `onlyslideswithnotes`, see below.

LYX In LyX, only the inside-frame `\note` command with the option `[item]` is available in the form of the NoteItem style.

Inside a frame, the effect of `\note`⟨*text*⟩ is the following: When you use it somewhere inside the frame on a specific slide, a note page is created after the slide, containing the ⟨*text*⟩. Since you can add an overlay specification to the `\note` command, you can specify after which slide the note should be shown. If you use multiple `\note` commands on one slide, they "accumulate" and are all shown on the same note.

To make the accumulation of notes more convenient, you can use the `\note` command with the option `[item]`. The notes added with this option are accumulated in an `enumerate` list that follows any text inserted using `\note`.

The following example will produce one note page that follows the second slide and has two entries.

```
\frame{
  \begin{itemize}
  \item<1-> Eggs
  \item<2-> Plants
    \note[item]<2>{Tell joke about plants.}
    \note[item]<2>{Make it short.}
  \item<3-> Animals
  \end{itemize}
}
```

Outside frames, the command `\note` creates a single note page. It is "independent" of any usage of the `\note` commands inside the previous frame. If you say `\note` inside a frame and `\note` right after it, *two* note pages are created.

In the following, the syntax and effects of the `\note` command *inside* frames is described:

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

Effects *inside* frames:

This command appends the ⟨*note text*⟩ to the note that follows the current slide. Multiple uses of this command on a slide accumulate. If you do not specify an ⟨*overlay specification*⟩, the note will be added to *all* slides of the current frame. This is often not what you want, so adding a specification like `<1>` is usually a good idea.

The following ⟨*options*⟩ may be given:

- `item` causes the note to be put as an item in a list that is shown at the end of the note page.

Example: `\note<2>{Do not talk longer than 2 minutes about this.}`

ARTICLE Notes are ignored in `article` mode.

LYX Use the NoteItem style to insert a note item.

Next, the syntax and effects of the `\note` command *outside* frames are described:

`\note[⟨options⟩]{⟨note text⟩}`

Outside frames, this command creates a note page. This command is *not* affected by the option `notes=onlyframeswithnotes`, see below.

The following ⟨*options*⟩ may be given:

- `itemize` will enclose the whole note page in an `itemize` environment. This is just a convenience.
- `enumerate` will enclose the whole note page in an `enumerate` environment.

Example:

```
\frame{some text}
\note{Talk no more than 1 minute.}

\note[enumerate]
{
\item Stress this first.
\item Then this.
}
```

ARTICLE Notes are ignored in `article` mode.

8.3.2 Specifying Which Notes and Frames Are Shown

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.

The `notes` class option takes several parameters whose effects are explained in the following.

`\documentclass[notes=hide]{beamer}`

Notes are not shown. This is the default in a presentation.

`\documentclass[notes=show]{beamer}`

Include notes in the output file. Normal slides are also included.

`\documentclass[notes=only]{beamer}`

Include only the notes in the output file and suppresses all frames. Useful for printing them. If you specify this command, the `.aux` and `.toc` files are *not* updated. So, if you add a section and reTEX your presentation, this will not be reflected in the navigation bars (which you do not see anyway since only notes are output).

`\documentclass[notes=onlyslideswithnotes]{beamer}`

This includes all notes and those slides that *contain* a `\note`. Frames that are just “followed” by a `\note` command will not be included.

If you use only `\note` commands, this option will cause the frames and the notes that apply to them to be nicely paired. This is useful for printing.

8.3.3 Changing the Appearance of Notes

By default, notes are put on a page that contains your text, some information that should make it easier to match the note to the frame while talking, and a little "mini version" of the slide coming before the note (this mini version contains only the body of the frame, the headline, footline, and sidebars are not shown).

You can change this appearance by specifying a different template for note rendering, see Section 9.4.21 for details. In most cases it will be sufficient to say either `\beamertemplatenoteplain` in your preamble, which will give you "plain" notes without anything on them but your text, or `\beamertemplatenotecompress`, which will give you notes with more space on them.

8.4 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 `llncs` 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.

8.4.1 Starting the Article 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 one: `\mode`. 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*⟩.

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`.

Since BEAMER gives over control to another class almost immediately, none of the usual commands like `\frame` and so on are defined in `article` mode initially. The only command that is guaranteed to be defined is `\mode`; which can be used to "comment out" all of BEAMER's commands. For example, in your preamble you might write things like

```
\mode<presentation>{\usepackage{beamerthemeshadow}}
\mode<article>{\usepackage{fullpage}}
\mode<all>{
  \usepackage{times}
  \newcommand{\myfavoritecommand}{...}
}
```

However, for the main text this is very bothersome and there is a much better way: You can include the package `beamerbasearticle`. This package will define virtually all of BEAMER's commands in a way that is sensible for the `article` mode. Also, overlay specifications can be given to commands like `\textbf` or `\item` once `beamerbasearticle` has been loaded. Note that, except for `\item` these overlay specifications also work: by writing `\section<presentation>{Name}` you will suppress this section command in the article version. For the exact effects overlay specifications have in `article` mode, please see the descriptions of the commands to which you wish to apply them.

`\usepackage[`⟨*options*⟩`]{beamerbasearticle}`

Makes most BEAMER commands available for an article.

The following ⟨*options*⟩ may be given:

- activeospeccharacters will leave the character code of the characters used in overlay specifications as specified by other packages. Normally, BEAMER will turn off the special behaviour of characters like : or ! in styles like french since they clash with the implementation of overlay specifications. Using this option, you can reinstall the original behaviour at the price of possible problems when using overlay specifications in the article mode.
- noamsthm will suppress the loading of the amsthm package. No theorems will be defined.
- notheorem will suppress the definition of standard environments like theorem, but amsthm is still loaded and the \newtheorem command still makes the defined environments overlay-specification-aware. Using this option allows you to define the standard environments in whatever way you like while retaining the power of the extensions to amsthm.
- envcountsect causes theorem, definitions and the like to be numbered with each section. Thus instead of Theorem 1 you get Theorem 1.1. I recommend using this option.
- noxcolor will suppress the loading of the xcolor package. No colors will be defined.

There is one remaining problem: While the article version can easily TeX the whole file, even in the presence of commands like \frame<2>, we do not want the special article text to be inserted into our original BEAMER presentation. That means, we would like all text *between* frames to be suppressed. More precisely, we want all text except for commands like \section and so on to be suppressed. This behaviour can be enforced by specifying the option ignorenonframetext in the presentation version. The option will insert a \mode* at the beginning of your presentation.

The following example shows a simple usage of the article mode:

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

\mode<article>{\usepackage{fullpage}}
\mode<presentation>{\usepackage{beamerthemesplit}}

% everyone:
\usepackage[english]{babel}
\usepackage{pgf}

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

\begin{document}

\section{Introduction}

This is the introduction text. This text is not shown in the
presentation, but will be part of the article.

\frame{
  \begin{figure}
    % In the article, this is a floating figure,
    % In the presentation, this figure is shown in the first frame
    \pgfuseimage{myimage}
  \end{figure}
}

This text is once more not shown in the presentation.

\section{Main Part}

While this text is not shown in the presentation, the section command
also applies to the presentation.

We can add a subsection that is only part of the article like this:
```

```
\subsection<article>{Article-Only Section}

With some more text.

\frame{
  This text is part both of the article and of the presentation.
  \begin{itemize}
  \item This stuff is also shown in both version.
  \item This too.
  \only<article>{\item This particular item is only part
      of the article version.}
  \item<presentation:only@0> This text is also only part of the article.
  \end{itemize}
}
\end{document}
```

There is one command whose behaviour is a bit special in `article` mode: The line break command `\\`. Inside frames, this command has no effect in `article` mode, except if an overlay specification is present. Then it has the normal effect dictated by the specification. The reason for this behaviour is that you will typically inserts lots of `\\` commands in a presentation in order to get control over all line breaks. These line breaks are mostly superfluous in `article` mode. If you really want a line break to apply in all versions, say `\\<all>`. Note that the command `\\` is often redefined by certain environments, so it may not always be overlay-specification-aware. In such a case you have to write something like `\only<presentation>{\\}`.

8.4.2 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[ignorenonframetext]{beamer}
  \input{main.tex}
  ```

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

  ```
  \documentclass[class=article]{beamer}
  \usepackage{beamerbasearticle}
  \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 presentation 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`.

8.4.3 Including Slides from the Presentation Version in the Article Version

If you use the package `beamerbasearticle`, the `\frame` command becomes available in `article` mode. By adjusting the frame template, you can "mimic" the appearance of frames typeset by BEAMER in your articles. However, sometimes you may wish to insert "the real thing" into the `article` version, that is, a precise "screenshot" of a slide from the presentation. The commands introduced in the following help you do exactly this.

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`.

8.5 Details on Modes

This subsection describes how modes work exactly and how you can use the `\mode` command to control what part of your text belongs to which mode.

When BEAMER typesets your text, it is always in one of the following four modes:

- `beamer` is the default mode.
- `handout` is the mode for creating handouts.
- `trans` is the mode for creating transparencies.

- `article` is the mode when control has been transferred to another class, like `article.cls`. Note that the mode is also `article` if control is transferred to, say, `book.cls`.

In addition to these modes, BEAMER recognizes the following names for modes sets:

- `all` refers to all modes.
- `presentation` refers to the first three modes, that is, to all modes except for the `article` mode.

Depending on the current mode, you may wish to have certain text inserted only in that mode. For example, you might wish a certain frame or a certain table to be left out of your article version. In some situations, you can use the `\only` command for this purpose. However, the command `\mode`, which is described in the following, is much more powerful than `\only`.

The command actually comes in three "flavors," which only slightly differ in syntax. The first, and simplest, is the version that takes one argument. It behaves essentially the same way as `\only`.

`\mode<`⟨*mode specification*⟩`>{`⟨*text*⟩`}`

Causes the ⟨*text*⟩ to be inserted only for the specified modes. Recall that a ⟨*mode specification*⟩ is just an overlay specification in which no slides are mentioned.

The ⟨*text*⟩ should not do anything fancy that involves mode switches or including other files. In particular, you should not put an `\include` command inside ⟨*text*⟩. Use the argument-free form below, instead.

Example:

```
\mode<article>{Extra detail mentioned only in the article version.}

\mode
<beamer| trans>
{\frame{\tableofcontents[current]}}
```

The second flavor of the `\mode` command takes no argument. "No argument" means that it is not followed by an opening brace, but any other symbol.

`\mode<`⟨*mode specification*⟩`>`

In the specified mode, this command actually has no effect. The interesting part is the effect in the non-specified modes: In these modes, the command causes TEX to enter a kind of "gobbling" state. It will now ignore all following lines until the next line that has a sole occurrence of one of the following commands: `\mode`, `\mode*`, `\begin{document}`, `\end{document}`. Even a comment on this line will make TEX skip it. Note that the line with the special commands that make TEX stop gobbling may not directly follow the line where the gobbling is started. Rather, there must either be one non-empty line before the special command or at least two empty lines.

When TEX encounters a single `\mode` command, it will execute this command. If the command is `\mode` command of the first flavor, TEX will resume its "gobbling" state after having inserted (or not inserted) the argument of the `\mode` command. If the `\mode` command is of the second flavor, it takes over.

Using this second flavor of `\mode` is less convenient than the first, but there are different reasons why you might need to use it:

- The line-wise gobbling is much faster than the gobble of the third flavor, explained below.
- The first flavor reads its argument completely. This means, it cannot contain any verbatim text that contains unbalanced braces.
- The first flavor cannot cope with arguments that contain `\include`.
- If the text mainly belongs to one mode with only small amounts of text from another mode inserted, this second flavor is nice to use.

Note: When searching line-wise for a `\mode` command to shake it out of its gobbling state, TEX will not recognize a `\mode` command if a mode specification follows on the same line. Thus, such a specification must be given on the next line.

Note: When a TEX file ends, TEX must not be in the gobbling state. Switch this state off using `\mode` on one line and `<all>` on the next.

Example:

```
\mode<article>

This text is typeset only in |article| mode.
\verb!verbatim text is ok {!

\mode
<presentation>
{ % this text is inserted only in presentation mode
\frame{\tableofcontents[current]}}

Here we are back to article mode stuff. This text
is not inserted in presentation mode

\mode
<presentation>

This text is only inserted in presentation mode.
```

The last flavor of the mode command behaves quite differently.

`\mode*`

The effect of this mode is to ignore all text outside frames in the `presentation` modes. In article mode it has no effect.

This mode should only be entered outside of frames. Once entered, if the current mode is a `presentation` mode, TEX will enter a gobbling state similar to the gobbling state of the second "flavor" of the `\mode` command. The difference is that the text is now read token-wise, not line-wise. The text is gobbled token by token until one of the following tokens is found: `\mode`, `\frame`, `\againframe`, `\part`, `\section`, `\subsection`, `\appendix`, `\note`, and `\end{document}` (this is not really a token, but it is recognized anyway).

Once one of these commands is encountered, the gobbling stops and the command is executed. However, all of these commands restore the mode that was in effect when they started. Thus, once the command is finished, TEX returns to its gobbling.

Normally, `\mode*` is exactly what you want TEX to do outside of frames: ignore everything except for the above-mentioned commands outside frames in `presentation` mode. However, there are cases in which you have to use the second flavor of the `\mode` command instead: If you have verbatim text that contains one of the commands, if you have very long text outside frames, or if you wish some text outside a frame (like a definition) to be executed also in `presentation` mode.

The class option `ignorenonframetext` will switch on `\mode*` at the beginning of the document.

Example:

```
\begin{document}
\mode*

This text is not shown in the presentation.

