ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Making Presentations with LaTeX Guidelines

Xavier Perseguers
xavier.perseguers@epfl.ch

Computer Sciences

Semester Project

March–June 2004

Person in charge
Prof. Serge Vaudenay
serge.vaudenay@epfl.ch
EPFL / LASEC

Contents

Disclaimer

This document is not intended to replace the reference manuals of the corresponding presentation packages available for LaTeX. It may however help you choosing the best package fitting your needs and then getting as quick as possible a basic understanding on how designing your slides.

The comprehension of this document does not require high LaTeX skills but assumes nevertheless being familiar with the LaTeX environment and a basic understanding of macros.

When possible, a try has been made to explain common problems or strange behaviours and how to prevent or correct them.

You will find all useful materials presented in this document (such as source code) on the Web:

http://www.perseguers.ch/latex/contrib/presentations/

1 | Introduction

Contents

1.1 Packages for LaTeX

This document tries to show some possible solutions for creating screen based presentations. As there exists lots of tools for creating screen or online presentations, a choice has been made over solutions like beamer, foiltex, HA-prosper, ifmslide, PPower4, Prosper, seminar.sty, TeXPower and so on, to retain only three of them: Beamer, Prosper and TeXPower. A good document for starting using other PDF based solutions or even HTML based solutions like DocBook slides or latex2slides is available at

http://www.miwie.org/presentations/

1.1.1 Beamer

Beamer is a really easy-to-use package to create nice PDF presentations. Changing a parameter at the beginning of the document allows you to output either a standard presentation, or handouts or even the whole presentation as a standard LaTeX article.

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

Index: Beamer-relative commands are followed by the symbol "[B]" in the index, starting at page 77.

1.1.2 Prosper

This is a set of macros which allows you to generate PostScript or PDF presentations. There are certain advantages of this package over the others. First, though it has a simple structure, it provides enough options to generate good-looking slides. All the features of a PDF document (such as transitions, overlays, etc.) are available. In addition, it is easy to generate different slide styles, à la PowerPoint. Of course, you still have access to the full power of TeX, so you are free to extend your documents if you have the knowhow. For LaTeX beginners, however, Prosper encapsulates a lot of the details in an easy-to-use manner.

Homepage: http://prosper.sourceforge.net

Good to Know: This package is provided with lots of presentation styles.

Index: PROSPER-relative commands are followed by the symbol "[P]" in the index, starting at page 77.

1.1.3 TEXPower

This is an "all-inclusive" bundle to aid creating presentations. It provides color and font management, basic effects for incremental display, panels, navigation aids. The main distinguish features of TEXPower are:

- independent of the way PDF is created;
- independent of document class;
- implement display effect by LATEX programming.

Homepage: http://texpower.sourceforge.net

Index: TEXPower-relative commands are followed by the symbol "[T]" in the index, starting at page 77.

2 | Beamer

Contents

2.1 Introduction

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 a `frame` environment.
4. Run `pdflatex` on the text.

The following code shows a typical usage of the class (*see also* Figures 2.1 and 2.2).

```
\documentclass{beamer}

% Load a theme (graphics, colors, ...) for the presentation
\usepackage{beamerthemesplit}

\title{Example Presentation}
\author{Xavier Pers\'eguers}
\date{\today}

\begin{document}

\frame{\titlepage}

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

\section{Introduction}
\subsection{Overview of this class}
\frame
{
  \frametitle{List displayed step-by-step}

  \begin{itemize}
    \item<1-> Normal LaTeX class;
    \item<2-> Easy overlays;
    \item<3-> Straightforward use!
  \end{itemize}
}

\section{Current Activities}
\subsection{...}
\subsection{...}

\section{Our Goals}
\subsection{...}
\subsection{...}

\end{document}
```

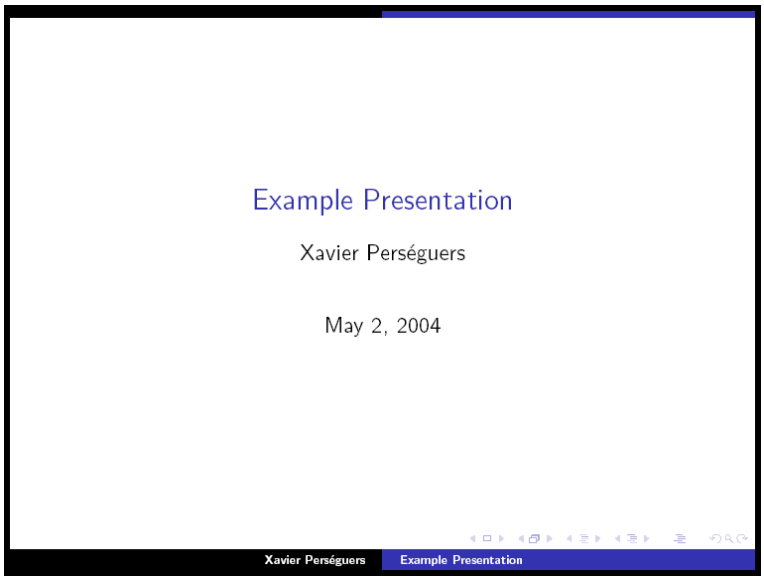


Figure 2.1: BEAMER: Title of a presentation.

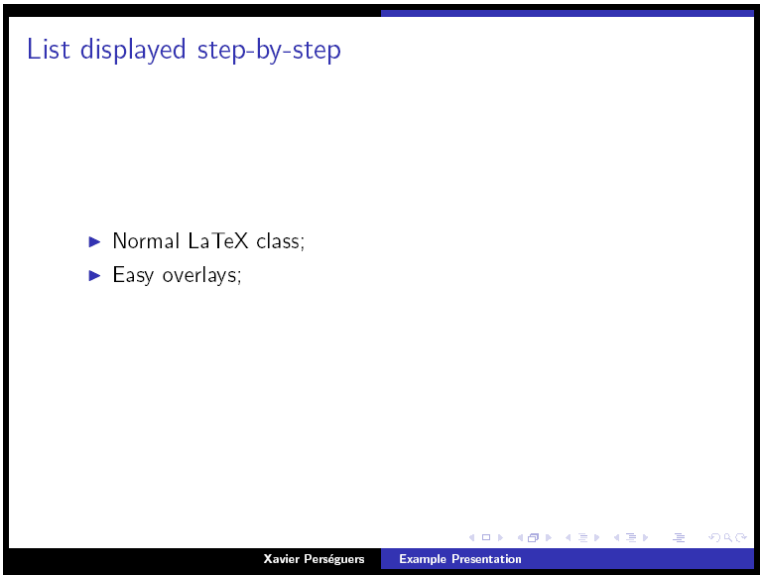


Figure 2.2: BEAMER: A list displayed incrementally.

2.1.1 Options of the Class

To make a presentation using the `beamer` class, you need to specify it in your `\documentclass`. Thus, the first line in the LaTeX file should be of the form:

`\documentclass`[⟨*options*⟩]`{beamer}`

There are several options that can be specified to the package:

slidestop puts frame titles on top left corner (default is **slidescentered**).

compress makes all navigation bars as small as possible (default is **uncompressed**).

mathserif/mathsans uses fonts with serif for maths (default is to use sans-serif fonts as for the text).

sans/serif uses fonts with or without serif for the text (default is to use sans-serif fonts).

handout for PDF handouts.

trans for PDF transparency.

Font Size default is 11pt but may take following values: 8pt, 9pt, 10pt, 11pt, 12pt, 14pt, 17pt, 20pt (*see also* C.1).

Another important option to specify is which presentation style to use. BEAMER comes with several styles which are described in § 2.7.

There are also options to specify slide background colors, slide numbers, etc. In general, unless you require black and white slides (e.g., for printing purposes), you will not need to set any color options in the `\documentclass`; the style files will manage them for you.

2.2 Frames

As with most presentation packages, the "unit" in presentations (or *slide*) is called a *frame* in Beamer. You have to inform LaTeX the contents to be typeset on each frame. This is easily performed with the command `\frame` as shown below:

```
\frame{
  \frametitle{Title of the frame}

  Contents such as maths, lists, ...
}
```

If you like the way LaTeX is able to deal with automatic page breaks, and you have seen that TeXPower does it too for slides, you may wonder if you could do the same with Beamer.

This is usually considered as a wrong method as 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. With automatic frame-break, you may create endless presentations that look more like a "paper projected on the wall" than a presentation. Nevertheless, if you would like to activate this feature for a certain frame, you may pass the option `allowframebreaks` to the frame definition:

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

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

When the option `allowframebreaks` is given, the frame will be automatically broken up into several frames, if it does not fit on a single slide. In details, the following things happen:

1. The option `containsverbatim` (*see* § 2.9.1) is automatically selected, as a side-effect. Thus, frames with this option set may contain verbatim text;
2. Consequently, overlays (*see* next section) are not supported;
3. Any footnotes for the frame will be inserted at the last page of the frame;
4. If there is a frame title, each page will have this frame title, with a special note (usually a Roman number) added indicating which page of the frame that page is.

Beware: 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. 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*⟩. The full syntax is:

`\frame[allowsframebreak=`⟨*fraction*⟩`]{ ... }`

2.3 Overlays

When creating overlays, how do you specify on which slides of a series of slides a certain text should be shown? The approach taken by most presentation classes is to introduce new commands, which get a certain slide number as input and which affect text on the slide following this command in a certain way.

Beamer uses a different approach. 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". Consider the following example.

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

2.3.1 Dynamically Replacing Text

Another example, using the command `\only` introduced by Beamer, lets you "throw away" its contents on slides that are not mentioned. In particular, it occupies no space.

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

There exists other replace commands:

`\uncover<`⟨*slides*⟩`>` If an 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.

`\invisible<`⟨*slides*⟩`>` Hides the text on the specified slides.

`\alt<`⟨*slides*⟩`>{`⟨*main*⟩`}{`⟨*alternative*⟩`}` The ⟨*main*⟩ text is shown on the specified slides, otherwise the ⟨*alternative*⟩ text.

`\temporal<`⟨*slides*⟩`>{`⟨*before*⟩`}{`⟨*main*⟩`}{`⟨*after*⟩`}` The ⟨*main*⟩ text is displayed on the specified slides, otherwise either the parameter ⟨*before*⟩ if the current slide is logically before the specified slides, or the parameter ⟨*after*⟩ if it is logically after the specified slides.

In case of problems with the heights of replacements, two environments may be used:

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.

```
\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}
```

overprint Inside the environment, use `\onslide` commands to specify different things that should be shown for this environment on different slides. The overlay specifications of the `\onslide` commands must be disjoint (*see* next section).

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

2.3.2 Specifying Ranges of Slides

The syntax of (basic) overlay specification is the following: they are comma-separated list of slides and ranges. Ranges are specified like this: `2-5`, that means slide two to five. The start or the beginning of a range or the end (but not both of them) may be omitted as it was the case with the itemize environment of the first example, at the beginning of the chapter. An example is `3-` meaning "slides three, four, five, and so on" as `-5` is equivalent to `1-5`.

2.3.3 Incremental Highlight

The *incremental highlight* is a way to step through an enumeration of items and displaying in another color (or *highlighting*) each item as it is introduced (*see also* § 2.8.2).

`\alert{`⟨*contents*⟩`}` Emphasizes ⟨*contents*⟩. If an overlay specification is given, as in the example below, only emphasizes ⟨*contents*⟩ at the corresponding slide(s).