\frame
{
  This text is shown both in article and presentation mode.
}
```

```
this text is not shown in the presentation again.

\section{This command also has effect in presentation mode}

Back to article stuff again.

\frame<presentation>
{ this frame is shown only in the presentation. }
\end{document}
```

Example: The following example shows how you can include other files in a main file. The contents of a `main.tex`:

```
\documentclass[ignorenonframetext]{beamer}
\begin{document}
This is star mode stuff.

Let's include files:
\mode<all>
\include{a}
\include{b}
\mode*

Back to star mode
\end{document}
```

And `a.tex` (and likewise `b.tex`):

```
\mode*
\section{First section}
Extra text in article version.
\frame
{
  Some text.
}
\mode<all>
```

9 Customization

9.1 Fonts

9.1.1 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 French *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 more conservative impression, which might be exactly what you wish to create.
- On projections with a very high resolutions serif text is just as readable as sans-serif text.

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.

9.1.2 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 most 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 is typeset 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.

9.1.3 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.

If you use professional fonts (fonts that you buy and that come with a complete set of every symbol in all modes), you may need to specify the class option `professionalfont`. This will tell BEAMER that it should not meddle with the fonts you use. The reason is that BEAMER normally replaces certain character glyphs in mathematical text by more appropriate versions. For example, BEAMER will normally replace glyphs such that the italic characters from the main font are used for variables in mathematical text. If your professional font package takes care of this already, BEAMER's meddling should be switched off. Note that BEAMER's substitution is automatically turned off if one of the following packages is loaded: `mathtime`, `mathpmnt`, `lucidabr`, `mtpro`, and `hvmath`. If your favorite professional font package is not among these, use the `professionalfont` option (and write me an email, so that the package can be added).

`\documentclass[professionalfont]{beamer}`

Deactivates BEAMER's internal font replacements for mathematical text. This option should be used if you use a professional font package that sets up all mathematical fonts correctly.

9.1.4 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 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.

9.1.5 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 encoding (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 word Fräulein) will work only if you use the T1 encoding. Unfortunately, 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).

To select the T1 encoding, use `\usepackage[T1]{fontenc}`. Thus, if you have the `lmodern` fonts installed, you could write

```
\usepackage{lmodern}
\usepackage[T1]{fontenc}
```

to get beautiful outline fonts and correct hyphenation.

9.2 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 sidebar. Thus, it is the distance between the right edge of the left sidebar 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}`

ARTICLE This command has no effect in `article` mode.

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

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

For more information on sidebars, see Section 9.4.10.

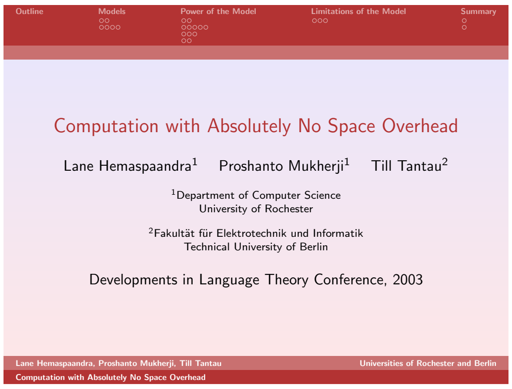
9.3 Themes

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:



Outline Models Power of the Model Limitations of the Model Summary

Computation with Absolutely No Space Overhead

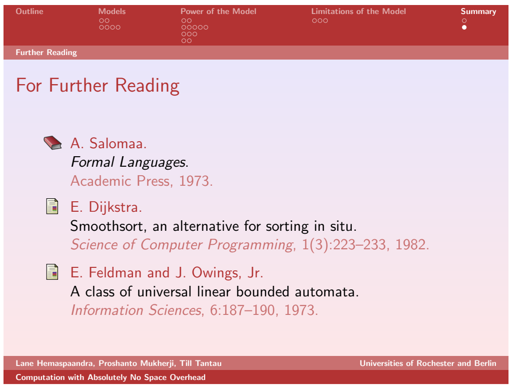
Lane Hemaspaandra[1] Proshanto Mukherji[1] Till Tantau[2]

[1]Department of Computer Science
University of Rochester

[2]Fakultät für Elektrotechnik und Informatik
Technical University of Berlin

Developments in Language Theory Conference, 2003

Lane Hemaspaandra, Proshanto Mukherji, Till Tantau Universities of Rochester and Berlin
Computation with Absolutely No Space Overhead



Outline Models Power of the Model Limitations of the Model Summary

Further Reading

For Further Reading

A. Salomaa.
Formal Languages.
Academic Press, 1973.

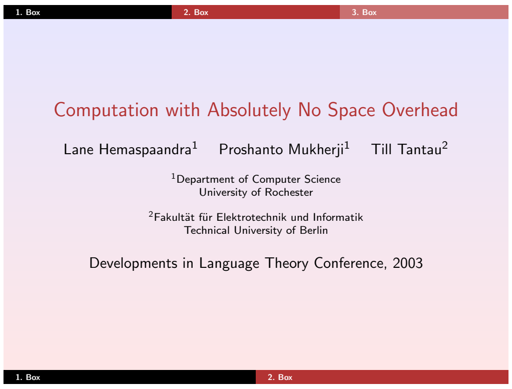
E. Dijkstra.
Smoothsort, an alternative for sorting in situ.
Science of Computer Programming, 1(3):223–233, 1982.

E. Feldman and J. Owings, Jr.
A class of universal linear bounded automata.
Information Sciences, 6:187–190, 1973.

Lane Hemaspaandra, Proshanto Mukherji, Till Tantau Universities of Rochester and Berlin
Computation with Absolutely No Space Overhead

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

Example:



1. Box 2. Box 3. Box

Computation with Absolutely No Space Overhead

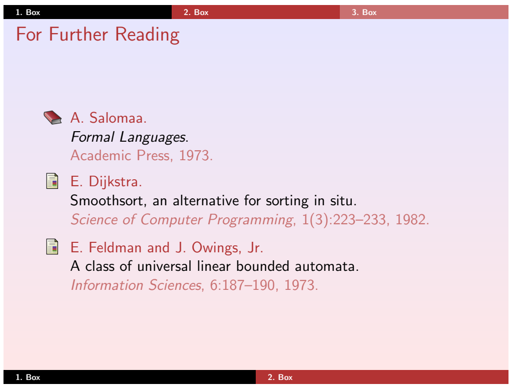
Lane Hemaspaandra[1] Proshanto Mukherji[1] Till Tantau[2]

[1]Department of Computer Science
University of Rochester

[2]Fakultät für Elektrotechnik und Informatik
Technical University of Berlin

Developments in Language Theory Conference, 2003

1. Box 2. Box



1. Box 2. Box 3. Box

For Further Reading

A. Salomaa.
Formal Languages.
Academic Press, 1973.

E. Dijkstra.
Smoothsort, an alternative for sorting in situ.
Science of Computer Programming, 1(3):223–233, 1982.

E. Feldman and J. Owings, Jr.
A class of universal linear bounded automata.
Information Sciences, 6:187–190, 1973.

1. Box 2. Box

Example:

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

For this theme, you can specify an arbitrary number of templates for the boxes in the headline and in the footline. 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}
```

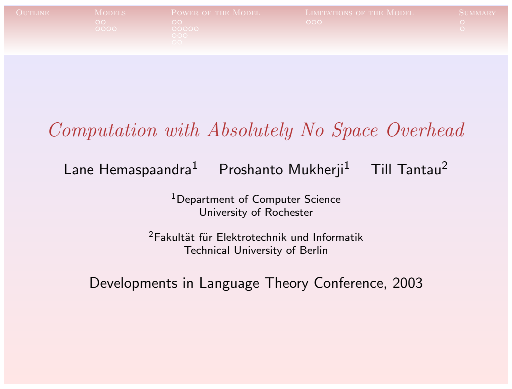
\addfootboxtemplate{⟨*background color command*⟩}{⟨*box template*⟩}

Example:

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

`\usepackage{beamerthemeclassic}`

Example:


Outline Models Power of the Model Limitations of the Model Summary

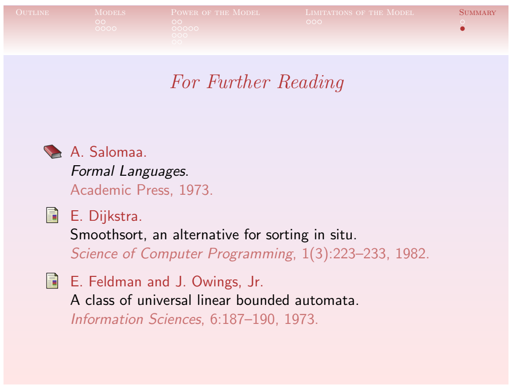
Computation with Absolutely No Space Overhead

Lane Hemaspaandra[1] Proshanto Mukherji[1] Till Tantau[2]

[1]Department of Computer Science
University of Rochester

[2]Fakultät für Elektrotechnik und Informatik
Technical University of Berlin

Developments in Language Theory Conference, 2003


Outline Models Power of the Model Limitations of the Model Summary

For Further Reading

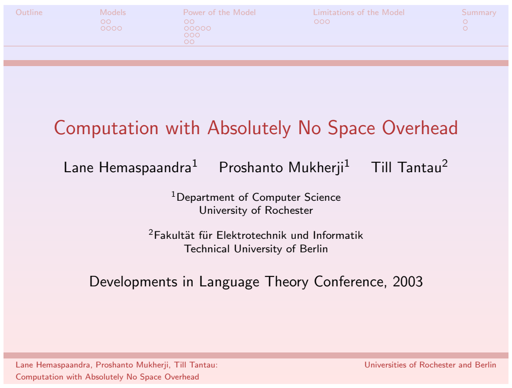
A. Salomaa.
Formal Languages.
Academic Press, 1973.

E. Dijkstra.
Smoothsort, an alternative for sorting in situ.
Science of Computer Programming, 1(3):223–233, 1982.

E. Feldman and J. Owings, Jr.
A class of universal linear bounded automata.
Information Sciences, 6:187–190, 1973.

`\usepackage{beamerthemelined}`

Example:


Outline Models Power of the Model Limitations of the Model Summary

Computation with Absolutely No Space Overhead

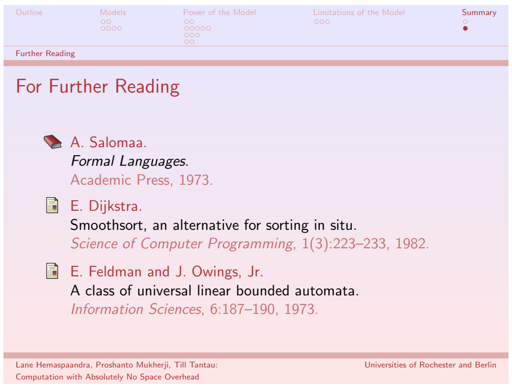
Lane Hemaspaandra[1] Proshanto Mukherji[1] Till Tantau[2]

[1]Department of Computer Science
University of Rochester

[2]Fakultät für Elektrotechnik und Informatik
Technical University of Berlin

Developments in Language Theory Conference, 2003

Lane Hemaspaandra, Proshanto Mukherji, Till Tantau:
Computation with Absolutely No Space Overhead
Universities of Rochester and Berlin


Outline Models Power of the Model Limitations of the Model Summary

Further Reading

For Further Reading

A. Salomaa.
Formal Languages.
Academic Press, 1973.

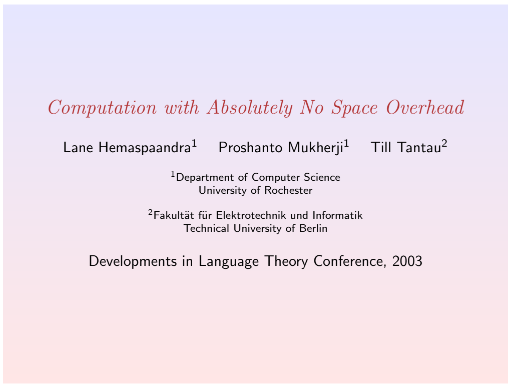
E. Dijkstra.
Smoothsort, an alternative for sorting in situ.
Science of Computer Programming, 1(3):223–233, 1982.

E. Feldman and J. Owings, Jr.
A class of universal linear bounded automata.
Information Sciences, 6:187–190, 1973.

Lane Hemaspaandra, Proshanto Mukherji, Till Tantau:
Computation with Absolutely No Space Overhead
Universities of Rochester and Berlin

`\usepackage{beamerthemeplain}`

Example:

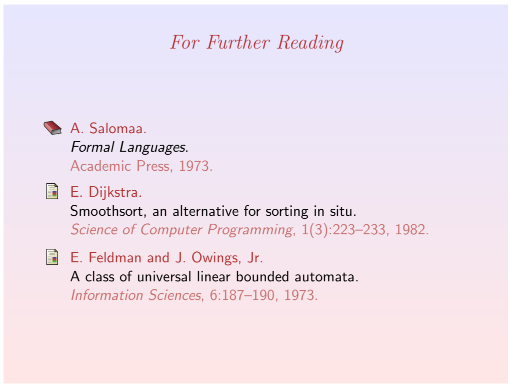

Computation with Absolutely No Space Overhead

Lane Hemaspaandra[1] Proshanto Mukherji[1] Till Tantau[2]

[1]Department of Computer Science
University of Rochester

[2]Fakultät für Elektrotechnik und Informatik
Technical University of Berlin

Developments in Language Theory Conference, 2003


For Further Reading

A. Salomaa.
Formal Languages.
Academic Press, 1973.

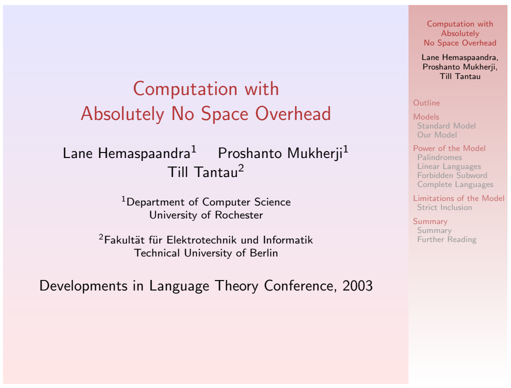
E. Dijkstra.
Smoothsort, an alternative for sorting in situ.
Science of Computer Programming, 1(3):223–233, 1982.

E. Feldman and J. Owings, Jr.
A class of universal linear bounded automata.
Information Sciences, 6:187–190, 1973.

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

The option `width` sets the width of the sidebar to ⟨*sidebar width*⟩. The option `dark` makes the sidebar and the whole theme darked. The option `tab` causes the current section or subsection to be hilighted by changing the background behind the entry, rather than hilighting the entry itself.

Example: `\usepackage{beamerthemesidebar}`



Computation with Absolutely No Space Overhead

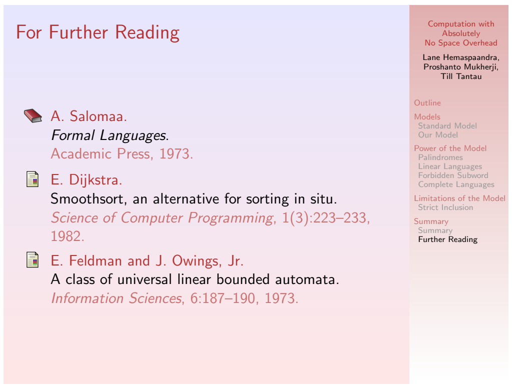
Lane Hemaspaandra[1] Proshanto Mukherji[1]
Till Tantau[2]

[1]Department of Computer Science
University of Rochester

[2]Fakultät für Elektrotechnik und Informatik
Technical University of Berlin

Developments in Language Theory Conference, 2003

Computation with Absolutely No Space Overhead
Lane Hemaspaandra, Proshanto Mukherji, Till Tantau
Outline
Models
Standard Model
Our Model
Power of the Model
Palindromes
Linear Languages
Forbidden Subword
Complete Languages
Limitations of the Model
Strict Inclusion
Summary
Summary
Further Reading



For Further Reading

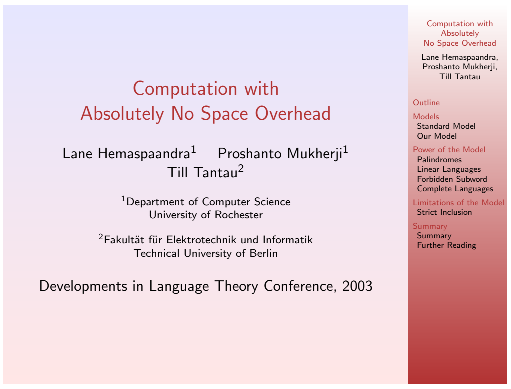
A. Salomaa.
Formal Languages.
Academic Press, 1973.

E. Dijkstra.
Smoothsort, an alternative for sorting in situ.
Science of Computer Programming, 1(3):223–233, 1982.

E. Feldman and J. Owings, Jr.
A class of universal linear bounded automata.
Information Sciences, 6:187–190, 1973.

Computation with Absolutely No Space Overhead
Lane Hemaspaandra, Proshanto Mukherji, Till Tantau
Outline
Models
Standard Model
Our Model
Power of the Model
Palindromes
Linear Languages
Forbidden Subword
Complete Languages
Limitations of the Model
Strict Inclusion
Summary
Summary
Further Reading

Example: `\usepackage[tab]{beamerthemesidebar}`



Computation with Absolutely No Space Overhead

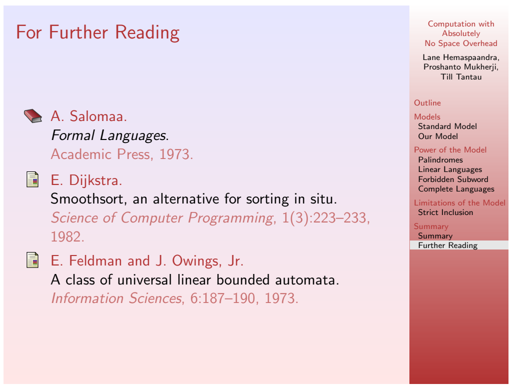
Lane Hemaspaandra[1] Proshanto Mukherji[1]
Till Tantau[2]

[1]Department of Computer Science
University of Rochester

[2]Fakultät für Elektrotechnik und Informatik
Technical University of Berlin

Developments in Language Theory Conference, 2003

Computation with Absolutely No Space Overhead
Lane Hemaspaandra, Proshanto Mukherji, Till Tantau
Outline
Models
Standard Model
Our Model
Power of the Model
Palindromes
Linear Languages
Forbidden Subword
Complete Languages
Limitations of the Model
Strict Inclusion
Summary
Summary
Further Reading



For Further Reading

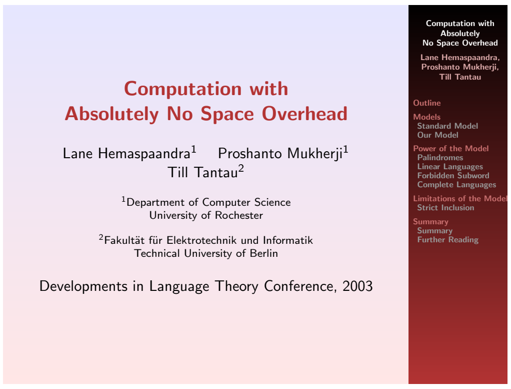
A. Salomaa.
Formal Languages.
Academic Press, 1973.

E. Dijkstra.
Smoothsort, an alternative for sorting in situ.
Science of Computer Programming, 1(3):223–233, 1982.

E. Feldman and J. Owings, Jr.
A class of universal linear bounded automata.
Information Sciences, 6:187–190, 1973.

Computation with Absolutely No Space Overhead
Lane Hemaspaandra, Proshanto Mukherji, Till Tantau
Outline
Models
Standard Model
Our Model
Power of the Model
Palindromes
Linear Languages
Forbidden Subword
Complete Languages
Limitations of the Model
Strict Inclusion
Summary
Summary
Further Reading

Example: `\usepackage[dark]{beamerthemesidebar}`



Computation with Absolutely No Space Overhead

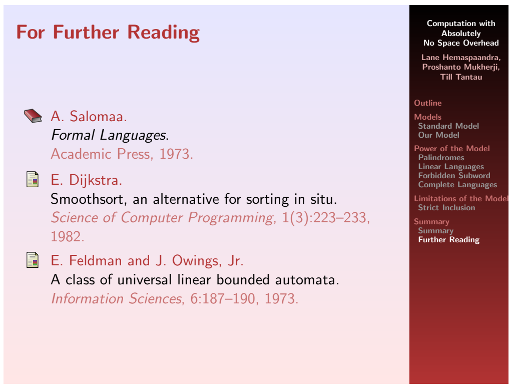
Lane Hemaspaandra[1] Proshanto Mukherji[1]
Till Tantau[2]

[1]Department of Computer Science
University of Rochester

[2]Fakultät für Elektrotechnik und Informatik
Technical University of Berlin

Developments in Language Theory Conference, 2003

Computation with Absolutely No Space Overhead
Lane Hemaspaandra, Proshanto Mukherji, Till Tantau
Outline
Models
Standard Model
Our Model
Power of the Model
Palindromes
Linear Languages
Forbidden Subword
Complete Languages
Limitations of the Model
Strict Inclusion
Summary
Summary
Further Reading



For Further Reading

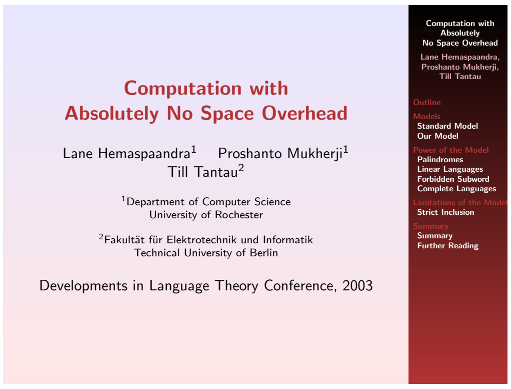
A. Salomaa.
Formal Languages.
Academic Press, 1973.

E. Dijkstra.
Smoothsort, an alternative for sorting in situ.
Science of Computer Programming, 1(3):223–233, 1982.

E. Feldman and J. Owings, Jr.
A class of universal linear bounded automata.
Information Sciences, 6:187–190, 1973.

Computation with Absolutely No Space Overhead
Lane Hemaspaandra, Proshanto Mukherji, Till Tantau
Outline
Models
Standard Model
Our Model
Power of the Model
Palindromes
Linear Languages
Forbidden Subword
Complete Languages
Limitations of the Model
Strict Inclusion
Summary
Summary
Further Reading

Example: `\usepackage[dark,tab]{beamerthemesidebar}`



Computation with Absolutely No Space Overhead

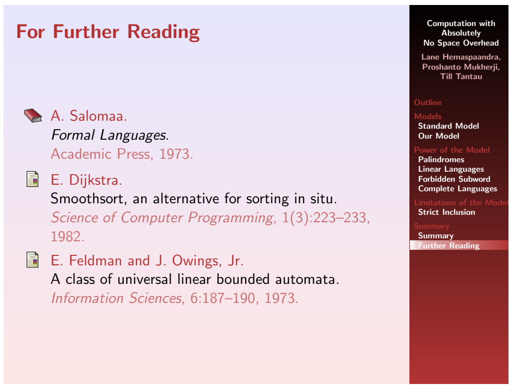
Lane Hemaspaandra[1] Proshanto Mukherji[1]
Till Tantau[2]

[1]Department of Computer Science
University of Rochester

[2]Fakultät für Elektrotechnik und Informatik
Technical University of Berlin

Developments in Language Theory Conference, 2003

Computation with Absolutely No Space Overhead
Lane Hemaspaandra, Proshanto Mukherji, Till Tantau
Outline
Models
Standard Model
Our Model
Power of the Model
Palindromes
Linear Languages
Forbidden Subword
Complete Languages
Limitations of the Model
Strict Inclusion
Summary
Summary
Further Reading



For Further Reading

A. Salomaa.
Formal Languages.
Academic Press, 1973.

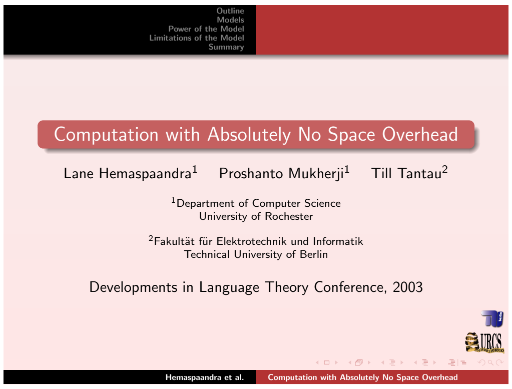
E. Dijkstra.
Smoothsort, an alternative for sorting in situ.
Science of Computer Programming, 1(3):223–233, 1982.

E. Feldman and J. Owings, Jr.
A class of universal linear bounded automata.
Information Sciences, 6:187–190, 1973.

Computation with Absolutely No Space Overhead
Lane Hemaspaandra, Proshanto Mukherji, Till Tantau
Outline
Models
Standard Model
Our Model
Power of the Model
Palindromes
Linear Languages
Forbidden Subword
Complete Languages
Limitations of the Model
Strict Inclusion
Summary
Summary
Further Reading

\usepackage{beamerthemeshadow}

Example:



Outline
Models
Power of the Model
Limitations of the Model
Summary

Computation with Absolutely No Space Overhead

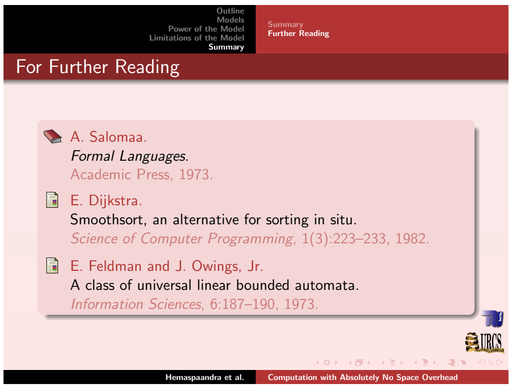
Lane Hemaspaandra[1] Proshanto Mukherji[1] Till Tantau[2]

[1]Department of Computer Science
University of Rochester

[2]Fakultät für Elektrotechnik und Informatik
Technical University of Berlin

Developments in Language Theory Conference, 2003

Hemaspaandra et al. Computation with Absolutely No Space Overhead



Outline
Models
Power of the Model
Limitations of the Model
Summary

Summary
Further Reading

For Further Reading

A. Salomaa.
Formal Languages.
Academic Press, 1973.

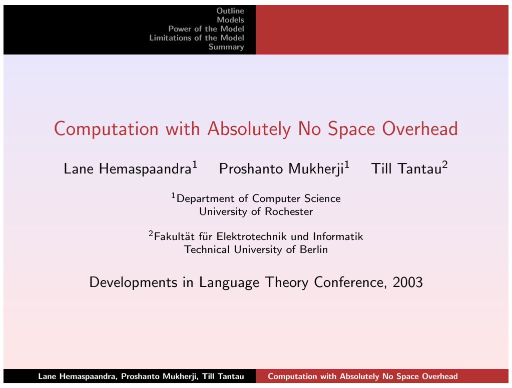
E. Dijkstra.
Smoothsort, an alternative for sorting in situ.
Science of Computer Programming, 1(3):223–233, 1982.