The following example shows the three items of the list starting from slide #2 (<2–>) and alerts them one after the other: item *foo* at step (or "slide") 2, item *foo bar* at step 3 and item *foo bar bar* at steps 4 and greater.

```
\begin{itemize}
  \item<2->\alert<2>{foo}
  \item<2->\alert<3>{foo bar}
  \item<2->\alert<4->{boo bar bar}
\end{itemize}
```

2.3.4 Incremental Specifications

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

```
\begin{itemize}
  \item<1-> Item foo
  \item<2-> Item foo bar
  \item<3-> Item foo bar bar
\end{itemize}
```

The problems starts if you decide to insert a new item, for instance, at the beginning. In this case you would have to adjust all overlay specifications. Beamer offers a special *incremental overlay specification:*

```
\begin{itemize}
  \item<+-> Item foo
  \item<+-> Item foo bar
  \item<+-> Item foo bar bar
\end{itemize}
```

The +-sign may be use 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. The counter is increased by 1, at each animation step.

Incremental Highlight with the Incremental Overlay Specification

In the following example, we use the incremental overlay specification to emphasize each item as it is introduced. The special specification `+-| alert@+` will be replaced by `1-| alert@1` for the first item, `2-| alert@2` for the second... The notation `| alert@1` is a way to specify an special action to be taken at the corresponding step(s). It should be understood as something like that:

```
\item<+- \alert<+>>
        or
\item<1- \alert<1>>
\item<2- \alert<2>>
         ⋮
```

```
\begin{itemize}
  \item<+-| alert@+> foo
  \item<+-| alert@+> foo bar
  \item<+-| alert@+> boo bar bar
\end{itemize}
```

Or even shorter...

```
\begin{itemize}[<+-| alert@+>]
  \item foo
  \item foo bar
  \item boo bar bar
\end{itemize}
```

2.3.5 Quick Animations

Chapter 5 explains how you may create advanced animation effects in your presentation. Auto-advancing is a way to create a series of slides shown in rapid succession (*see* § 5.2.1). To facilitate the creation of animations using the auto-advancing feature, the following commands may be used:

`\animate<`⟨*overlay specification*⟩`>` Shows the slides specified by the parameter ⟨*overlay specification*⟩ 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,
```

```
8     the fifth slide is shown.
9
10    % Code for creating an animation with five slides
11    % [...]
12  }
```

`\animatevalue<`⟨*slides*⟩`>{`⟨*name*⟩`}{`⟨*start value*⟩`}{`⟨*end value*⟩`}` Lets vary a counter or a dimension ⟨*name*⟩ 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* (first argument in ⟨*slides*⟩), the counter or dimension is set to ⟨*start value*⟩; on the slides after the *end slide* (last argument in ⟨*slides*⟩) it is set to ⟨*end value*⟩.

Example:

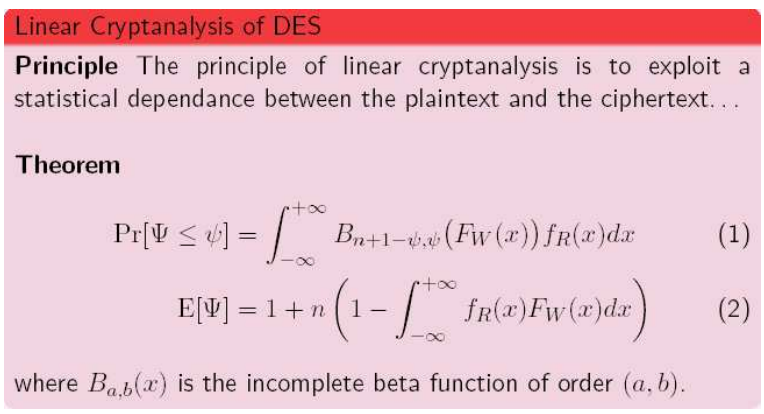
```
\newcount\opaqueness
\frame{
  \animate<2-10>
  \animatevalue<1-10>{\opaqueness}{100}{0}
  \begin{colormixin}{\the\opaqueness!averagebackgroundcolor}
    \frametitle{Fadeout Frame}

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

2.4 Framed Text

If you wish to emphasize a block of text such as a theorem, a formula or anything else, you may use the `beamerboxesrounded` environment to draw a box around your text, as shown on the image below.

```
\begin{beamerboxesrounded}{Linear Cryptanalysis of DES}
  % [...]
  \begin{align}
    \Pr[\Psi \leq \psi] &=
          \int_{-\infty}^{+\infty} B_{n+1-\psi,\psi}
          \bigl(F_W(x)\bigr) f_R(x) dx \\
    \mathrm{E}[\Psi] &=
          1+n\left(1-\int_{-\infty}^{+\infty} f_R(x) F_W(x) dx\right)
  \end{align}
  % [...]
\end{beamerboxesrounded}
```



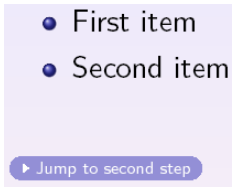


Figure 2.3: BEAMER: Hyperlink as a button.

2.5 Interaction

During most presentations, you would like to present your slides in a linear fashion. However, there are different reasons why you might have to deviate from this linear order such as:

- go back to an earlier slide;
- present a complicated picture and "zoom out" different part to explain details;

2.5.1 Jumps

To create nonlinear jumps in your presentation, you can add hyperlinks. A hyperlink is usually rendered as a button that, when you click on it, jumps to some other slide. The example below shows how you may create a hyperlink to go to a specific step of the animation of a *labeled* frame.

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

  \hyperlink{itemAnimationSlide<2>}%
            {\beamergotobutton{Jump to second step}}
}
```

`\beamerbutton{⟨text⟩}` Draws a button with the given ⟨*text*⟩.

`\beamergotobutton{⟨text⟩}` Draws a button with the given ⟨*text*⟩. Before the text, a small symbol (usually a right-pointing arrow) is inserted.

`\beamerskipbutton{⟨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 presentation, such as a demonstration.

Example:

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

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

```
11        \begin{proof}
12          ...
13        \end{proof}
14    \end{overprint}
15  }
```

`\beamerreturnbutton{`*⟨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:

```
1  \frame<1>[label=mytheorem]{
2    \begin{theorem}
3      ...
4    \end{theorem}
5
6    \begin{overprint}
7      \onslide<1>
8        \hfill\hyperlink{mytheorem<2>}{%
9          \beamergotobutton{Go to proof details}}
10     \onslide<2>
11       \begin{proof}
12         ...
13       \end{proof}
14       \hfill\hyperlink{mytheorem<1>}{%
15         \beamerreturnbutton{Return}}
16   \end{overprint}
17 }
18 \appendix
19 \againframe<2>{mytheorem}
```

2.5.2 Zoom

Sometimes, a graphic may be complex and you are willing to spend much time explaining it in great detail. In this case, you will often run into the problem that fine details of the graphic are hard to discern. 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.

`\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 area is created inside the frame. The size of this area is given by *⟨zoom area width⟩* and *⟨zoom area depth⟩*. The upper left corner of this area is given by *⟨upper left x⟩* and *⟨upper left y⟩*. 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 the frame title).

By default, the button is clickable, but it will not be indicated in any special way. To draw a border around the button, use the following

⟨*option*⟩: `border=`⟨*width in pixels*⟩. If not given, ⟨*width in pixels*⟩ is equal to 1.

When you press the button, Adobe Acrobat Reader will jump to the frame specified by the ⟨*zoomed overlay specification*⟩. Clicking the whole text area of the zoomed frame jumps back to the previous location.

Example:

```
\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]{complicatedimage}
}
```

Zoom area as a whole frame:

```
\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]{complicatedimage}
}
```

2.6 Compatibility with Other Packages

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

`\usepackage{`⟨*amsmath*⟩`}` This package is automatically loaded since package 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.

`\usepackage[`⟨*french*⟩`]{`⟨*babel*⟩`}` When using the ⟨*french*⟩ style, certain features that clash with the functionality of the `beamer` class will be turned off. For instance, enumerations are still produced the way the theme dictates, not the way the ⟨*french*⟩ style does. Also, the characters : and ! will not be 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".

2.7 Presentation Styles

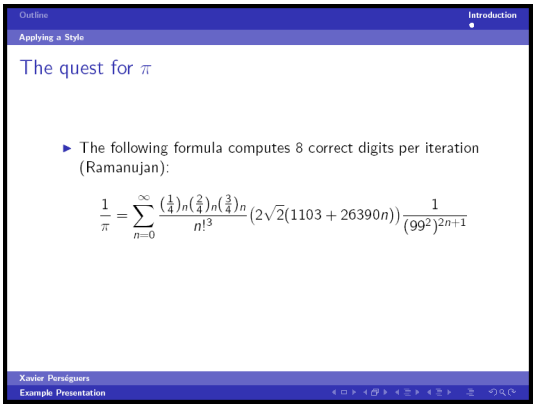
All automatically installed BEAMER styles have a name of the form

`beamerthemestyle`.

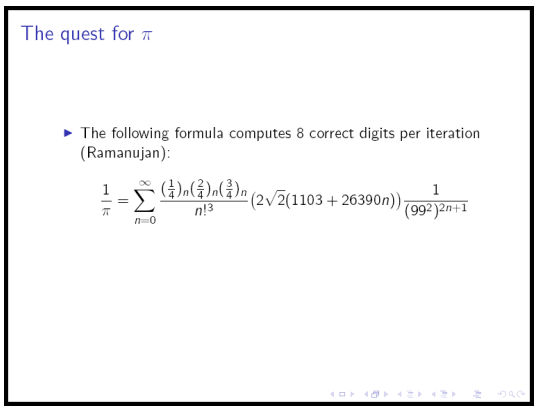
In § 2.7.1, a style with a dash is in fact a *parametrized style*.

E.g., Style `beamerthemesidebar-tab` should be included in the document preamble as `\includepackage[tab]{beamerthemesidebar}`.

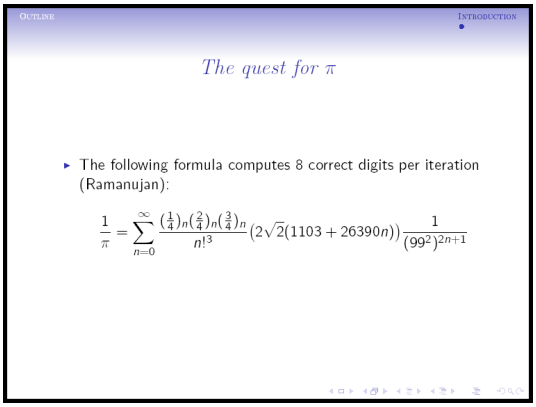
2.7.1 Available Styles



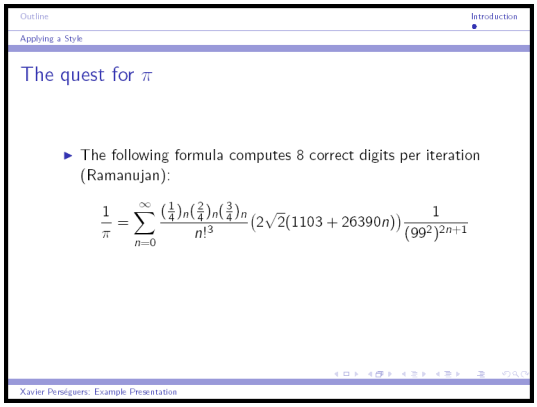
beamerthemebars



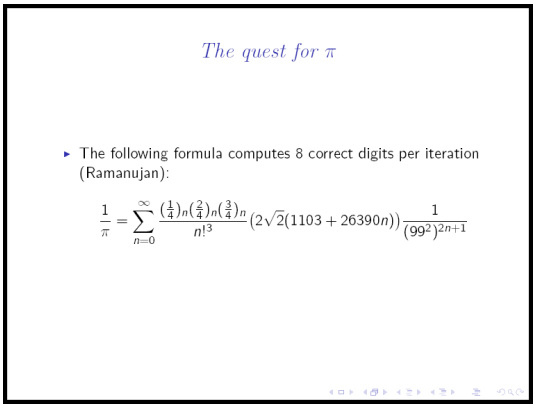
beamerthemeboxes



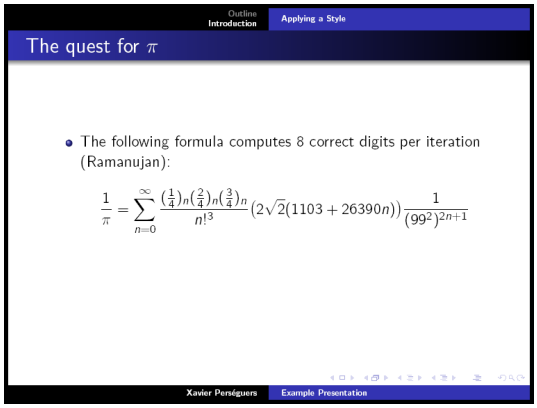
beamerthemeclassic



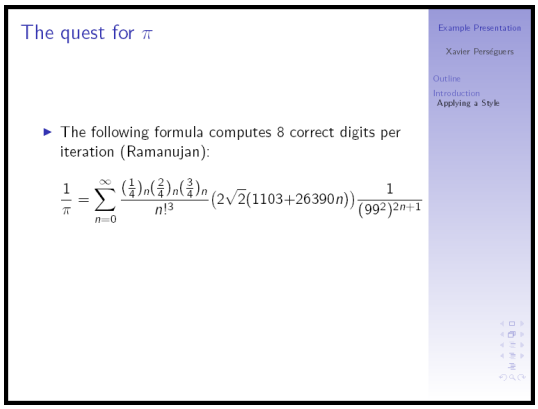
beamerthemelined



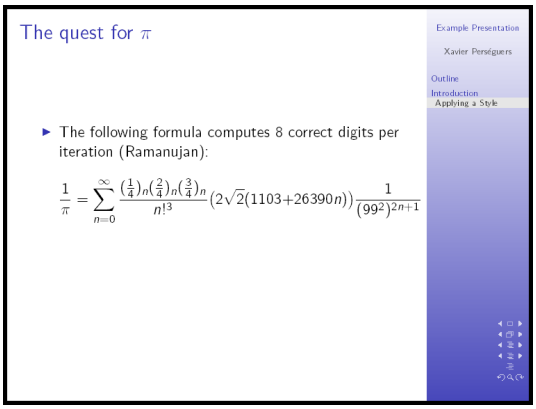
beamerthemeplain



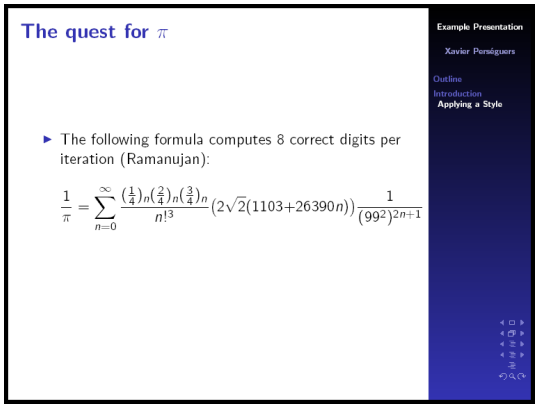
beamerthemeshadow



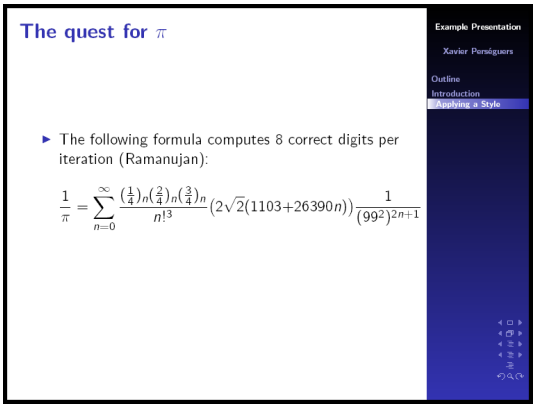
beamerthemesidebar



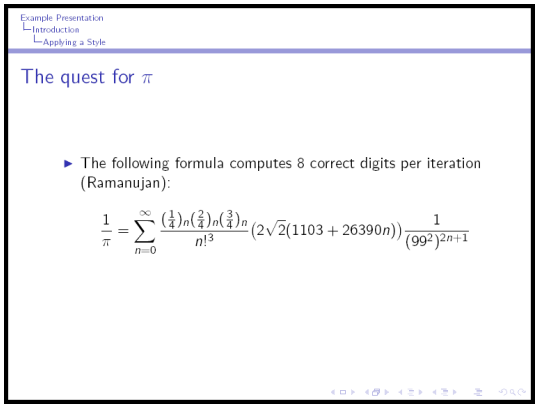
beamerthemesidebar-tab



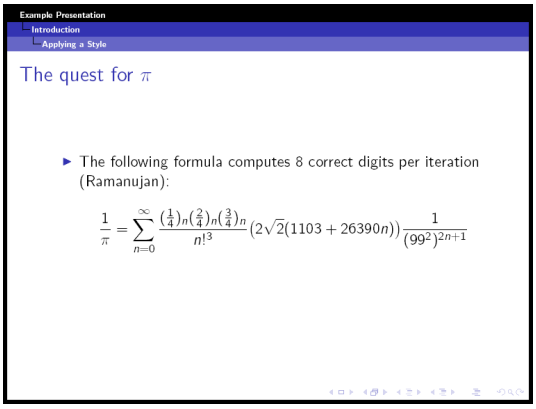
beamerthemesidebardark



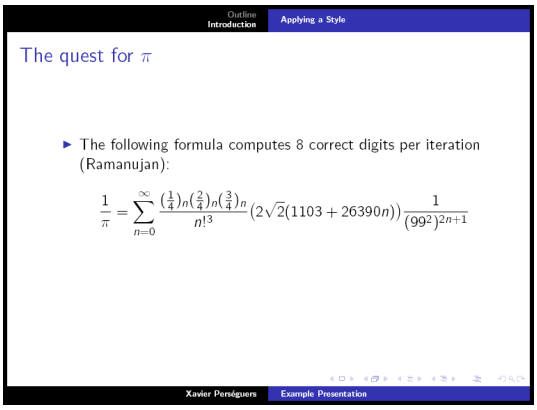
beamerthemesidebardark-tab



beamerthemetree



beamerthemetree-bars



beamerthemesplit

2.7.2 Choosing Another Color Theme

There exists three additional options you may specify with the `\documentclass` definition:

red changes navigation bars and titles to reddish color.

brown changes navigation bars and titles to brownish color.

blackandwhite changes navigation bars and titles to black, white and gray colors.

Colors

BEAMER automatically loads `xcolor` package by Uwe KERN, which supports `color` and `pstcol` packages.

Predefined Colors: **black**, **blue**, **brown**. **cyan**, **darkgray**, **gray**, **lightgray**, **green**, **magenta**, **orange**, **purple**, **red**, **violet**, **white** and **yellow**.

Defining new Colors. You should use `xcolor` definition scheme:

- `\xdefinecolor{lavendar}{rgb}{0.8,0.6,1}`
- `\colorlet{mygreen}{green!60!gray}`
  which means 60% green + 20% gray.
- When a color is needed, you may use directly the method above (e.g., `blue!70` for having a 70% blue).

Background Color

- To set solid background color:
  `\beamersetaveragebackground`{⟨*color*⟩} *or*
  `\beamertemplatesolidbackgroundcolor`{⟨*color*⟩}
- To set gradient background color:
  `\beamertemplateshadingbackground`{⟨*color1*⟩}{⟨*color2*⟩}
- To set grid background:
  `\beamertemplategridbackground`

2.8 Useful Macros

2.8.1 Removing Navigation symbols

Insert the command `\beamertemplatenavigationsymbolsempty`.

2.8.2 Incremental Highlight

The *incremental highlight* is a way to step through an enumeration of items and displaying in another color (or *highlighting*) each item as it is introduced (*see also* § 2.3.3).

File name: `beamer-highlight.tex`
Source: taken from the BEAMER User Guide

```
\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}
}
```

2.8.3 Using Packages only when Printing

If, for some reason, you wish to include packages only when generating an article out of a presentation, you may include the structure described in the code below in the preamble of the document.

```
\mode<article>       % only for the article version
{
  \usepackage{beamerbasearticle}
  \usepackage{fullpage}
  \usepackage{hyperref}
}
\mode<presentation> % only for the presentation version
{
  \usepackage{beamerthemeshadow}
}
```

2.8.4 Structuring Presentation

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 highlighted. To facilitate this, you can use the following 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]
  }
}
```

`\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]
  }
}
```

2.9 Identified Limitations

2.9.1 Verbatim Environment

"`\verb`" or "`verbatim`" cannot be *directly* used in BEAMER. However, if there is **no overlay**, use `\frame[containsverbatim]` as shown on example below.

```
\frame[containsverbatim]{
This slide contains a few lines of \LaTeX{} using the Verbatim
environment of package \texttt{fancyvrb}!

\begin{Verbatim}[gobble=6]
   1> \mode<article>       % only for article
   2> {
   3>   \usepackage{beamerbasearticle}
   4>   \usepackage{fullpage}
   5>   \usepackage{hyperref}
   6> }
   7> \mode<presentation> % only for presentation
   8> {
   9>   \usepackage{beamerthemeshadow}
  10> }
\end{Verbatim}

\emph{To be continued\ldots}
}
```

2.9.2 In-line Verbatim

Use the command `\path{...}` instead of `\verb`.

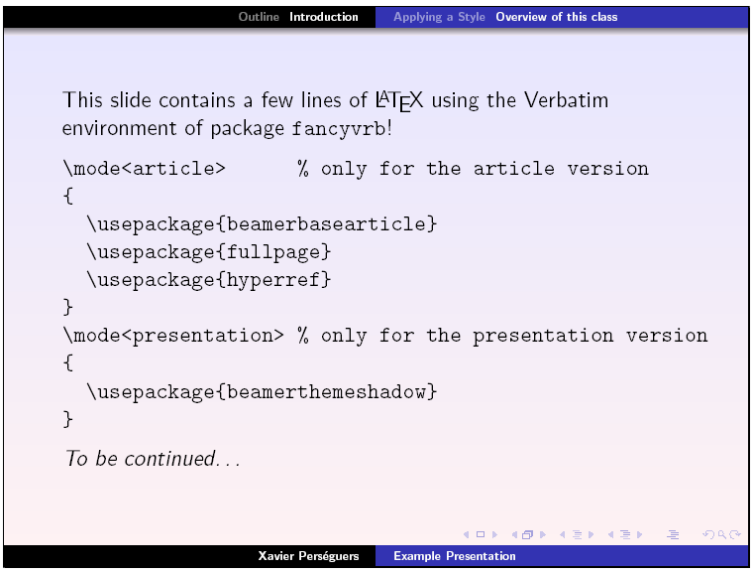


Figure 2.4: Including verbatim data in a frame.

3 | Prosper

Contents

3.1 Introduction

Figure 3.1 presents a bird's-eye view of the structure of a LaTeX file using the `prosper` class. The following code shows a typical usage of the class. Additional references may be found in [4].