E. Feldman and J. Owings, Jr.
A class of universal linear bounded automata.
Information Sciences, 6:187–190, 1973.

Hemaspaandra et al. Computation with Absolutely No Space Overhead

\usepackage{beamerthemesplit}

Example:



Outline
Models
Power of the Model
Limitations of the Model
Summary

Computation with Absolutely No Space Overhead

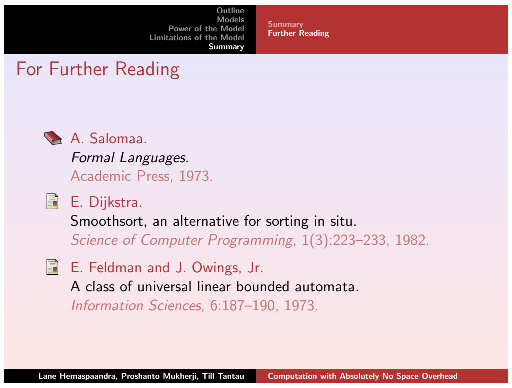
Lane Hemaspaandra[1] Proshanto Mukherji[1] Till Tantau[2]

[1]Department of Computer Science
University of Rochester

[2]Fakultät für Elektrotechnik und Informatik
Technical University of Berlin

Developments in Language Theory Conference, 2003

Lane Hemaspaandra, Proshanto Mukherji, Till Tantau Computation with Absolutely No Space Overhead



Outline
Models
Power of the Model
Limitations of the Model
Summary

Summary
Further Reading

For Further Reading

A. Salomaa.
Formal Languages.
Academic Press, 1973.

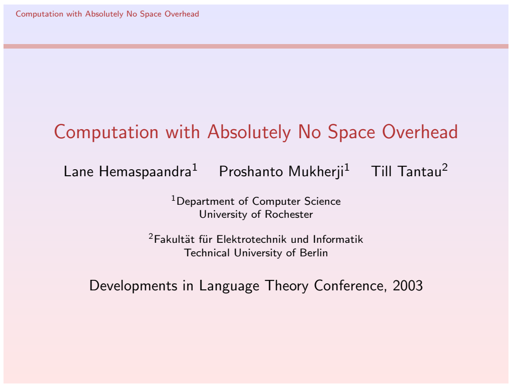
E. Dijkstra.
Smoothsort, an alternative for sorting in situ.
Science of Computer Programming, 1(3):223–233, 1982.

E. Feldman and J. Owings, Jr.
A class of universal linear bounded automata.
Information Sciences, 6:187–190, 1973.

Lane Hemaspaandra, Proshanto Mukherji, Till Tantau Computation with Absolutely No Space Overhead

\usepackage{beamerthemetree}

Example:



Computation with Absolutely No Space Overhead

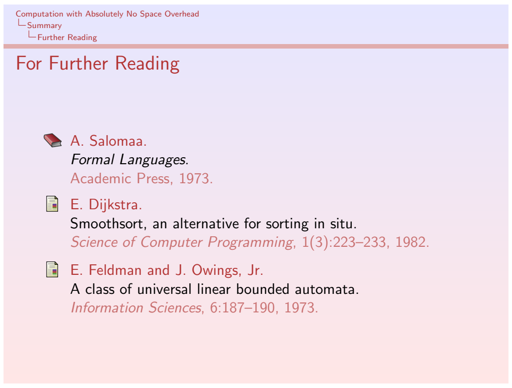
Computation with Absolutely No Space Overhead

Lane Hemaspaandra[1] Proshanto Mukherji[1] Till Tantau[2]

[1]Department of Computer Science
University of Rochester

[2]Fakultät für Elektrotechnik und Informatik
Technical University of Berlin

Developments in Language Theory Conference, 2003



Computation with Absolutely No Space Overhead
└Summary
 └Further Reading

For Further Reading

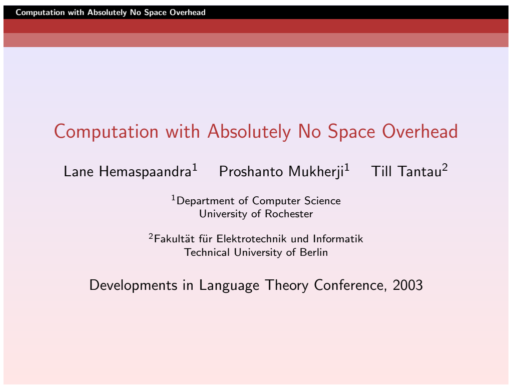
A. Salomaa.
Formal Languages.
Academic Press, 1973.

E. Dijkstra.
Smoothsort, an alternative for sorting in situ.
Science of Computer Programming, 1(3):223–233, 1982.

E. Feldman and J. Owings, Jr.
A class of universal linear bounded automata.
Information Sciences, 6:187–190, 1973.

```
\usepackage[bar]{beamerthemetree}
```

Example:



Computation with Absolutely No Space Overhead

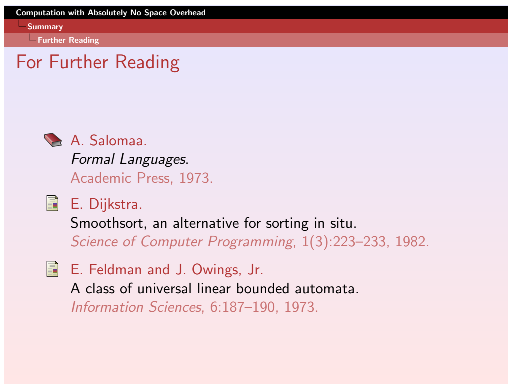
Computation with Absolutely No Space Overhead

Lane Hemaspaandra[1] Proshanto Mukherji[1] Till Tantau[2]

[1]Department of Computer Science
University of Rochester

[2]Fakultät für Elektrotechnik und Informatik
Technical University of Berlin

Developments in Language Theory Conference, 2003



Computation with Absolutely No Space Overhead
Summary
Further Reading

For Further Reading

A. Salomaa.
Formal Languages.
Academic Press, 1973.

E. Dijkstra.
Smoothsort, an alternative for sorting in situ.
Science of Computer Programming, 1(3):223–233, 1982.

E. Feldman and J. Owings, Jr.
A class of universal linear bounded automata.
Information Sciences, 6:187–190, 1973.

9.4 Templates

9.4.1 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.

ARTICLE In `article` mode, most of the template mechanism is switched off and has no effect. However, a few templates are also available. If this is the case, it is specially indicated.

Some of the below subsections start with commands for using *predefined* templates. Calling one of them will change a template in a predefined way, making it unnecessary to worry about how exactly one creates, say, these cute little balls in different sizes. Using them, you can use, for example, your favorite theme together with a shading background and a numbered table of contents.

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. The file `beamerbasetemplates` might be useful starting point.
- 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 (headlines, sidebars, etc.) can only be changed in the preamble. Other templates can be changed during the document.
- The height of the headline and footline templates is calculated automatically. This is done by typesetting the templates and then "having a look" at their heights. This recalculation is done right at the beginning of the document, *after* all packages have been loaded and even *after* these have executed their `\AtBeginDocument` initialization.
- The left and right margins of the head- and footline templates are the same as of the normal text. In order to start the headline and footline 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.

9.4.2 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 footline in the title page, use `\frame[plain]{\titlepage}`.

Inserts for these Templates

`\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.

`\insertsubtitle`

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

`\inserttitlegraphic`

Inserts the title graphic into a template.

9.4.3 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 these Templates

`\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.

9.4.4 Frames

Template Installation Commands

`\useframetemplate{`⟨*begin of frame*⟩`}{`⟨*end of frame*⟩`}`

BEAMER This command is currently *not* available in the presentation modes.

ARTICLE The ⟨*begin of frame*⟩ text is inserted at the beginning of each frame, when it is inserted into the article. The text ⟨*end of frame*⟩ is inserted at the end. You can use this template to put, say, lines around a frame or to put the whole frame into a minipage. By default, nothing is inserted.

Example:

```
\useframetemplate{\par\medskip\hrule\smallskip}{\par\smallskip\hrule\medskip}
```

9.4.5 Background

Predefined Templates

`\beamertemplatesolidbackgroundcolor{`⟨*color*⟩`}`

Installs the given color as the background color for every frame.

Example: `\beamertemplatesolidbackgroundcolor{white!90!red}`

`\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[`⟨*spacing*⟩`]`

Installs a light grid as background with lines spaced apart by ⟨*spacing*⟩. Default is half a centimeter.

Example: `\beamertemplategridbackground[0.2cm]`

Template Installation Commands

`\usebackgroundtemplate{`⟨*background template*⟩`}`

Installs a new background template. Call `\beamersetaveragebackground` after you have called this macro, see Section 6.2 for details.

Example:

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

9.4.6 Table of Contents

Predefined Templates

`\beamertemplateplaintoc`

Installs a simple table of contents template with indented subsections.

`\beamertemplateballtoc`

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

`\beamertemplatenumberedsectiontoc`

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

`\beamertemplatenumberedcirclesectiontoc`

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

`\beamertemplatenumberedballsectiontoc`

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

`\beamertemplatenumberedsubsectiontoc`

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

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 these Templates

`\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.

9.4.7 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.

9.4.8 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.

`\beamertemplatecontinuationroman`

Causes the text in the frame title informing that the current frame has been broken up into several pages to be Roman numbers. Thus if you have a frame with title "Foo" that is broken up into two pages, the first page will have the title "Foo I" and the second will have "Foo II".

`\beamertemplatecontinuationtext`

Causes the text in the frame title informing that the current frame has been broken up into several pages to be the text `\insertcontinuationtext` on all pages but the first. This text inserted by this insert is "(cont.)" by default. Thus if you have a frame with title "Foo" that is broken up into two pages, the first page will have the title "Foo" and the second will have "Foo (cont.)" or "Foo (Forts.)" if you have redefined `\insertcontinuationtext` to "(Forts.)".

Template Installation Commands

`\useframetitletemplate{`⟨*frame title template*⟩`}`

Example:

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

ARTICLE This command is also available in `article` mode. By default, a new paragraph is created. You may wish to install a template that will simply suppress the frame title.

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

The ⟨*template*⟩ will be added at the end of the text inserted by the `\insertframetitle` text if a frame has the `allowframebreaks` option set.

Example:

```
\usecontinuationtemplate{ \ifnum\insertcontinuationcount>1(Forts.)\fi}
```

Inserts for these Templates

`\insertframetitle`

Inserts the current frame title into a template. If the current frame has the option `allowframebreaks` set, at the end of this insert the template that has been set using `\usecontinuationtemplate` will be appended.

`\insertframesubtitle`

Inserts the current frame subtitle into a template.

`\insertcontinuationtext`

Inserts the text "(cont.)" into a template. Redefine this insert if you use a different language. This insert is used by the template installed by the command `\beamertemplatecontinuationtext`.

`\insertcontinuationcount`

Inserts which page of the current frame is currently presented. If the `allowframebreaks` option is *not* set, this number is 0.

`\insertcontinuationcountroman`

Inserts which page of the current frame is currently presented as a Roman number.

9.4.9 Headlines and Footlines

Predefined Templates

`\beamertemplateheadempty`

Makes the headline empty.

`\beamertemplatefootempty`

Makes the footline empty.

`\beamertemplatefootpagenumber`

Shows only the page number in the footline.

Template Installation Commands

`\usefoottemplate{`⟨*footline template*⟩`}`

The final height of the footline is calculated by invoking this template just before the beginning of the document and by setting the footline 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{`⟨*headline 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 the bar to the left or right.

Example:

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

`\insertshortauthor[`⟨*options*⟩`]`

Inserts the short version of the author into a template. The text will be printed in one long line, line breaks introduced using the `\\` command are suppressed. The following ⟨*options*⟩ may be given:

- `width=`⟨*width*⟩ causes the text to be put into a multi-line minipage of the given size. Line breaks are still suppressed by default.
- `center` centers the text inside the minipage created using the `width` option, rather than having it left aligned.
- `respectlinebreaks` causes line breaks introduced by the `\\` command to be honored.

Example: `\insertauthor[width={3cm},center,respectlinebreaks]`

`\insertshortdate[`⟨*options*⟩`]`

Inserts the short version of the date into a template. The same options as for `\insertshortauthor` may be given.

`\insertshortinstitute[⟨options⟩]`

Inserts the short version of the institute into a template. The same options as for `\insertshortauthor` may be given.

`\insertshortpart[⟨options⟩]`

Inserts the short version of the part name into a template. The same options as for `\insertshortauthor` may be given.

`\insertshorttitle[⟨options⟩]`

Inserts the short version of the document title into a template. Same options as for `\insertshortauthor` may be given.

`\insertshortsubtitle[⟨options⟩]`

Inserts the short version of the document subtitle. Same options as for `\insertshortauthor` may be given.

`\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.

9.4.10 Sidebars

Sidebars are vertical areas that stretch from the lower end of the headline to the top of the footline. There can be a sidebar at the left and one at the right (or even both). Sidebars can show a table of contents, but they could also be added for purely aesthetic reasons.

When you install a sidebar template, you must explicitly specify the horizontal size of the sidebar. The vertical size is determined automatically. Each sidebar 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 sidebar 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 sidebar, you might write `\hskip-\beamer@leftmargin` to get there.

In the following, only the commands for the left sidebars are listed. Each of these commands also exists for the right sidebar, with “left” replaced by “right” everywhere.

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

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

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 sidebar.

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 sidebar.

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 sidebar.

Example:

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

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

Both parameters should be `\hbox`es. 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}
```

9.4.11 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.

9.4.12 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 sphere.

`\beamertemplatesphereminiframeinverted`

Changes the symbols in a navigation bar used to represent a frame to a small sphere, 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
  {\hfill\color{white}\tiny\textbf{\insertsectionheadnumber.\ \
    \insertsectionhead}}
  {\hfill\color{white!50!black}\tiny\textbf{\insertsectionheadnumber.\ \
    \insertsectionhead}}
```

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

See `\usesectionheadtemplate`.

Example:

```
\usesubsectionheadtemplate{\color{white}%
  \tiny\textbf{\insertsectionheadnumber.\insertsubsectionheadnumber\ \
  \insertsubsectionhead}}%
  {\color{white!50!beamerstructure}%
  \tiny\textbf{\insertsectionheadnumber.\insertsubsectionheadnumber\ \
  \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}
```

Inserts

`\insertsectionhead`

Inserts the text of the section that is to be typeset in a navigation bar.

`\insertsubsectionhead`

Inserts the text of the subsection that is to be typeset in a navigation bar.

`\insertsectionheadnumber`

Inserts the number of the section that is to be typeset in a navigation bar.

`\insertsubsectionheadnumber`

Inserts the number of the subsection that is to be typeset in a navigation bar.

`\insertpartheadnumber`

Inserts the number of the part of the current section of subsection that is to be typeset in a navigation bar.

9.4.13 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{\insertslidenavigationsymbol}
  \hbox{\insertframenavigationsymbol}
  \hbox{\insertsubsectionnavigationsymbol}
  \hbox{\insertsectionnavigationsymbol}
  \hbox{\insertdocnavigationsymbol}
  \hbox{\insertbackfindforwardnavigationsymbol}}}
```

Inserts for these Templates

The following inserts are useful for the navigation symbols template:

`\insertslidenavigationsymbol`

Inserts the slide navigation symbol, see Section 5.2.5.

`\insertframenavigationsymbol`

Inserts the frame navigation symbol, see Section 5.2.5.

`\insertsubsectionnavigationsymbol`

Inserts the subsection navigation symbol, see Section 5.2.5.

`\insertsectionnavigationsymbol`

Inserts the section navigation symbol, see Section 5.2.5.

`\insertdocnavigationsymbol`

Inserts the presentation navigation symbol and (if necessary) the appendix navigation symbol, see Section 5.2.5.

`\insertbackfindforwardnavigationsymbol`

Inserts a back, a find, and a forward navigation symbol, see Section 5.2.5.

9.4.14 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.

9.4.15 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.

9.4.16 Lists (Itemizations, Enumerations, Descriptions)

Predefined Templates

`\beamertemplateballitem`

Changes the symbols shown in an `itemize` and an `enumerate` environment to small plastic balls.

`\beamertemplatedotitem`

Changes the symbols shown in an `itemize` environment to dots.

`\beamertemplatetriangleitem`

Changes the symbols shown in an `itemize` environment to triangles.

`\beamertemplateenumeratealpha`

Changes the labels of first-level enumerations to “1.”, “2.”, “3.”, and so on, and to “1.1”, “1.2”, “1.3”, and so on for the second level.

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}}`

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

The ⟨*template*⟩ is used to render the default item in the third level of an itemize list.

Example: `\usesubsubitemizeitemtemplate{\pgfuseimage{mysubsubbullet}}`

`\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}}`

`\usesubsubitemizetemplate{`⟨*begin text*⟩`}{`⟨*end text*⟩`}`

The ⟨*begin text*⟩ is inserted at the beginning of a third-level itemize list, the ⟨*end text*⟩ at its end.

Example: `\usesubitemizetemplate{\begin{footnotesize}}{\end{footnotesize}}`

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

The ⟨*template*⟩ is used to render the default item in the top-level of an enumeration.

Example: `\useenumerateitemtemplate{\insertenumlabel}`

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

The ⟨*template*⟩ is used to render the items in an enumeration where the optional argument ⟨*mini template*⟩ is used (see Section 5.3.1).

Example: `\useenumerateitemminitemplate{\color{structure}\insertenumlabel}`

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

The ⟨*template*⟩ is used to render the default item in the second level of an enumeration.

Example: `\usesubenumerateitemtemplate{\insertenumlabel-\insertsubenumlabel}`

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

The ⟨*template*⟩ is used to render the default item in the third level of an enumeration.

Example:

```
\usesubsubenumerateitemtemplate
{\insertenumlabel-\insertsubenumlabel-\insertsubsubenumlabel}
```

`\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}}`

`\usesubsubenumeratetemplate{`⟨*begin text*⟩`}{`⟨*end text*⟩`}`

The ⟨*begin text*⟩ is inserted at the beginning of a third-level enumeration, the ⟨*end text*⟩ at its end.

Example: `\usesubsubenumeratetemplate{\begin{footnotesize}}{\end{footnotesize}}`

`\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`

Normally, this command inserts the current number of the top-level enumeration (as an Arabic number) into a template. However, in an enumeration where the optional ⟨*mini template*⟩ option is used, this command inserts the current number rendered by this mini template. For example, if the ⟨*mini template*⟩ is `(i)` and this command is used in the fourth item, `\insertenumlabel` would yield `(iv)`.

`\insertsubenumlabel`

Inserts the current number of the second-level enumeration (as an Arabic number) into a template.

`\insertsubsubenumlabel`

Inserts the current number of the third-level enumeration (as an Arabic number) into a template.

9.4.17 Hilighting Commands

Template Installation Commands

`\usealerttemplate{`⟨*alert template begin*⟩`}{`⟨*alert template end*⟩`}`

In an `\alert` command and in an `alertenv` environment, the text ⟨*alert template begin*⟩ is inserted at the beginning, the text ⟨*alert template end*⟩ at the end.

Example: `\usealerttemplate{\color{red}}{}`

ARTICLE This command is also available in `article` mode.

`\usestructuretemplate{`⟨*structure template begin*⟩`}{`⟨*structure template end*⟩`}`

Same as for alerts.

Example: `\usestructuretemplate{\color{blue}}{}`

ARTICLE This command is also available in `article` mode.

9.4.18 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}
```

ARTICLE This command is also available in `article` mode.

`\usealertblocktemplate`{⟨*block beginning template*⟩}{⟨*block end template*⟩}

Example:

```
\usealertblocktemplate
  {%
    \medskip
    {\alert{\textbf{\insertblockname}}}%
  \par}
  {\medskip}
```

ARTICLE This command is also available in `article` mode.

`\useexampleblocktemplate`{⟨*block beginning template*⟩}{⟨*block end template*⟩}

Example:

```
\useexampleblocktemplate
  {%
    \medskip
    \begingroup\color{darkgreen}{\textbf{\insertblockname}}
    \par}
  {%
     \endgroup
     \medskip
  }
```

ARTICLE This command is also available in `article` mode.

Inserts for these Templates

`\insertblockname`

Inserts the name of the current block into a template.

9.4.19 Theorem Environments

Predefined Templates

`\beamertemplatetheoremssimple`

Causes the theorem head and text to be directly passed to the `block` or `exampleblock` environment. All font specifications for theorems are ignored.

`\beamertemplatetheoremsunnumbered`

Causes theorems to be typeset as follows: The font specification for the body is honored, the font specification for the head is ignored. No theorem number is printed. This is the default.

`\beamertemplatetheoremsnumbered`

Like `\beamertemplatetheoremsunnumbered`, except that the theorem number is printed for environments that are numbered.

`\beamertemplatetheoremsamslike`

This causes theorems to be put in a `block` or `exampleblock`, but to be otherwise typeset as is normally done in `amsthm`. Thus the head font and body font depend on the setting for the theorem to be typeset and theorems are numbered.

Template Installation Commands

`\usetheoremtemplate{`⟨*block beginning template*⟩`}{`⟨*block end template*⟩`}`

BEAMER Whenever an environment declared using the command `\newtheorem` is to be typeset, the ⟨*block beginning template*⟩ is inserted at the beginning and the ⟨*block end template*⟩ at the end. If there is a overlay specification when an environment like `theorem` is used, this overlay specification will directly follow the ⟨*block beginning template*⟩ upon invocation. This is even true if there was an optional argument to the `theorem` environment. This optional argument is available via the insert `\inserttheoremaddition`.

Numerous inserts are available in this template, see below.

Before the template starts, the font is set to the body font prescribed by the environment to be typeset.

Example: The following typesets theorems like `amsthm`:

```
\usetheoremtemplate{\begin{\inserttheoremblockenv}
  {%
    \inserttheoremheadfont
    \inserttheoremname
    \inserttheoremnumber
    \ifx\inserttheoremaddition\@empty\else\ (\inserttheoremaddition)\fi%
    \inserttheorempunctuation
  }%
}{\end{\inserttheoremblockenv}}
```

Example: In the following example, all font “suggestions” for the environment are suppressed or ignored; and the theorem number is suppressed.

```
\usetheoremtemplate{%
  \normalfont% ignore body font
  \begin{\inserttheoremblockenv}
  {%
    \inserttheoremname
    \ifx\inserttheoremaddition\@empty\else\ (\inserttheoremaddition)\fi%
  }%
}{\end{\inserttheoremblockenv}}
```

ARTICLE This command is not available in `article` mode.

Inserts for these Templates

`\inserttheoremblockenv`

This will normally expand to `block`, but if a theorem that has theorem style `example` is typeset, it will expand to `exampleblock`. Thus you can use this insert to decide which environment should be used when typesetting the theorem.

`\inserttheoremheadfont`

This will expand to a font chainging command that switches to the font to be used in the head of the theorem. By not inserting it, you can ignore the head font.

`\inserttheoremname`

This will expand to the name of the environment to be typeset (like “Theorem” or “Corollary”).

`\inserttheoremnumber`

This will expand to the number of the current theorem preceeded by a space or to nothing, if the current theorem does not have a number.

`\inserttheoremaddition`

This will expand to the optional argument given to the environment or will be empty, if there was no optional argument.

`\inserttheorempunctuation`

This will expand to the punctuation character for the current environment. This is usually a period.

9.4.20 Verse, Quotation and Quote Environments

Template Installation Commands

`\usetemplateverse{`⟨*block beginning template*⟩`}{`⟨*block end template*⟩`}`

In a `verse` environment, the ⟨*block beginning template*⟩ is inserted before the verse, the ⟨*block end template*⟩ after the verse. The margins are not setup in these templates; this is done in the `verse` environment and cannot be changed.

Example: `\usetemplateverse{\rmfamily\itshape}{}`

`\usetemplatequotation{`⟨*block beginning template*⟩`}{`⟨*block end template*⟩`}`

Both in `quotation` and in `quote` environments, the ⟨*block beginning template*⟩ is inserted before the quotation, the ⟨*block end template*⟩ after the quotation. As for verses, the margins are not setup in these templates and cannot be changed.

Example: `\usetemplatequotation{\itshape}{}`

9.4.21 Typesetting Notes

Predefined Templates

`\beamertemplatenoteplain`

Causes all note pages to contain only the note text.

`\beamertemplatenotecompress`

Causes the “routing information” at the top of a note to be smaller.

Template Installation Commands

`\usetemplatenote{`⟨*note template*⟩`}`

Each note is typeset by inserting the ⟨*note template*⟩. The template should contain a mentioning of the insert `\insertnote`, which will contain the note text.

Example: `\usetemplatenote{\tiny\insertnote}`

Inserts for these Templates

`\insertnote`

Inserts the text of the current note into the template.

`\insertslideintonotes{`⟨*magnification*⟩`}`

Inserts a “mini picture” of the last slide into the current note. The slide will be scaled by the given magnification.

Example: `\insertslideintonotes{0.25}`

This will give a mini slide whose width and height are one fourth of the usual size.

10 Tips and (Dirty) Tricks

The aim of this section is to collect some hints and tricks that make use of the basic BEAMER class concepts.

10.1 Piecewise Uncovering

10.1.1 Uncovering an Enumeration Piecewise

A common usage of overlays is to show a list of points in an enumeration in a piecewise fashion. The easiest and most flexible way to do this is the following:

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

The advantage of this approach is that you retain total control over the order in which items are shown. By changing, for example, the last specification to `<2->`, you can have the last two points uncovered at the same time.

A disadvantage of the approach is that you will have to renumber everything if you add a new item. This is usually not such a big problem, but it can be a nuiseance.