```
\documentclass[slideColor,pdf,mancini]{prosper}

\title{Example Presentation}
\author{Xavier Pers\'eguers}
\email{xavier.perseguers@epfl.ch}
\institution{EPFL}

\begin{document}

\maketitle

\begin{slide}{Introduction}
  Nothing more to say!
\end{slide}

\overlays{3}{
  \begin{slide}[Dissolve]{List displayed step-by-step}
    \begin{itemstep}
      \item Prosper showing
      \item a list of items
      \item step-by-step
    \end{itemstep}
```

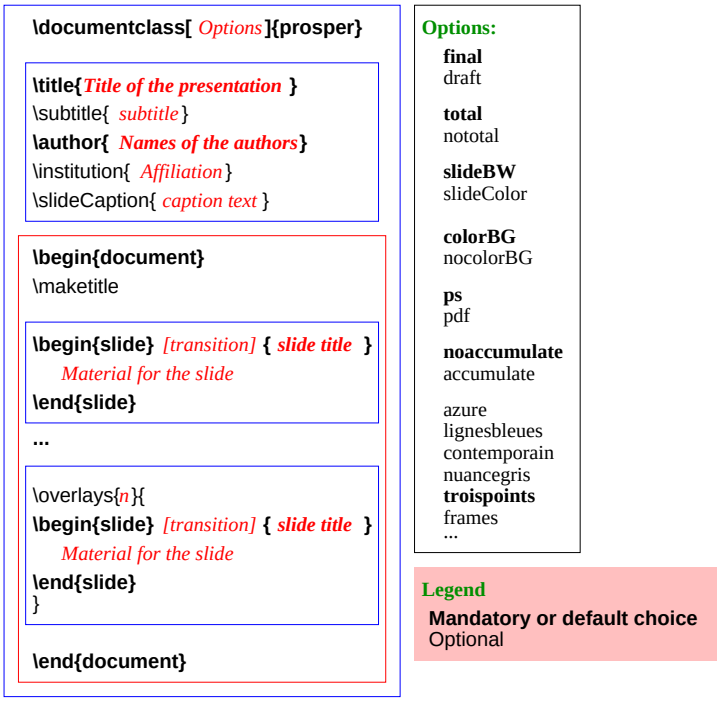


Figure 3.1: Structure for a LaTeX file using `prosper`

```
23    \end{slide}
24  }
25
26  \end{document}
```

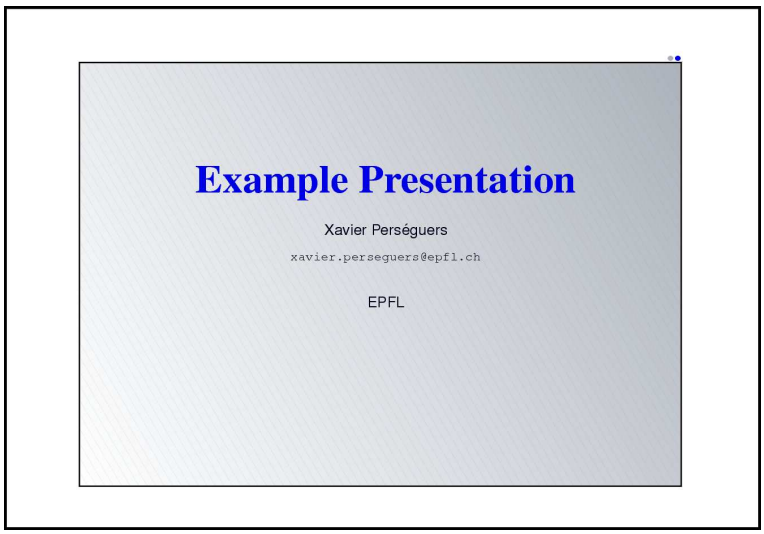


Figure 3.2: Prosper: Title of a presentation.

3.1.1 Options of the Class

To make a presentation using the Prosper package, you need to specify it in your `\documentclass` (you can also specify it in a `\usepackage` command in the preamble). Thus, the first line in the LaTeX file should be of the form:

`\documentclass[`⟨*options*⟩`]{prosper}`

Figure 3.3: PROSPER: A list displayed incrementally.

There are several options that can be specified to the package. You can read about all the options in detail in the documentation that comes with PROSPER. The useful ones are:

draft compiles a draft version of the presentation, with figures replaced by bounding boxes.

final compiles a complete version of the presentation with figures and captions in their proper places.

ps compiles the LaTeX file to PostScript for printing purposes.

pdf compiles the LaTeX file to a PDF format suitable for projectors.

Another important option to specify is which presentation style to use. PROSPER comes with several styles which are described in § 3.5.

There are also options to specify slide background colors, slide numbers, etc. In general, unless you require black and white slides (e.g., for printing purposes), you will not need to set any color options in the `\documentclass`; the style files will manage them for you.

3.2 Preamble

In the section between `\documentclass` and `\begin{document}`, you should specify the contents of the title page and some options (such as logos and slide captions) that can be applied to all slides. The normal LaTeX macros have been redefined to generate the title and associated texts with proper font sizes, etc. Some of the macros available for designing the title slide include:

`\title` Defines the title of the presentation.

`\subtitle` Defines the subtitle of the presentation.

`\author` Defines the author of the presentation.

`\email` Defines the email of the author.

`\institution` Defines the institution.

`\slideCaption` Puts a caption at the bottom of each slide.

`\Logo` Places a logo on each slide at a specified position:
`\Logo(-1.2,-1.2){\includegraphics{logo-filename}}`

`\DefaultTransition` Defines the type of transition that should occur between slides. *See also* § 5.1, p. 47.

Since the `hyperref` package is included by PROSPER, you can use the `\href` command to include `mailto:` links or direct hyperlinks to Web pages in the above commands (and, of course, in the rest of your document). As in standard LaTeX, the title slide is generated by the `\maketitle` command in the document body.

3.3 Slides

The "frame" of BEAMER is the "slide" in PROSPER. You still have to inform LaTeX the contents to be typeset on each slide. This is easily performed with the environment `slide` as shown below:

```
\begin{slide}{Title of the slide}

Contents such as maths, lists, ...

\end{slide}
```

3.3.1 Special Slide

`\part[`⟨*transition*⟩`]{`⟨*text*⟩`}` Creates a slide only containing ⟨*text*⟩ vertically and horizontally centered. The optional transition ⟨*transition*⟩ will be used for this slide, if specified (*see* § 5.1 for more informations on transitions).

3.4 Overlays

Overlays may be used to animate contents of a slide. To create a sequence of elements appearing and disappearing, you have to embed the corresponding slide environment into an `\overlays` definition:

```
\overlays{n}{
  \begin{slide}{...}
    ...
  \end{slide}
}
```

The argument ⟨*n*⟩ stands for the number of steps composing the animation. This is a manually computed value.

`\fromSlide{`⟨*p*⟩`}{`⟨*mat*⟩`}` Puts ⟨*mat*⟩ on slides ⟨*p*⟩ through ⟨*n*⟩.

`\onlySlide{`⟨*p*⟩`}{`⟨*mat*⟩`}` Puts ⟨*mat*⟩ on slide ⟨*p*⟩ only.

`\untilSlide{`⟨*p*⟩`}{`⟨*mat*⟩`}` Puts ⟨*mat*⟩ on slides 1 through ⟨*p*⟩.

`\FromSlide{⟨p⟩}` All the material after the occurence of this command will be put on slides ⟨p⟩ through ⟨n⟩.

`\OnlySlide{⟨p⟩}` All the material after the occurence of this command will be put on slide ⟨p⟩ only.

`\UntilSlide{⟨p⟩}` All the material after the occurence of this command will be put on slides 1 through ⟨p⟩.

3.4.1 List of Items

If you wish to step through a list of items, you should use the `itemstep` environment instead of the well-known `itemize` environment.

```
\overlays{3}{
  \begin{slide}{List of Items}
    \begin{itemstep}
      \item First item
      \item Second item
      \item Third item
    \end{itemstep}
  \end{slide}
}
```

3.4.2 Replacing Contents

The commands `\fromSlide*`, `\onlySlide*` and `\untilSlide*` have the same definition as their unstarred version except that they typeset ⟨*mat*⟩ in a zero dimension box, meaning that the position pointer is not moved.