To automatize the uncovering, you can use the following code:

```
\begin{itemize}[<+->]
\item First point.
\item Second point.
\item Third point.
\end{itemize}
```

The effect of the `[<+->]` is to install a *default overlay specification*, see the definition of `itemize` for details.

Now, suppose you wish the second and third point to be shown at the same time. You could achieve this by adding the specificaiton `<2->` to either the second or third `\item` command. However, then you still have to do some renumbering if you add a new item at the beginning. A better, though more cumbersome, approach is to decrease the counter `beamerpause` before the last item:

```
\begin{itemize}[<+->]
\item First point.
\item Second point.
  \addtocounter{beamerpause}{-1}
\item Third point.
\end{itemize}
```

This does not look so nice, but it works. Also, you might wish to build your own macros based on these ideas (like an `itemstep` environment or a `\itemlikeprevious` command).

10.1.2 Hilighting the Current Point in an Enumeration

If you uncover an enumeration piecewise, it is sometimes a good idea to hilight the last uncovered point to draw the audience's attention to it. This is best achieved as follows:

```
\begin{itemize}
\item<1-| alert@1> First point.
\item<2-| alert@2> Second point.
\item<3-| alert@3> Third point.
\end{itemize}
```

or

```
\begin{itemize}[<+-| alert@+>]
\item First point.
\item Second point.
\item Third point.
\end{itemize}
```

Note that this will draw the little item symbol also in red.

10.1.3 Changing Symbol Before an Enumeration

When uncovering a list of tasks or problems, you may desire that the symbol in front of the last uncovered symbol is, say, a ballot X, while for the previous items it is a check mark (you'll find these characters in some Dingbats fonts).

The best way to achieve this is to implement a new action environment. If this action is activated, it temporarily changes the item symbol template to the other symbol:

```
\newenvironment{ballotenv}
{\only{%
  \useitemizeitemtemplate{code for showing a ballot}%
  \usesubitemizeitemtemplate{code for showing a smaller ballot}%
  \usesubsubitemizeitemtemplate{code for showing a smaller ballot}}}
{}

\useitemizeitemtemplate{code for showing a check mark}
\usesubitemizeitemtemplate{code for showing a smaller check mark}
\usesubsubitemizeitemtemplate{code for showing a smaller check mark}
```

The effect of the code is to install a check mark as the default template. If the action `ballot` is now requested for some item, this template will temporarily be replaced by the ballot templates.

Note that the `ballotenv` is invoked with the overlay specification given for the action directly following it. This causes the `\only` to be invoked exactly for the specified overlays.

Here are example usages:

```
\begin{itemize}
\item<1-| ballot@1> First point.
\item<2-| ballot@2> Second point.
\item<3-| ballot@3> Third point.
\end{itemize}
```

and

```
\begin{itemize}[<+-| ballot@+>]
\item First point.
\item Second point.
\item Third point.
\end{itemize}
```

In the following example, more and more items become "checked" from slide to slide:

```
\begin{itemize}[<ballot@+-| visible@1-,+(1)>]
\item First point.
\item Second point.
\item Third point.
\end{itemize}
```

The important point is `ballot@+`. The funny `visible@1-,+(1)` has the following effect: Although it has no effect with respect to what is shown (after all, it applies to all slides), it ensures that in the enumeration the slide number 4 is mentioned. Thus there will also be a slide in which all three points are checked.

10.1.4 Uncovering Tagged Formulas Piecewise

Suppose you have a three-line formula as the following:

```
\begin{align}
  A &= B \\
    &= C \\
    &= D
\end{align}
```

Uncovering this formula line-by-line is a little tricky. A first idea is to use the `\pause` or `\onslide` commands. Unfortunately, these do not work since `align` internally reprocesses its input several times, which messes up the delicate internals of the commands. The next idea is the following, which works a little better:

```
\begin{align}
  A &= B \\
    \uncover<2->{&= C \\}
    \uncover<3->{&= D}
\end{align}
```

Unfortunately, this does not work in the presence of tags (so it works for the `align*` environment). What happens is that the tag of the last line is shown on all slides. The problem here is that the tag is created when `\\` is encountered or when `\end{align}` is encountered. In the last line these are already "behind" the `\uncover`.

To solve this problem, you can add an empty line without a tag and then insert a negative vertical skip to undo the last line:

```
\begin{align}
  A &= B \\
    \uncover<2->{&= C \\}
    \uncover<3->{&= D \\}
    \notag
  \end{align}
\vskip-1.5em
```

10.1.5 Uncovering a Table Linewise

When you wish to uncover a table line-by-line, you will run into all sorts of problems if there are vertical and horizontal lines in the table. The reason is that the first vertical line at the left end is drawn before the line is even read (and thus, in particular, before any `\onslide` command can be read). However, placing a `\pause` or `\uncover` at the end of the line before is also not helpful since it will then suppress the horizontal line below the last uncovered line.

A possible way to solve this problem is not to use either horizontal or vertical lines. Instead, colouring the lines using the `colortbl` package is a good alternative to structure the table. Here is an optically pleasing example, where the table is uncovered line-wise:

```
\rowcolors[]{1}{blue!20}{blue!10}
\begin{tabular}{l!{\vrule}cccc}
  Class & A & B & C & D \\\hline
  X     & 1 & 2 & 3 & 4 \pause\\
  Y     & 3 & 4 & 5 & 6 \pause\\
  Z     & 5 & 6 & 7 & 8
\end{tabular}
```

By using `\onslide` instead of `\pause`, you can get more fine-grained control over which line is shown on which slide.

10.1.6 Uncovering a Table Columnwise

The same problems as for uncovering a table linewise arise for uncovering it columnwise.

Once more, using the `colortbl` package offers a solution. In the following example, the `tabular` header is used to insert `\onslide` commands, one for each column, that cover the entries in the column from a certain slide on. At the end of the last column, the `\onslide` without a specification ensures that the first column on the next row is once more shown normally.

Inserting a horizontal line is tricky since it will protrude over the full width of the table also in the covered version. The best idea is just not to use horizontal bars.

```
\rowcolors[]{1}{blue!20}{blue!10}
\begin{tabular}{l!{\vrule}c<{\onslide<2->}c<{\onslide<3->}c<{\onslide<4->}c<{\onslide}c}
  Class & A & B & C & D \\
  X     & 1 & 2 & 3 & 4 \\
  Y     & 3 & 4 & 5 & 6 \\
  Z     & 5 & 6 & 7 & 8
\end{tabular}
```

11 Compatibility and Emulation

11.1 Compatibility with Other Packages and Classes

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

`\usepackage{amsthm}`

This package is automatically loaded since BEAMER uses it for typesetting theorems. If you do not wish it to be loaded, which can be necessary especially in `article` mode if the package is incompatible with the document class, you can use the class option `noamsthm` to suppress its loading. See Section 5.3.4 for more details.

`\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. Also, the characters `:` and `!` will not be a active characters. This means, that the little space that is inserted before them in the `french` style is not inserted. You have to do this "by hand."

ARTICLE To make the characters `:` and `!` active in `article` mode, pass the option `activespeccharacters` to the package `beamerbasearticle`. However, this may lead to problems with overlay specifications.

`\usepackage[spanish]{babel}`

BEAMER 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.

ARTICLE To make the characters `<` and `>` active in `article` mode, pass the option `activespeccharacters` to the package `beamerbasearticle`. As for the `french` package, this may lead to problems with overlay specifications.

`\usepackage{color}`

BEAMER The `color` package is automatically loaded by `beamer.cls`. This makes it impossible to pass options to `color` in the preamble of your document. To pass a ⟨*list of options*⟩ to `color`, you must use the following class option:

`\documentclass[color=`⟨*list of options*⟩`]{beamer}`

Causes the ⟨*list of options*⟩ to be passed on to the `color` package. If the ⟨*list of options*⟩ contains more than one option you must enclose it in curly brackets.

ARTICLE The `color` package is not loaded automatically if `beamerbasearticle` is loaded with the `noxcolor` option.

`\usepackage{CJK}`

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

`\usepackage{deluxetable}`

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

`\usepackage{enumerate}`

ARTICLE This package is loaded automatically in the `presentation` modes, but not in the `article` mode. If you use its features, you have to load the package "by hand" in the `article` mode.

`\documentclass{foils}`

If you wish to emulate the `foils` class using BEAMER, please see Section 11.2.3.

`\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 originally 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 9.1.5.

`\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{HA-prosper}`

You cannot use this package with BEAMER. However, you might try to use the package `beamerprosper` instead, see Section 11.2.1.

`\usepackage{hyperref}`

BEAMER The `hyperref` package is automatically loaded by `beamer.cls` and certain options are setup. In order pass additional options to `hyperref` or to override options, you can use the following class option:

`\documentclass[hyperref=`⟨*list of options*⟩`]{beamer}`

Causes the ⟨*list of options*⟩ to be passed on to the `hyperref` package.

Example: `\documentclass[hyperref={bookmarks=false}]{beamer}`

Alternatively, you can also use the `\hypersetup` command.

ARTICLE In the `article` version, you must include `hyperref` manually if you want to use it. It is not included automatically.

`\usepackage[utf8]{inputenc}`

BEAMER When using Unicode, you may wish to use one of the following class options:

`\documentclass[ucs]{beamer}`

Loads the package `ucs` and passes the correct Unicode options to `hyperref`. Also, it preloads the Unicode code pages zero and one.

`\documentclass[utf8]{beamer}`

Same as the option `ucs`, but also sets the input encoding to `utf8`. You could also use the option `ucs` and say `\usepackage[utf8]{inputenc}` in the preamble.

If you use a Unicode character outside the first two code pages (which includes the Latin alphabet and the extended Latin alphabet) in a section or subsection heading, you have to use the command `\PreloadUnicodePage{`⟨*code page*⟩`}` to give `ucs` a chance to preload these code pages. You will know that a character has not been preloaded, if you get a message like "Please insert into preamble." The code page of a character is given by the unicode number of the character divided by 256.

`\usepackage{listings}`

BEAMER Note that you must treat `lstlisting` environments exactly the same way as you would treat `verbatim` environments. When using `\defverbatim` that contains a colored `lstlisting`, use the `colored` option of `\defverbatim`.

`\usepackage{`⟨*professional font package*⟩`}`

BEAMER If you use a professional font package, BEAMER's internal redefinition of how variables are typeset may interfere with the font package's superior way of typesetting them. In this case, you should use the class option `professionalfont` to suppress any font substitution. See Section 9.1.3 for details.

`\documentclass{prosper}`

If you wish to (partly) emulate the `prosper` class using BEAMER, please see Section 11.2.1.

`\usepackage{pstricks}`

You should add the option `xcolor=pst` to make `xcolor` aware of the fact that you are using `pstricks`.

`\documentclass{seminar}`

If you wish to emulate the `seminar` class using BEAMER, please see Section 11.2.2.

`\usepackage{texpower}`

You cannot use this package with BEAMER. However, you might try to use the package `beamertexpower` instead, see Section 11.2.4.

`\usepackage{textpos}`

BEAMER BEAMER automatically installs a white background behind everything, unless you install a different background template. Because of this, you must use the `overlay` option when using `textpos`, so that it will place boxes *before* everything. Alternatively, you can install an empty background template, but this may result in an incorrect display in certain situtations with older versions of the Acrobat Reader.

`\usepackage{ucs}`

See `\usepackage[utf8]{inputenc}`.

`\usepackage{xcolor}`

BEAMER The `xcolor` package is automatically loaded by `beamer.cls`. The same applies as to `color`.

`\documentclass[xcolor=`⟨*list of options*⟩`]{beamer}`

Causes the ⟨*list of options*⟩ to be passed on to the `xcolor` package.

When using BEAMER together with the `pstricks` package, be sure to pass the `xcolor=pst` option to BEAMER (and hence to `xcolor`).

ARTICLE The `color` package is not loaded automatically if `beamerbasearticle` is loaded with the `noxcolor` option.

11.2 Emulation of other Classes

The BEAMER class comes with a number of emulation layer for classes or packages that do not support BEAMER directly. For example, the package `beamerseminar` maps some (not all) commands of the SEMINAR class to appropriate BEAMER commands. This way, individual slides or whole sets of slides that have been prepared for a presentation using SEMINAR can be used inside BEAMER, provided they are reasonably simple.

None of the emulation layers is a perfect substitute for the original (emulations seldom are) and it is not intended that they ever will be. If you want/need/prefer the features of another class, use that class for preparing your presentations. The intension of these layers is just to help speed up creating BEAMER presentations that use parts of old presentations. You can simply copy these parts in verbatim, without having to worry about the subtle differences in syntax.

A useful effect of using an emulation layer is that you get access to all the features of BEAMER while using the syntax of another class. For example, you can use the `article` mode to create a nice article version of a PROSPER talk.

11.2.1 Prosper and HA-Prosper

The package `beamerprosper` maps the commands of the PROSPER package, developped by Frédéric Goualard, to BEAMER commands. Also, some commands of the HA-PROSPER package, developped by Hendri Adriaens, are mapped to BEAMER commands. *These mappings cannot perfectly emulate all of Prosper!* Rather, these mappings are intended as an aid when porting parts of presentations created using PROSPER to BEAMER. *No styles are implemented that mimick Prosper styles.* Rather, the normal BEAMER themes must be used (although, one could implement BEAMER themes that mimicks existing PROSPER styles; I have not done that and do not intend to).

The workflow for creating a BEAMER presentation that uses PROSPER code is the following:

1. Use the document class `beamer`, not `prosper`. Most options passed to `prosper` do not apply to `beamer` and should be omitted.
2. Add a `\usepackage{beamerprosper}` to start the emulation.
3. If you add slides relying on HA-PROSPER, you may wish to add the option `framesassubsections` to `beamerprosper`, though I do not recommend it (use the normal `\subsection` command instead; it gives you more fine-grained control).
4. If you also copy the title commands, it may be necessary to adjust the content of commands like `\title` or `\author`. Note that in PROSPER the `\email` command is given outside the `\author` command, whereas in BEAMER and also in HA-PROSPER it is given inside.
5. When copying slides containing the command `\includegraphics`, you will almost surely have to adjust its usage. If you use pdfLaTeX to typeset the presentation, than you cannot include PostScript file. You should convert them to `.pdf` or to `.png` and adjust any usage of `\includegraphics` accordingly.
6. When starting to change things, you can use all of BEAMER's commands and even mix them with PROSPER commands.

An example can be found in the file `beamerexample-prosper.tex`.

There are, unfortunately, quite a few places where you may run into problems:

- In BEAMER, the command `\PDForPS` will do exactly what the name suggests: insert the first argument when run by `pdflatex`, insert the second argument when run by `latex`. However, in PROSPER, the code inserted for the PDF case is actually PostScript code, which is only later converted to PDF by some external program. You will need to adjust this PostScript code such that it works with `pdflatex` (which is not always possible).
- If you used fine-grained spacing commands, like adding a little horizontal skip here and a big negative vertical skip there, the typesetting of the text may be poor. It may be a good idea to just remove these spacing commands.

- If you use pstricks commands, you will either have to stick to using latex and dvips or will have to work around them using, for example, pgf. Porting lot's of pstricks code is bound to be difficult, if you wish to switch over to pdflatex, so be warned.
- If the file cannot be compiled because some PROSPER command is not implemented, you will have to delete this command and try to mimick its behaviour using some BEAMER command.

\usepackage{beamerprosper}

Include this package in a beamer presentation to get access to PROSPER commands. Use beamer as the document class, not prosper. Most of the options passed to the class prosper make no sense in beamer, so just delete them.

This package takes the following options:

- framesassubsections causes each frame to create its own subsection with the frame title as subsection name. This behaviour mimicks HA-PROSPER's behaviour. In a long talk this will create way too many subsections.

ARTICLE The framesassubsections option has no effect in article mode.

Example:

```
\documentclass[notes]{beamer}

\usepackage[framesassubsections]{beamerprosper}
\mode<presentation>
{
  \definecolor{beamerstructure}{RGB}{43,79,112}
  \definecolor{sidebackground}{RGB}{230,242,250}
  \color{beamerstructure}
  \usepackage[tab,width=3.25cm]{beamerthemesidebar}
  \usepackage{times}
  \userightsidebarcolortemplate{\color{sidebackground}}
  \beamertemplateballitem
  \beamertemplateboldframetitle
  \beamertemplateboldtitlepage
}

\title{A Beamer Presentation Using (HA-)Prosper Commands}
\subtitle{Subtitles Are Also Supported}
\author{Till Tantau}
\institution{The Institution is Mapped To Institute}

\begin{document}

\maketitle

\tsectionandpart{Introduction}

\overlays{2}{
\begin{slide}{About this file}
  \begin{itemstep}
  \item
    This is a beamer presentation.
  \item
    You can use the prosper and the HA-prosper syntax.
  \item
    This is done by mapping prosper and HA-prosper commands to beamer
    commands.
  \item
    The emulation is by no means perfect.
  \end{itemstep}
\end{slide}
}

\section{Second Section}
\subsection{A subsection}
\frame
{
  \frametitle{A frame created using the \texttt{frame} command.}

  \begin{itemize}[<+->]
  \item You can still use the original beamer syntax.
  \item The emulation is intended only to make recycling slides
    easier, not to install a whole new syntax for beamer.
  \end{itemize}
}

\begin{notes}{Notes for these slides}
My notes for these slides.
\end{notes}
\end{document}
```

You can run, for example, pdfLaTeX on the file to get a BEAMER presentation with overlays. Adding the `notes` option will also show the note. Certain commands, like `\LeftFoot`, are ignored. You can change the theme using the usual commands. You can also use all normal BEAMER commands and concepts, like overlay-specifications, in the file. You can also create an `article` version by adding the class option `class=article` and including the package `beamerbasearticle`.

In the following, the effect of PROSPER commands in BEAMER are listed.

`\email{`⟨*text*⟩`}`

Simply typesets its argument in typewriter text. Should hence be given *inside* the `\author` command.

`\institution{`⟨*text*⟩`}`

This command is mapped to BEAMER's `\institute` command if given *outside* the `\author` command, otherwise it typesets its argument in a smaller font.

`\Logo(`⟨*x*⟩`,`⟨*y*⟩`){`⟨*logo text*⟩`}`

This is mapped to `\logo{`⟨*logo text*⟩`}`. The coordinates are ignored.

`\begin{slides}[`⟨*options*⟩`]{`⟨*frame title*⟩`}`
⟨*environment contents*⟩
`\end{slides}`

Inserts a frame with the `containsverbatim` option set. The ⟨*frame title*⟩ will be enclosed in a `\frametitle` command.

The following ⟨*options*⟩ may be given:

- `trans=`⟨*prosper transition*⟩ installs the specified ⟨*prosper transition*⟩ as the transition effect when showing the slide.
- ⟨*prosper transition*⟩ has the same effect as `trans=`⟨*prosper transition*⟩.
- `toc=`⟨*entry*⟩ overrides the subsection table of contents entry created by this slide by ⟨*entry*⟩. Note that a subsection entry is created for a slide only if the `framesassubsections` options is specified.
- `template=`⟨*text*⟩ is ignored.

Example: The following two texts have the same effect:

```
\begin{slide}[trans=Glitter,toc=short]{A Title}
  Hi!
\end{slide}
```

and

```
\subsection{short} % omitted, if framesassubsections is not specified
\frame[containsverbatim]
{
  \transglitter
  \frametitle{A Title}
  Hi!
}
```

`\overlays{`⟨*number*⟩`}{`⟨*slide environment*⟩`}`

This will put the ⟨*slide environment*⟩ into a frame that does not have the `containsverbatim` option and which can hence contain overlayed text. The ⟨*number*⟩ is ignored since the number of necessary overlays is computed automatically by BEAMER.

Example: The following code fragments have the same effect:

```
\overlays{2}{
\begin{slide}{A Title}
  \begin{itemstep}
  \item Hi!
  \item Ho!
  \end{itemstep}
\end{slide}}
```

and

```
\subsection{A Title} % omitted, if framesassubsections is not specified
\frame
{
  \frametitle{A Title}
  \begin{itemstep}
  \item Hi!
  \item Ho!
  \end{itemstep}
}
```

`\fromSlide{`⟨*slide number*⟩`}{`⟨*text*⟩`}`

This is mapped to `\uncover<`⟨*slide number*⟩`->{`⟨*text*⟩`}`.

`\fromSlide*{`⟨*slide number*⟩`}{`⟨*text*⟩`}`

This is mapped to `\only<`⟨*slide number*⟩`->{`⟨*text*⟩`}`.

`\onlySlide{`⟨*slide number*⟩`}{`⟨*text*⟩`}`

This is mapped to `\uncover<`⟨*slide number*⟩`>{`⟨*text*⟩`}`.

`\onlySlide*{`⟨*slide number*⟩`}{`⟨*text*⟩`}`

This is mapped to `\only<`⟨*slide number*⟩`>{`⟨*text*⟩`}`.

`\untilSlide{`⟨*slide number*⟩`}{`⟨*text*⟩`}`

This is mapped to `\uncover<-`⟨*slide number*⟩`>{`⟨*text*⟩`}`.

`\untilsSlide*{`⟨*slide number*⟩`}{`⟨*text*⟩`}`

This is mapped to `\only<-`⟨*slide number*⟩`>{`⟨*text*⟩`}`.

`\FromSlide{`⟨*slide number*⟩`}`

This is mapped to `\onslide<`⟨*slide number*⟩`->`.

`\OnlySlide{`⟨*slide number*⟩`}`

This is mapped to `\onslide<`⟨*slide number*⟩`>`.

`\UntilSlide{⟨slide number⟩}`

This is mapped to `\onslide<-`⟨*slide number*⟩`>`.

`\slideCaption{⟨text⟩}`

This is mapped to `\date{`⟨*text*⟩`}`.

`\fontTitle{⟨text⟩}`

Simply inserts ⟨*text*⟩.

`\fontText{⟨text⟩}`

Simply inserts ⟨*text*⟩.

`\PDFtransition{⟨prosper transition⟩}`

Maps the ⟨*prosper transition*⟩ to an appropriate `\transxxxx` command.

```
\begin{Itemize}
⟨environment contents⟩
\end{Itemize}
```

This is mapped to `itemize`.

```
\begin{itemstep}
⟨environment contents⟩
\end{itemstep}
```

This is mapped to `itemize` with the option `[<+->]`.

```
\begin{enumstep}
⟨environment contents⟩
\end{enumstep}
```

This is mapped to `enumerate` with the option `[<+->]`.

`\hiddenitem`

This is mapped to `\addtocounter{beamerpauses}{1}`.

`\prosperpart[⟨options⟩]{⟨text⟩}`

This command has the same effect as PROSPER's `\part` command. BEAMER's normal `\part` command retains its normal sematics. Thus, you might wish to replace all occurences of `\part` by `\prosperpart`.

`\tsection*{⟨section name⟩}`

Creates a section named ⟨*section name*⟩. The star, if present, is ignored.

`\tsectionandpart*{⟨part text⟩}`

Mapped to a `\section` command followed by a `\prosperpart` command.

ARTICLE In `article` mode, no part page is added.

`\dualslide[⟨x⟩][⟨y⟩][⟨z⟩]{⟨options⟩}{⟨left column⟩}{⟨right column⟩}`

This command is mapped to a `columns` environment. The ⟨*left column*⟩ text is shown in the left column, the ⟨*right column*⟩ text is shown in the right column. The options ⟨*x*⟩, ⟨*y*⟩, and ⟨*z*⟩ are ignored. Also, all *options* are ignored, except for `lcolwidth=` and `rcolwidth=`. These set the width of the left or right column, respectively.

`\PDForPS{⟨PostScript text⟩}{⟨PDF text⟩}`

Inserts either the ⟨*PostScript text*⟩ or the ⟨*PDF text*⟩, depending on whether `latex` or `pdflatex` is used. When porting, the ⟨*PDF text*⟩ will most likely be *incorrect*, since in PROSPER the ⟨*PDF text*⟩ is actually PostScript text that is later transformed to PDF by some external program.

If the ⟨*PDF text*⟩ contains an `\includegraphics` command (which is its usual use), you should change the name of the graphic file that is included to a name ending `.pdf`, `.png`, or `.jpg`. Typically, you will have to convert your graphic to this format.

`\onlyInPDF`⟨*PDF text*⟩

The ⟨*PDF text*⟩ is only included if `pdflatex` is used. The same as for the command `\PDForPS` applies here.

`\onlyInPS`⟨*PS text*⟩

The ⟨*PS text*⟩ is only included if `latex` is used.

`\begin{notes}{`⟨*title*⟩`}`
⟨*environment contents*⟩
`\end{notes}`

Mapped to `\note{\textbf{`⟨*title*⟩`}`⟨*environment contents*⟩`}` (more or less).

The following commands are parsed by BEAMER, but have no effect:

- `\myitem`,
- `\FontTitle`,
- `\FontText`,
- `\ColorFoot`,
- `\DefaultTransition`,
- `\NoFrenchBabelItemize`,
- `\TitleSlideNav`,
- `\NormalSlideNav`,
- `\HAPsetup`,
- `\LeftFoot`, and
- `\RightFoot`.

11.2.2 Seminar

The package `beamerseminar` maps a subset of the commands of the SEMINAR package to BEAMER. As for PROSPER, the emulation cannot be perfect. For example, no portrait slides are supported, no automatic page breaking, the framing of slides is not emulated. Unfortunately, for all frames (`slide` environments) that contain overlays, you have to put the environment into a `\frame` "by hand" and must remove all occurences of `\newslide` inside the environment by closing the slide and opening a new one (and them putting these into `\frame` commands).

The workflow for the migration is the following:

1. Use the document class `beamer`, not `seminar`. Most options passed to `seminar` do not apply to `beamer` and should be omitted.
2. If you copy parts of a presentation that is mixed with normal text, add the `ignorenonframetext` option and place *every* `slide` environment inside a `\frame` since BEAMER will not recognize the `\begin{slide}` as the beginning of a frame.
3. Add a `\usepackage{beamerseminar}` to start the emulation. Add the option `accumulate` if you wish to create a presentation to be held with a video projector.
4. Possibly add commands to install themes and templates.
5. The should be not commands in the preamble having to do with page and slide styles. They do not apply to `beamer`.
6. If a `\newslide` command is used in a `slide` (or similarly `slide*`) environment that contains an overlay, you must replace it by a closing `\end{slide}` and an opening `\begin{slide}`.
7. Next, for each `slide` or `slide*` environment that contains an overlay, you must place a `\frame` command around it. You can remove the environment, unless you use the `accumulate` option.
8. If you use `\section` or `\subsection` commands inside slides, you will have to move them *outside* the frames. It may then be necessary to add a `\frametitle` command to the slide.