```
\overlays{3}{%
  \begin{slide}{Example}
    \onlySlide*{1}{\includegraphics{example-1.eps}}%
    \onlySlide*{2}{\includegraphics{example-2.eps}}%
    \onlySlide*{3}{\includegraphics{example-3.eps}}%
    \onlyInPS{\includegraphics{example.eps}}%
  \end{slide}
}
```

The example above will put image `example-1.eps` on the first slide, replace it with image `example-2.eps` on the second slide and replace it again with image `example-3.eps` on the third and last slide. In PDF mode, the slide will be displayed in three steps. In PS mode, however, there will be only one slide containing image `example.eps`.

The other usefull commands for choosing the contents depending on the chosen mode (PS or PDF) are:

`\PDForPS{⟨ifPDF⟩}{⟨ifPS⟩}` Puts contents ⟨*ifPDF*⟩ if the chosen mode is PDF, otherwise puts contents ⟨*ifPS*⟩.

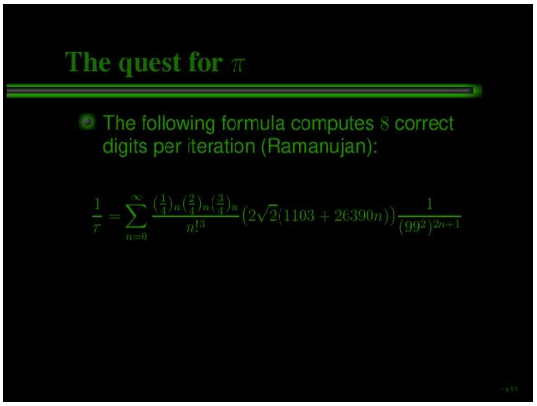
`\onlyInPS{⟨contents⟩}` Puts ⟨*contents*⟩ only if the chosen mode is PS.

`\onlyInPDF{⟨contents⟩}` Puts ⟨*contents*⟩ only if the chosen mode is PDF.

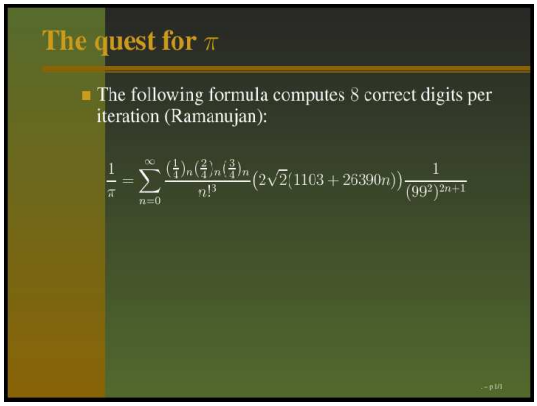
3.5 Presentation Styles

Prosper already offers several styles tuned for printing slides in both color and black & white, as well as displaying on a screen. Nevertheless, the class has been devised in such a way that it is fairly easy to add your own style if you are disatisfied with the existing ones. You are strongly encouraged to share Prosper styles you are proud of with other users by sending them to Prosper's author such that he can add them to the next release.

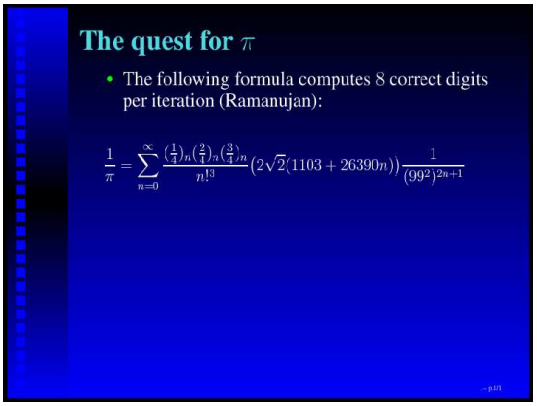
3.5.1 Available Styles



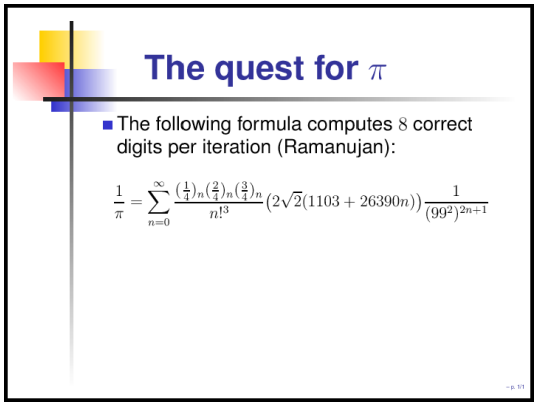
alienglow



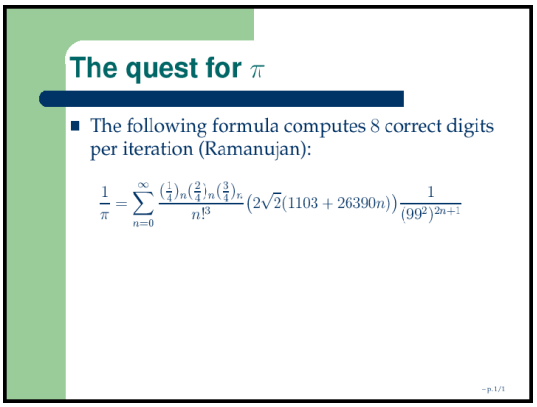
autumn



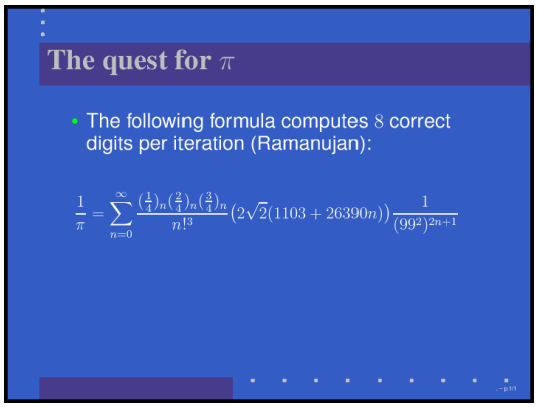
azure



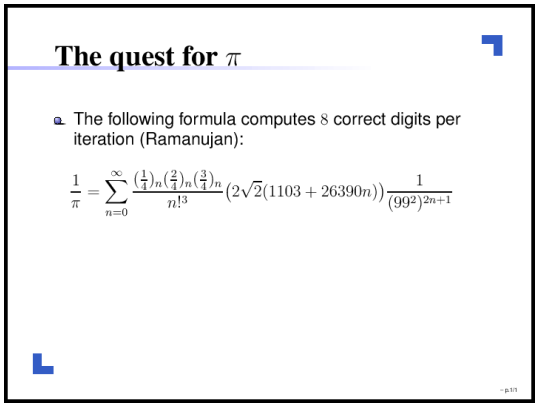
blends



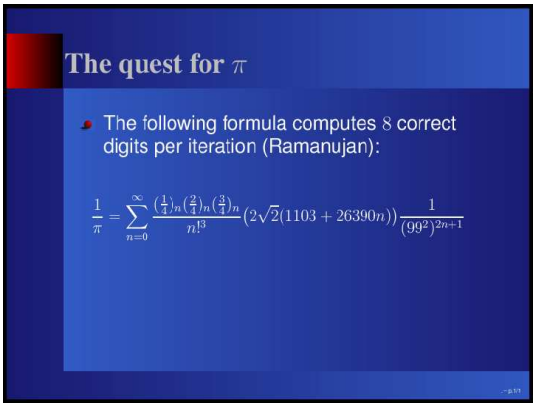
capsules



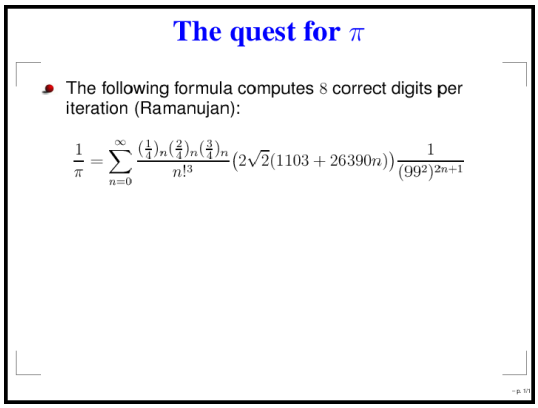
contemporain



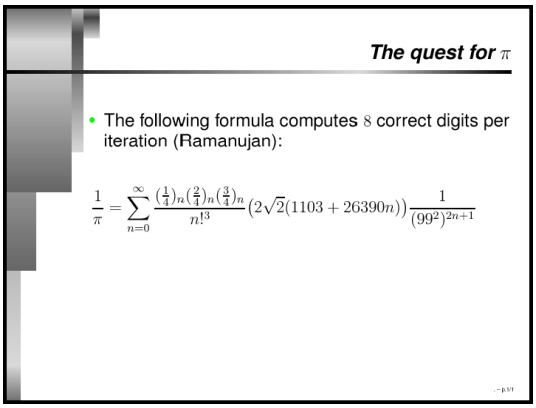
corners



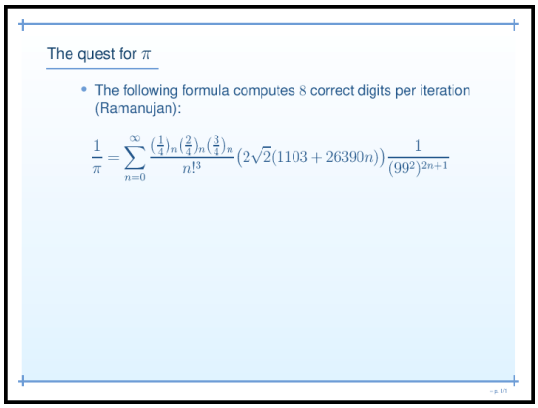
darkblue



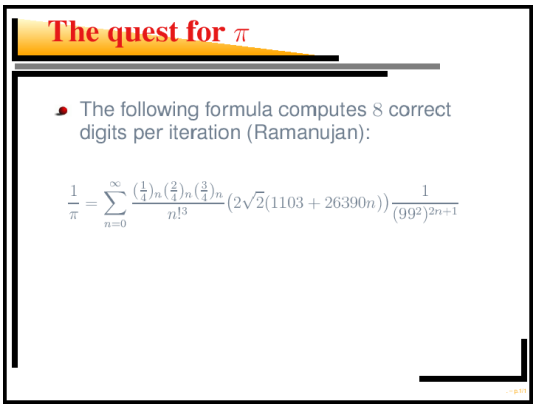
default



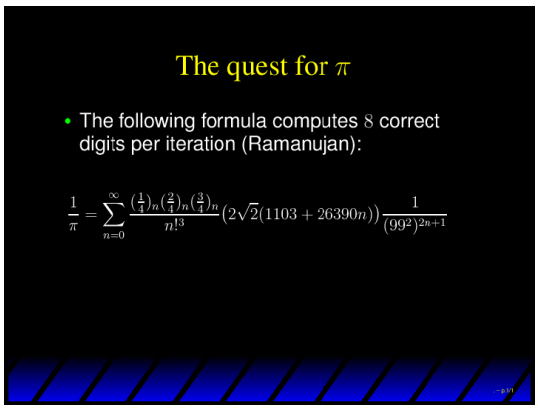
frames



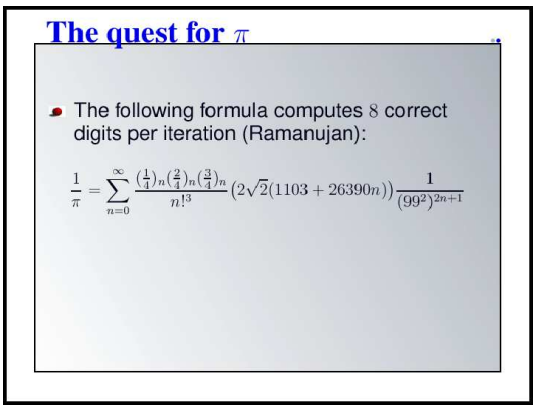
fyma



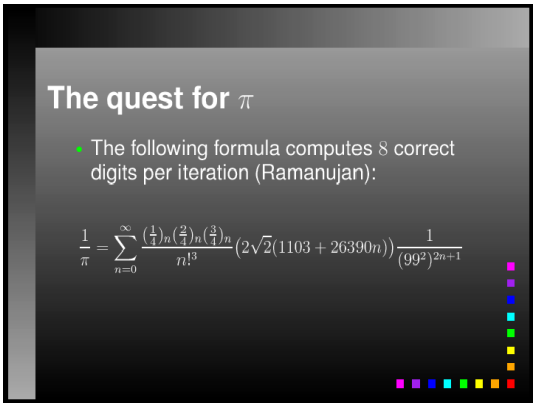
gyom



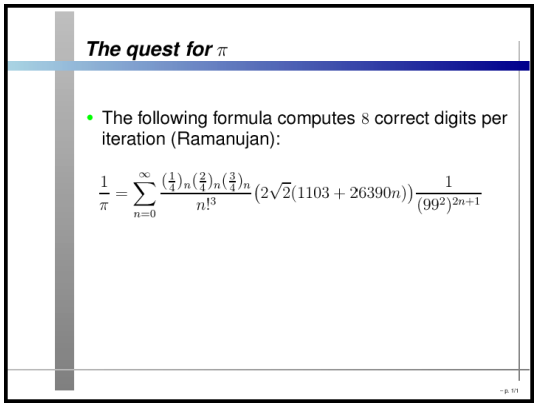
lignesbleues



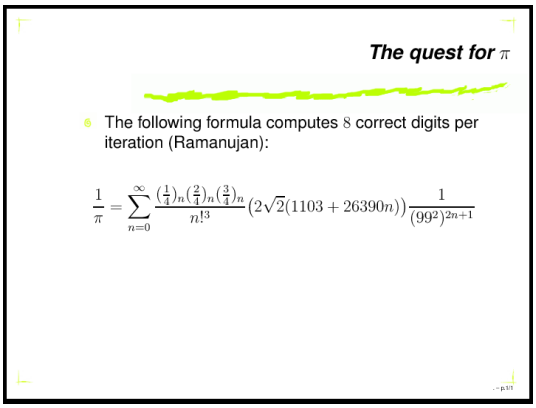
mancini



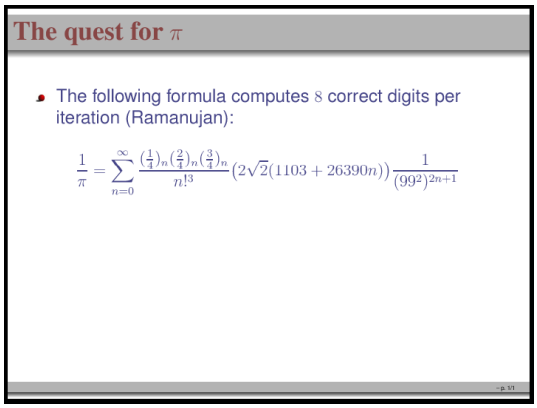
nuancegris



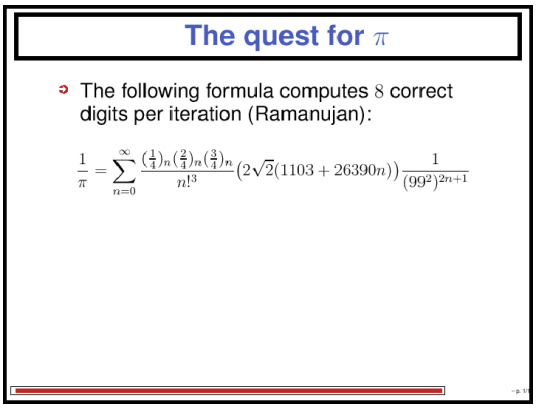
prettybox



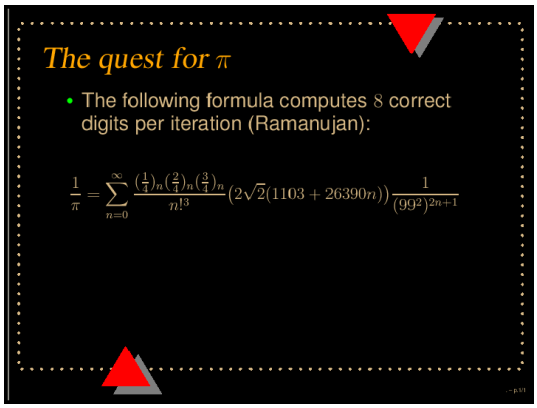
rico



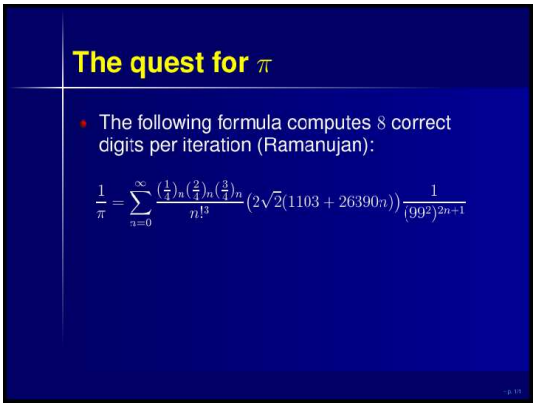
serpaggi



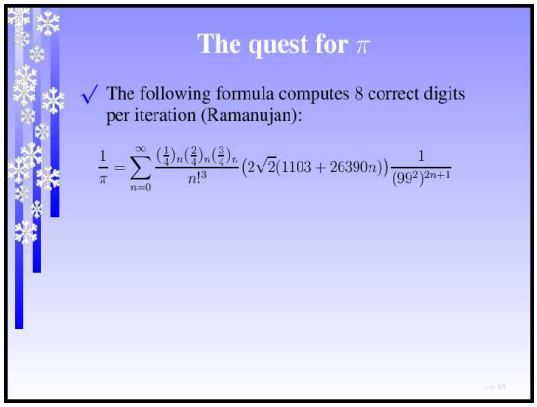
thomasd



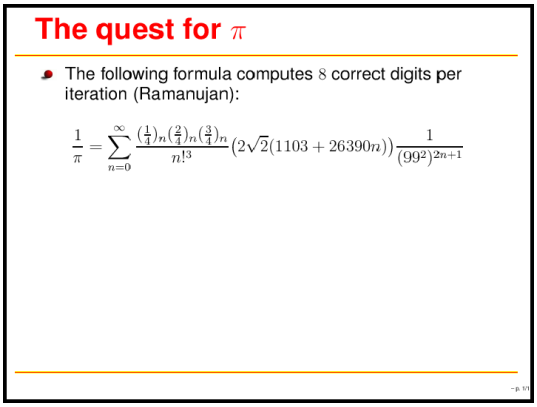
troispoints



whitecross



winter



wj

3.5.2 Defining new Styles

You can edit any of the above styles to create your own style — colors, ornaments, font and margins. Each of the above styles is defined in a file named "PPR*xxx*`.sty`", e.g., `PPRwinter.sty`, `PPRserpaggi.sty`, etc. These files are located somewhere in directory `TEXMF/tex/latex/prosper`, where `TEXMF` denotes the directory of your TEX tree.

Local Definition

Copy one of these `PPR*.sty` files to your own presentation directory, rename it (e.g., `PPR`*yourname*`.sty`), and edit it as you wish. Use that name as the style option in your TEX file:

`\documentclass[`*yourname*`]{prosper}`

Global Definition

Copy one of these `PPR*.sty` files in the same directory and edit it as you wish. You may need to rebuild the TEX-tree before using it.[1]

3.6 How do I...

3.6.1 How can I get a slide in portrait orientation?

If you want to add a "slide" of text between two common slides in portrait orientation, simply enter your text between the end of the previous slide (ended by `\end{slide}`) and the start of the next one (usually indicated by `\begin{slide}`).

3.6.2 How can I incrementally display tables?

To show a table cell-by-cell, you need to use the command `\fromSlide` with the contents of each cell as argument. Do not enclose the column delimiters (&) nor the new line character. Add the comment character (%) at the end of each line to prevent LATEX adding unwanted misalignment.

File name: `prosper-table-1.tex`

```
1 \overlays{6}{%
2 \begin{slide}{Table}
3   \begin{tabular}{|l|l|l|}
```

[1] If you use teTEX, this is achieved with the command `mktexlsr`.

```
    \hline
    a & %
    \fromSlide{2}{b} & %
    \fromSlide{3}{cdefghijklm} \\%
    \hline %
    \fromSlide{4}{nopq} & %
    \fromSlide{5}{rstuvw} & %
    \fromSlide{6}{xyz} \\ %
    \hline
  \end{tabular}
\end{slide}
}
```

Problem. The whole table is displayed from first slide with empty cells. What should I do?

Solution (preliminary). Create a single table for each step of the animation, allowing us to have a better control on which cell to display or not, and therefor not having empty cells at all.

Cell-by-Cell (preliminary)

File name: `prosper-table-2.tex`

```
\overlays{6}{%
\begin{slide}{Table}
  \onlySlide*{1}{%
    \begin{tabular}{|l|}
      \hline
      a \\
      \hline
    \end{tabular}
  }
  \onlySlide*{2}{%
    \begin{tabular}{|l|l|}
      \hline
      a & b \\
      \hline
    \end{tabular}
  }
  \onlySlide*{3}{%
    \begin{tabular}{|l|l|l|}
      \hline
      a & b & cdefghijklm \\
      \hline
    \end{tabular}
  }
  \onlySlide*{4}{%
    \begin{tabular}{|l|l|l|}
      \hline
      a & b & cdefghijklm \\
      \hline
      nopq \\
      \cline{1-1}
    \end{tabular}
  }
  \onlySlide*{5}{%
    \begin{tabular}{|l|l|l|}
      \hline
      a & b & cdefghijklm \\
      \hline
      nopq & rstuvwx \\
      \cline{1-2}
    \end{tabular}
  }
  \onlySlide*{6}{%
    \begin{tabular}{|l|l|l|}
      \hline
      a & b & cdefghijklm \\
      \hline
      nopq & rstuvw & xyz \\
      \hline
    \end{tabular}
  }
\end{slide}
}
```

Problem. Second row display makes cell grow as cell (2,1) is widder than cell (1,1). Problem is similar with cell (2,2) being widder than cell (1,2). Is it possible to get rid of this?

Solution (final). As said before, we need to create different tables for each step of the animation. Environment `tabular` cannot calculate properly the width of cells as it has no information on a second row being displayed later. The solution is hence to force the width of a cell according to the maximal width it will have at the end of the animation. There is a few adjustments to

a		

a	b	

a	b	cdefghijklm
nopq	rstuvw	

Figure 3.4: Cell-by-Cell — (`prosper-table-1.tex`)

be done manually whenever the cell needs to be stretch. The commands at lines 6 and 7 define a box whose width is equal to the largest contents of the first column of the table (line 6) or second column of the table (line 7). You may define other commands for further columns according to your needs.

Cell-by-Cell (final)

File name: `prosper-table-3.tex`

```
1  \newlength{\txtwidth}
2  \newcommand{\stretchto}[3][c]{%
3  \settowidth{\txtwidth}{#3}%
4  \makebox[\txtwidth][#1]{#2}}
5
6  \def\colOne#1{\stretchto[l]{#1}{nopq}}
7  \def\colTwo#1{\stretchto[l]{#1}{rstuvw}}