9. If you use pdfLaTeX to typeset the presentation, you cannot include PostScript files. You should convert them to `.pdf` or to `.png` and adjust any usage of `\includegraphics` accordingly.
10. When starting to change things, you can use all of BEAMER's commands and even mix them with SEMINAR commands.

An example can be found in the file `beamerexample-seminar.tex`.
There are, unfortunately, numerous places where you may run into problems:

- The whole `note` management of `seminar` is so different from `beamer`'s, that you will have to edit notes "by hand." In particular, commands like `\ifslidesonly` and `\ifslide` may not do exactly what you expect.
- If you use `pstricks` commands, you will either have to stick to using `latex` and `dvips` or will have to work around them using, for example, `pgf`. Porting lot's of `pstricks` code is bound to be difficult, if you wish to switch over to `pdflatex`, so be warned.
- If the file cannot be compiled because some SEMINAR command is not implemented, you will have to delete this command and try to mimick its behaviour using some BEAMER command.

`\usepackage{beamerseminar}`

Include this package in a `beamer` presentation to get access to SEMINAR commands. Use `beamer` as the document class, not `seminar`. Most of the options passed to the class `seminar` make no sense in `beamer`, so just delete them.

This package takes the following options:

- `accumulate` causes overlays to be accumulated. The original behaviour of the SEMINAR package is that in each overlay only the really "new" part of the overlay is shown. This makes sense, if you really print out the overlays on transparencies and then really stack overlays on top of each other. For a presentation with a video projector, you rather want to present an "accumulated" version of the overlays. This is what this option does: When the new material of the ith overlay is shown, the material of all previous overlays is also shown.

Example: The following example is an extract of `beamerexample-seminar.tex`:

```
\documentclass[ignorenonframetext]{beamer}
\usepackage[accumulated]{beamerseminar}
\usepackage{beamerthemeclassic}

\title{A beamer presentation using seminar commands}
\author{Till Tantau}

\let\heading=\frametitle

\begin{document}

\frame{
\maketitle
}

This is some text outside any frame. It will only be shown in the
article version.

\frame
{
  \begin{slide}
    \heading{This is a frame title.}

    \begin{enumerate}
      {\overlay1
      \item Overlays are a little tricky in seminar.
```

```
        {\overlay2
        \item But it is possible to use them in beamer.
        }
      }
    \end{enumerate}
\end{slide}
}
\end{document}
```

You can use all normal BEAMER commands and concepts, like overlay-specifications, in the file. You can also create an `article` version by adding the class option `class=article` and including the package `beamerbasearticle`.

In the following, the effect of SEMINAR commands in BEAMER are listed.

`\overlay{`⟨*number*⟩`}`

Shows the material till the end of the current TEX group only on overlay numbered ⟨*number*⟩ + 1 or, if the `accumulate` option is given, from that overlay on. Usages of this command may be nested (as in SEMINAR). If an `\overlay` command is given inside another, it temporarily "overrules" the outer one as demonstrated in the following example, where it is assumed that the `accumulate` option is given.

Example:

```
\frame{
\begin{slide}
  This is shown from the first slide on.
  {\overlay{2}
    This is shown from the third slide on.
    {\overlay{1}
      This is shown from the second slide on.
    }
    This is shown once more from the third slide on.
  }
\end{slide}
}
```

`\begin{slide*}`
⟨*environment contents*⟩
`\end{slide*}`

Mainly installs an `\overlay{0}` around the ⟨*environment contents*⟩. If the `accumulate` option is given, this has no effect, but otherwise it will cause the main text of the slide to be shown *only* on the first slide. This is useful if you really wish to physically place slides on top of each other.

The starred version does the same as the nonstarred one.

If this command is not issued inside a `\frame`, it sets up a frame with the `containsverbatim` option set. Thus, this frame will contain only a single slide.

Example:

```
\begin{slide}
  Some text.
\end{slide}

\frame{
\begin{slide}
  Some text. And an {\overlay{1} overlay}.
\end{slide}
}
```

`\red`

Mapped to `\color{red}`.

`\blue`

Mapped to `\color{blue}`.

`\green`

Mapped to `\color{green}`.

`\ifslide`

True in the `presentation` modes, false in the `article` mode.

`\ifslidesonly`

Same as `\ifslide`.

`\ifarticle`

False in the `presentation` modes, true in the `article` mode.

`\ifportrait`

Always false.

The following commands are parsed by BEAMER, but have no effect:

- `\ptsize`.

11.2.3 FoilTEX

The package `beamerfoils` maps a subset of the commands of the FOILS package to BEAMER. Since this package defines only few non-standard TEX commands and since BEAMER implements all the standard commands, the emulation layer is pretty simple.

A copyright notice: The FoilTEX package has a restricted license. For this reason, no example from the FOILS package is included in the BEAMER class. The emulation itself does not use the code of the FOILS package (rather, it just maps FOILS commands to BEAMER commands). For this reason, my understanding is that the *emulation* offered by the BEAMER class is "free" and legally so. IBM has a copyright on the FOILS class, not on the effect the commands of this class have. (At least, that's my understanding of things.)

The workflow for the migration is the following:

1. Use the document class `beamer`, not `foils`.
2. Add a `\usepackage{beamerfoils}` to start the emulation.
3. Possibly add commands to install themes and templates.
4. If the command `\foilhead` is used inside a `\frame` command, it behaves like `\frametitle`. If it used outside a frame, it will start a new frame (with the `allowframebreaks` option, thus no overlays are allowed). This frame will persist till the next occurence of `\foilhead` or of the new command `\endfoil`. Note that a `\frame` command will *not* end a frame started using `\foilhead`.
5. If you rely on automatic frame creation based on `\foilhead`, you will need to insert an `\endfoil` before the end of the document to end the last frame.
6. If you use pdfLATEX to typeset the presentation, than you cannot include PostScript files. You should convert them to `.pdf` or to `.png` and adjust any usage of `\includegraphics` accordingly.
7. Sizes of objects are different in BEAMER, since the scaling is done by the viewer, not by the class. Thus a framebox of size 6 inches will be way too big in a BEAMER presentation. You will have to manually adjust explicit dimension occuring in a foilTEX presentation.

`\usepackage{beamerfoils}`

Include this package in a `beamer` presentation to get access to FOILS commands. Use `beamer` as the document class, not `foils`.

Example: In the following example, frames are automatically created. The `\endfoil` at the end is needed to close the last frame.

```
\documentclass{beamer}
\usepackage{beamerfoils}

\begin{document}

\maketitle

\foilhead{First Frame}

This is on the first frame.
\pagebreak
This is on the second frame, which is a continuation of the first.

\foilhead{Third Frame}

This is on the third frame.

\endfoil
\end{document}
```

Example: In this example, frames are manually inserted. No `\endfoil` is needed.

```
\documentclass{beamer}
\usepackage{beamerfoils}

\begin{document}

\frame{\maketitle}

\frame{
\foilhead{First Frame}
This is on the first frame.
}

\frame{
\foilhead{Second Frame}
This is on the second frame.
}
\end{document}
```

In the following, the effect of FOILS commands in BEAMER are listed.

`\MyLogo{`⟨*logo text*⟩`}`

This is mapped to `\logo`, though the logo is internally stored, such that it can be switched on and off using `\LogoOn` and `\LogoOff`.

`\LogoOn`

Makes the logo visible.

`\LogoOff`

Makes the logo invisible.

`\foilhead[`⟨*dimension*⟩`]{`⟨*frame title*⟩`}`

If used inside a `\frame` command, this is mapped to `\frametitle{`⟨*frame title*⟩`}`. If used outside any frames, a new frame is started with the option `allowframebreaks`. If a frame was previously started using this command, it will be closed before the next frame is started. The ⟨*dimension*⟩ is ignored.

`\rotatefoilhead[⟨dimension⟩]{⟨frame title⟩}`

This command has exactly the same effect as `\foilhead`.

`\endfoil`

This is a command that is *not* available in FOILS. In BEAMER, it can be used to end a frame that has automatically been opened using `\foildhead`. This command must be given before the end of the document if the last frame was opened using `\foildhead`.

```
\begin{boldequation*}
⟨environment contents⟩
\end{boldequation*}
```

This is mapped to the `equation` or the `equation*` environment, with `\boldmath` switched on.

`\FoilTeX`

Typests the foilTEX name as in the FOILS package.

`\bm{⟨text⟩}`

Implemented as in the FOILS package.

`\bmstyle{⟨text⟩}{⟨more text⟩}`

Implemented as in the FOILS package.

The following additional theorem-like environments are predefined:

- `Theorem*`,
- `Lemma*`,
- `Corollary*`,
- `Proposition*`,
- `Definition*`.

For example, the first is defined using `\newtheorem*{Theorem*}{Theorem}`.

The following commands are parsed by BEAMER, but have not effect:

- `\leftheader`,
- `\rightheader`,
- `\leftfooter`,
- `\rightfooter`,
- `\Restriction`, and
- `\marginpar`.

11.2.4 TEXPower

The package `beamertexpower` maps a subset of the commands of the TEXPOWER package, due to Stephan Lehmke, to BEAMER. This subset is currently rather small, so a lot of adaptions may be necessary. Note that TEXPOWER is not a full class by itself, but a package that needs another class, like `seminar` or `prosper` to do the actualy typesetting. It may thus be necessary to additionally load an emulation layer for these also. Indeed, it *might* be possible to directly use TEXPOWER inside BEAMER, but I have not tried that. Perhaps this will be possible in the future.

Currently, the package `beamertexpower` mostly just maps the `\stepwise` and related commands to appropriate BEAMER commands. The `\pause` command need not be mapped since it is directly implemented by BEAMER anyway.

The workflow for the migration is the following:

1. Replace the document class by `beamer`. If the document class is `seminar` or `prosper`, you can use the above emulation layers, that is, you can include the files `beamerseminar` or `beamerprosper` to emulate the class.

   All notes on what to do for the emulation of SEMINAR or PROSPER also apply here.

2. Additionally, add `\usepackage{beamertexpower}` to start the emulation.

`\usepackage{beamertexpower}`

Include this package in a `beamer` presentation to get access to the TEXPOWER commands having to do with the `\stepwise` command.

A note on the `\pause` command: Both BEAMER and TEXPOWER implement this command and they have the same semantics; so there is no need to map this command to anything different in `beamertexpower`. However, a difference is that `\pause` can be used almost anywhere in BEAMER, whereas is may only be used in non-nested situtations in TEXPOWER. Since BEAMER is only more flexible than TEXPOWER here, this will not cause problems when porting.

In the following, the effect of TEXPOWER commands in BEAMER are listed.

`\stepwise{`⟨*text*⟩`}`

As in TEXPower, this initiates text in which commands like `\step` or `\switch` may be given. Text contained in a `\step` command will be enclosed in an `\only` command with the overlay specification `<+(1)->`. This means that the text of the first `\step` is inserted from the second slide onward, the text of the second `\step` is inserted from the third slide onward, and so on.

`\parstepwise{`⟨*text*⟩`}`

Same as `\stepwise`, only `\uncover` is used instead of `\only` when mapping the `\step` command.

`\liststepwise{`⟨*text*⟩`}`

Same as `\stepwise`, only an invisible horizontal line is inserted before the ⟨*text*⟩. This is presumable useful for solving some problems related to vertical spacing in TEXPOWER.

`\step{`⟨*text*⟩`}`

This is either mapped to `\only<+(1)->`⟨*text*⟩ or to `\uncover<+(1)->`⟨*text*⟩, depending on whether this command is used inside a `\stepwise` environment or inside a `\parstepwise` environment.

`\steponce{`⟨*text*⟩`}`

This is either mapped to `\only<+(1)>`⟨*text*⟩ or to `\uncover<+(1)>`⟨*text*⟩, depending on whether this command is used inside a `\stepwise` environment or inside a `\parstepwise` environment.

`\switch{`⟨*alternate text*⟩`}{`⟨*text*⟩`}`

This is mapped to `\alt<+(1)->{`⟨*text*⟩`}{`⟨*alternate text*⟩`}`. Note that the arguments are swapped.

`\bstep{`⟨*text*⟩`}`

This is always mapped to `\uncover<+(1)->`⟨*text*⟩.

`\dstep`

This just advances the counter `beamerpauses` by one. It has no other effect.

`\vstep`

Same as `\dstep`.

`\restep{`⟨*text*⟩`}`

Same as `\step`, but the ⟨*text*⟩ is shown one the same slide as the previous `\step` command. This is implemented by first decreasing the countern `beamerpauses` by one before calling `\step`.

`\reswitch{⟨alternate text⟩}⟨text⟩`

Like `\restep`, only for the `\switch` command.

`\rebstep⟨text⟩`

Like `\restep`, only for the `\bstep` command.

`\redstep`

This command has no effect.

`\revstep`

This command has no effect.

`\boxedsteps`

Temporarily (for the current TEX group) changes the effect of `\step` to issue an `\uncover`, even if used inside a `\stepwise` environment.

`\nonboxedsteps`

Temporarily (for the current TEX group) changes the effect of `\step` to issue an `\only`, even if used inside a `\parstepwise` environment.

`\code{⟨text⟩}`

Typesets the argument using a boldface typewriter font.

`\codeswitch`

Switches to a boldface typewriter font.

12 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):

12.1 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.

12.2 Terms and Conditions For Copying, Distribution and Modification

12.3 No Warranty