8  \overlays{6}{%
9  \begin{slide}{Table}
10   \onlySlide*{1}{%
11     \begin{tabular}{|l|}
12       \hline
13       \colOne{a} \\
14       \hline
15     \end{tabular}
16   }
17   \onlySlide*{2}{%
18     \begin{tabular}{|l|l|}
19       \hline
20       \colOne{a} &
21       \colTwo{b} \\
22       \hline
23     \end{tabular}
24   }
25   \onlySlide*{3}{%
26     \begin{tabular}{|l|l|l|}
27       \hline
28       \colOne{a} & \colTwo{b}
29         & cdefghijklm \\
30       \hline
31     \end{tabular}
32   }
33   \onlySlide*{4}{%
34     \begin{tabular}{|l|l|l|}
35       \hline
36       a & \colTwo{b}
37         & cdefghijklm \\
38       \hline
39       nopq\\
40       \cline{1-1}
41     \end{tabular}
42   }
43   \onlySlide*{5}{%
44     \begin{tabular}{|l|l|l|}
45       \hline
46       a & b & cdefghijklm \\
47       \hline
48       nopq & rstuvw \\
49       \cline{1-2}
50     \end{tabular}
51   }
52   \onlySlide*{6}{%
53     \begin{tabular}{|l|l|l|}
54       \hline
55       a & b & cdefghijklm \\
56       \hline
57       nopq & rstuvw & xyz \\
58       \hline
59     \end{tabular}
60   }
61 \end{slide}
62 }
```

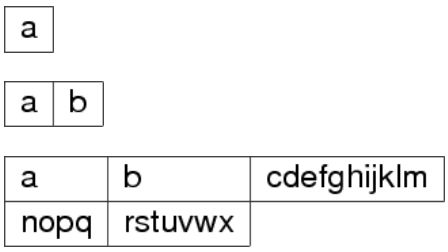


Figure 3.5: Cell-by-Cell — (`prosper-table-2.tex`)

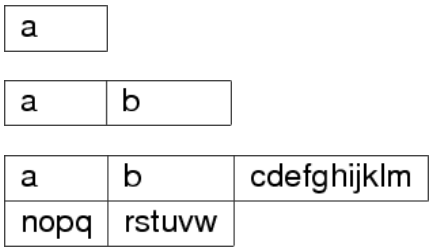


Figure 3.6: Cell-by-Cell — (`prosper-table-3.tex`)

3.7 Identified Bugs

Black Boxes in Adobe Acrobat Reader

You might encounter strange behaviour with your presentations when using Adobe Acrobat Reader. Sometimes the contents of a slide is replaced by a big black box that disappears when zooming in or out. This seems to be a PROSPER-style related bug and the trick is either to rewrite properly the style or to choose another one.

Other Bugs

You may browse for a list of identified bugs and sometimes workaround solutions on the PROSPER SourceForge repository:

http://sourceforge.net/tracker/?group_id=14812&atid=114812

4 | TEXPower

Contents

4.1 Introduction

The TEXPower bundle contains style and class files for creating dynamic online presentations with LATEX.

The heart of the bundle is the package `texpower.sty` which implements some commands for presentation effects. This includes setting page transitions, color highlighting and displaying pages incrementally.

The package TEXPower is completely independent of the document class used and the method of PDF creation.

Beware: If you use the teTEX or the MikTEX distributions, the class `foils.cls` is not part anymore of them because it has a non-free license from IBM Research Center.

4.1.1 Example

The following code shows a typical usage of the class (*see also* Figures 4.1, 4.2 and 4.3).

```
\documentclass[landscape,a4paper]{foils}

\usepackage{fixseminar}
\usepackage[display]{texpower}

\title{The \code{texpower} / {\normalfont \texttt{foils} Demo}}
\author{Stephan Lehmke \\
        \code{mailto:Stephan.Lehmke@cs.uni-dortmund.de}}

\begin{document}

\maketitle

\foilhead{A list environment}

\pause

\stepwise
{
  \begin{description}
    \item[foo.] \step{bar.}
    \step{\item[baz.]} \step{qux.}
  \end{description}
}

\foilhead{An aligned equation}

\pause

\parstepwise
{
  \begin{eqnarray}
    \sum_{i=1}^{n} i & \step{=} & \restep{1 + 2 + %
      \cdots + (n-1) + n}\\
    & \step{=} & \restep{1 + n + 2 + (n-1) + \cdots}\\
    & \step{=} & \restep
        {
          \switch
              {
                \vphantom{\underbrace{(1 + n) +
                    \cdots + (1 + n)}_{\times\frac{n}{2}}}%
                (1 + n) + \cdots + (1 + n)%
              }
              {\underbrace{(1 + n) + \cdots +
                  (1 + n)}_{\times\frac{n}{2}}}%
        }
        \\
        & \step{=} & \restep{\frac{(1 + n)\step{{} %
              \cdot n}}{\restep{2}}}
  \end{eqnarray}
}

\end{document}
```

4.1.2 Options of the Class

General options

display enables "dynamic" features. If not set, it is assumed that the document is to be printed, and all commands for dynamic presentation have no effect.

printout (default) disables "dynamic" features.

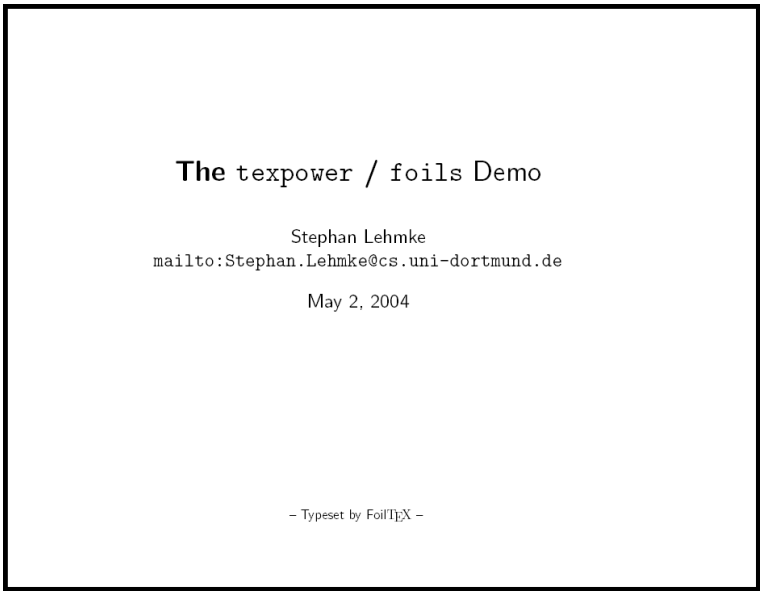


Figure 4.1: TeXPower: Title of a presentation.

Color options

whitebackground (default) sets standard colors (*see* § 4.5) to match a white background color.

lightbackground sets standard colors to match a light (but not white) background color.

darkbackground sets standard colors to match a dark (but not dark) background color.

blackbackground sets standard colors to match a black background color.

colorhighlight enables highlight of item (*see* § 4.7.1).

colormath colors *all* mathematical formulae.

coloremph makes `\em` and `\emph` switch colors instead of fonts.

4.2 Preamble

In the so-called section *preamble* between `\documentclass` and `\begin{document}`, you should specify the contents of the title page and some options (such as logos and slide captions) that can be applied to all slides. The normal LaTeX macros have been redefined to generate the title and associated texts with proper font sizes, etc. Some of the macros available for designing the title slide include:

`\title` Defines the title of the presentation.

`\author` Defines the author of the presentation.

`\MyLogo` Puts a logo or whatever you wish at the bottom of each slide, replacing the standard message "Typeset by FoilTEX".

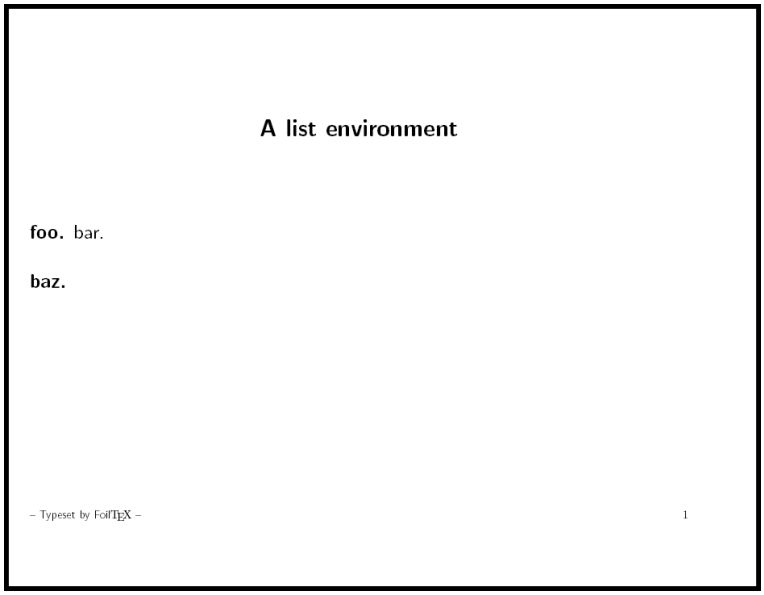


Figure 4.2: TeXPower: A list displayed incrementally.

`\Restriction` Was included in case you want to have each slide identified for a particular audience (e.g., "Confidential").

By design, the footline consists of the contents of `\MyLogo` followed by the contents of `\Restriction` all left justified, with the page number right justified.

As in standard LaTeX, the title slide is generated by the `\maketitle` command in the document body. Your preamble should at least look that this:

```
1  \documentclass[landscape,a4paper]{foils}
2
3  \usepackage{color}
4  \usepackage{hyperref}
5  \usepackage{soul}
6  \usepackage{fixseminar}
7  \usepackage[display]{texpower}
8
9  \title{Title of my Presentation}
10 \author{Your Name}
11 \MyLogo{Copyright text or logo}
12
13 \begin{document}
```

Description of the Packages Above

color to use colors in the presentation.

hyperref is neccessary for page transition effects to work (*see* § 5.1). In particular, the `\pageDuration` (*see* § 5.2.1) command only works if the version of `hyperref` loaded is at least v6.70a (where the key `pdfpageduration` was introduced).

soul is neccessary for the implementation of the commands `\hidetext` and `\highlighttext`.

fixseminar unfortunately, there are some fixes to `seminar` which can not be applied in the TEXPower packages because they have to be applied

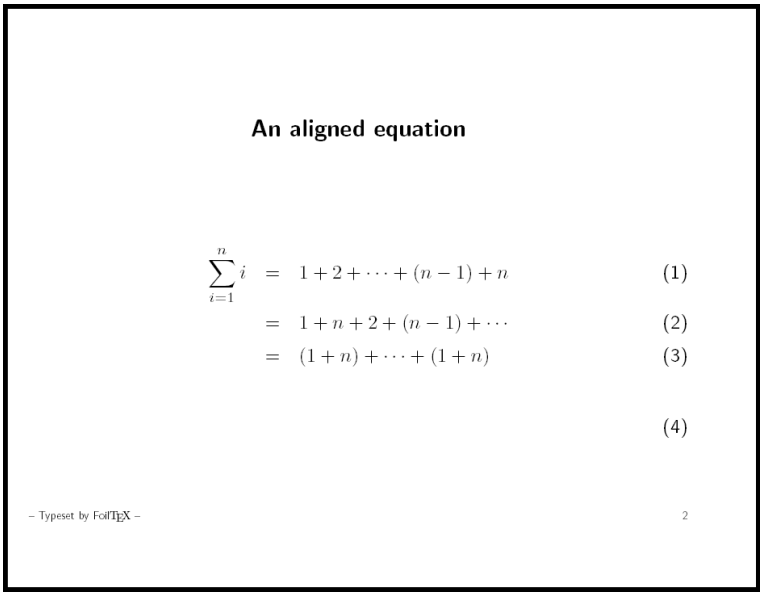


Figure 4.3: TeXPower: A mathematical equation displayed incrementally.

after `hyperref` is loaded (if this package should be loaded). The package `fixseminar` applies these fixes, so this package should be loaded after `hyperref` (if `hyperref` is loaded at all, otherwise `fixseminar` can be loaded anywhere in the preamble).

4.3 Toggling the Logo

There is a better method than undefining/redefining `\MyLogo` for inhibiting a logo from appearing on selected slides or all slides. This feature is implemented with two switches. These macros are `\LogoOn` and `\LogoOff` and they do exactly what their names imply. If `\LogoOff` appears before the footer is processed by the output routine, no logo will appear (as if `\MyLogo{}` were declared). This stays in effect until `\LogoOn` is encountered, at which point the contents of `\MyLogo` are restored. So, for instance, if you do not want the logo to appear at all, you can put the `\LogoOff` command before the `\begin{document}` command. If you want the logo only on the title page, then you can put this command after the first occurence of `\foilhead`. You can then turn the logo back on by putting the `\LogoOn` command in a convenient place.

4.4 The Other Three Corners

Since the macros `\Restriction` and `\MyLogo` control the bottom left corner of the page, there are other macros for putting text in the other three corners. These are,

- `\rightheader{`⟨*text*⟩`}`
- `\leftheader{`⟨*text*⟩`}`
- `\rightfooter{`⟨*text*⟩`}`

They each take one argument, the text you want to place in the associated corner of the page. These can also be redeclared within the document

with the appropriate attention paid to the output routine. By default the headers are empty and the lower right footer is just the page number:

- `\rightheader{}`
- `\leftheader{}`
- `\rightfooter{\quad\textsf{\thepage}}`

except on the title page where they are all suppressed. You can easily suppress page numbering by declaring `\rightfooter{}`.

4.5 Standard Colors

TEXPower maintains a list of *standard colors* which are recognized and handled by TEXPower's color management. Some commands like `\dimcolors` affect all standard colors.

`\defineTPcolor{⟨name⟩}{⟨model⟩}{⟨def⟩}` Acts like `\definecolor` from the `color` package, but the color ⟨*name*⟩ is also added to the list of standard colors.

`\addTPcolor{⟨name⟩}` Adds the existing color ⟨*name*⟩ to the list of standard colors.

4.5.1 Slide's Background Color

As other packages, TEXPower allows you change the background color. You may choose a single solid color or one of the provided gradient method. The way the background is displayed depends on the color option you set in the package declaration (*see* § 4.1.2).

`\backgroundstyle[⟨options⟩]{⟨style⟩}` Is the central command for structured page backgrounds. It works like `\pagestyle` and other commands of this type. This means ⟨*style*⟩ is a symbolic name specifying the general method by which the page background is constructed.

The detailed construction is influenced by parameters which can be set in ⟨*options*⟩, which should be a comma-separated list of ⟨*key*⟩=⟨*value*⟩ pairs.

Using a Gradient Background

⟨*style*⟩ may have one of the following values[1]:

vgradient Vertical gradient. The page background is constructed using the `\vgradrule` command. If there are panels (, the gradient rule fills the rectangular space left between the specified panels.

Options: stripes, startcolor, endcolor.

hgradient Horizontal gradient. The page background is constructed using the `\hgradrule` command.

doublevgradient Double vertical gradient. The page background is constructed using the `\dblvgradrule` command.

Options: stripes, startcolor, midcolor, endcolor.

[1] Only a few options are listed here. *See* [11] for a full specification.

doublehgradient Double horizontal gradient. The page background is constructed using the `\dblhgradrule` command.

E.g., `\backgroundstyle[stripes=25,startcolor=red]{vgradient}`

4.6 Panels

Panels can be added to your slides anchored either at left, right, top or bottom, as shown on Figure 4.4.

```
1  \documentclass[a4paper,landscape]{foils}
2
3  \usepackage{graphicx,color,hyperref,soul,fixseminar}
4  \usepackage[darkbackground,display]{texpower}
5
6  \title{First Presentation}
7  \author{Xavier Pers\'eguers}
8  \LogoOff
9  \rightfooter{}
10
11 \begin{document}
12
13 \DeclarePanel{right}{%
14   My Panel
15
16   %\hfill % if panel is at top/bottom
17   \vfill  % if panel is at left/right
18
19   \button{\Acrobatmenu{PrevPage}}{Back}
20   \button{\Acrobatmenu{NextPage}}{Next}
21 }
22
23 \backgroundstyle[stripes=50,startcolor=blue,%
24                  endcolor=black]{vgradient}
25
26 \maketitle
27
28 \end{document}
```

Beware: If you use panels and a gradient background, you have to declare the gradient *after* the panel.

The command `\button` takes four optional arguments which are left out in the example below. These are ⟨*width*⟩, ⟨*height*⟩, ⟨*depth*⟩ and ⟨*alignment*⟩ in that order. If given, ⟨*width*⟩, ⟨*height*⟩, ⟨*depth*⟩ set the dimensions of the framed area comprising the button, without the shadow. If specified, the optional parameter ⟨*alignment*⟩ (one of **l**, **c**, **r**) gives the alignment of the text inside the button box. *See* 6.4.3 for a list of available buttons you may add to your panels.

4.7 Overlays

As there exists no concept of "slide" in TEXPower as this was the case with BEAMER or PROSPER, the concept of overlays is a bit easier with TEXPower.

`\pause` Ships out the current page, starts a new page and copies whatever was on the current page onto the new page, where typesetting is resumed. This creates the effect of a *pause* in the presentation. This

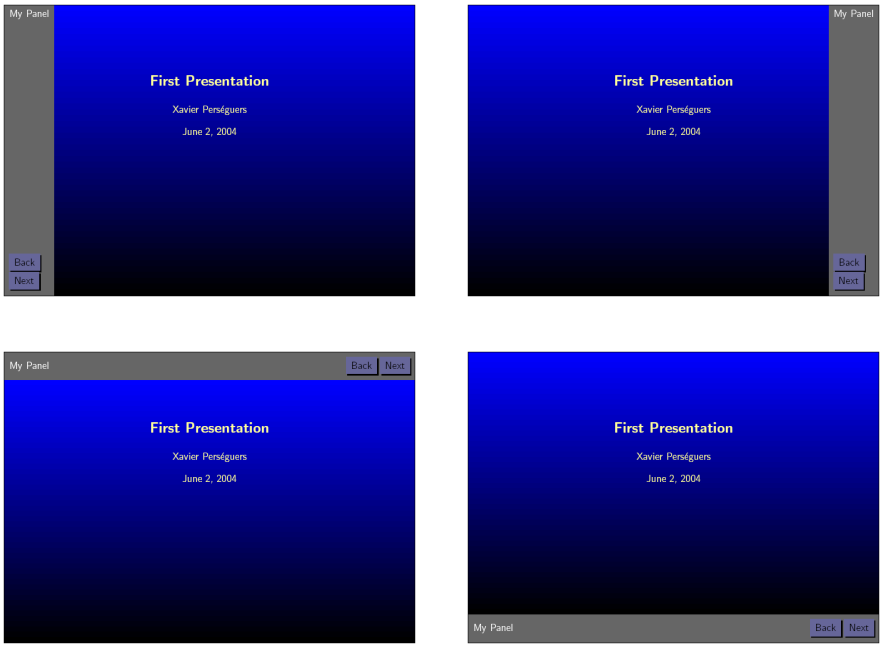


Figure 4.4: Panels (left, right, top and bottom) with TEXPower.

command *must* be used only between paragraphs or at places where ending the current paragraph does not hurt.

Example:

```
\foilhead{A Slide with Pause}

Formula below tells us how we may get energy
out of water.

$e = \alpha_2 ...$

\pause

\textbf{Proof:} As said in ...
```

`\stepwise{`⟨*contents*⟩`}` Is a command for displaying ⟨*contents*⟩ "step by step". This command *must* be used only between paragraphs or at places where ending the current paragraph does not hurt.

If you wish the first page of a sequence produced contains not only material which is *not* part of any ⟨*stepcontents*⟩, you may use the command `\stepwise*`.[2]

Usually, ⟨*contents*⟩ contains the following command too:

`\step[`⟨*activatefirst*⟩`][`⟨*whenactive*⟩`]{`⟨*stepcontents*⟩`}` Is a command used within the contents of a `\stepwise` command (*see* above). Without the two optional arguments ⟨*activatefirst*⟩ and ⟨*whenactive*⟩, the behaviour of this command is:

- As many pages as there are `\step` commands in ⟨*contents*⟩ are produced;

[2] All variants of `\stepwise` have a starred version too which also shows the first step of a sequence on the first page.

- Every page starts with what was on the current page when `\stepwise` started;
- The first page also contains everything in ⟨*contents*⟩ which is not in ⟨*stepcontents*⟩ for any `\step` command;
- The second page additionally contains the ⟨*stepcontents*⟩ for the first `\step` command, and so on, until all ⟨*stepcontents*⟩ are displayed;
- When all ⟨*stepcontents*⟩ are displayed, `\stepwise` ends and typesetting is resumed on the current page.

Use of the optional parameters. These should be conditions in the syntax of the `\ifthenelse` (*see* Listings 4.9.1 and 4.9.2 for examples of use).

⟨*activatefirst*⟩ checks whether this `\step` is to be activated *for the first time*. The default is

`\value{step} = \value{stepcommand}`.

If one uses `\value{step} =` ⟨*n*⟩, this `\step` can be forced to appear as the ⟨*n*⟩th one.

⟨*whenactive*⟩ checks whether this `\step` is to be considered active *at all*. The default behaviour is to check whether this `\step` has been activated before (this is saved internally for every step).

For more information on advanced programming steps, *see* [11].

`\steponce` Like `\step`, but goes inactive again in the subsequent step.

Hint: Combining `\stepwise*` and `\steponce` lets you create a step-by-step sequence where each step *replaces* the previous one instead of being shown *after* it.

Example:

```
\stepwise*{
  \steponce{This text}
  \steponce{is changing}
  \steponce{over time}
}
```

`\liststepwise{`⟨*contents*⟩`}` Works exactly like `\stepwise`, but should be used for list environments and aligned equations.

Example:

```
\liststepwise*{
  \begin{itemize}
    \step{\item A list}
    \steponce{\item with an item\ldots}
    \step{\item replaced by}
    \step{something else} % continues the previous item
    \step{\item that's all}
  \end{itemize}
}
```

`\parstepwise{`⟨*contents*⟩`}` Works like `\liststepwise`, but `\boxedsteps` (*see* § 4.8.2) is turned on by default. Use for texts where `\steps` are to be filled into blank spaces.

Example:

```
\parstepwise{
  \step{A paragraph of text.}

  \step{Another paragraph of text.}
}
```

4.7.1 Changing the way ⟨*stepcontents*⟩ is displayed

You should think of `\step` as a command replaced by

`\hidestepcontents{`⟨*stepcontents*⟩`}` When this step is not yet active.

`\displaystepcontents{\activatestep{`⟨*stepcontents*⟩`}}` When this step is activated *for the first time*.

`\displaystepcontents{`⟨*stepcontents*⟩`}` When this step has already been activated before.

You may redefine these macros to change the behaviour. Simply use an instruction of the form `\let`⟨*stepcommand*⟩=⟨*action*⟩. where ⟨*stepcommand*⟩ is one of `\hidestepcontents`, `\displaystepcontents` or `\activatestep` and ⟨*action*⟩ may be:

Options for `\displaystepcontents`

`\displayidentical` Simply expands to its argument. this is used by default by `\activatestep`.

`\displayboxed` Expands to an `\mbox` containing its argument. This is used by `\boxedsteps` for interpreting `\hidestepcontents`

`\hideignore` Expands to nothing. Used by `\nonboxedsteps`.

`\hidephantom` Expands to a `\phantom` containing its argument. Used by `\boxedsteps`.

`\hidevanish` In a colored document, makes its argument "vanish" by setting all colors to `\vanishcolor`[3].

`\hidetext` Produces blank space of the same dimensions as the space that would be taken if its argument would be typeset in the current paragraph. Respects automatic hyphenation and line breaks.

`\hidedimmed` In a colored document, displays its argument with dimmed colors. This does not make the argument invisible.

[3] It is set to `\pagecolor` by default.

Options for `\activatestep`

`\highlightboxed` If the `colorhighlight` package option is set, expands to a box with colored background containing its argument.

`\highlighttext` If the `colorhighlight` package option is set, expands to its argument typeset on a colored background.

`\highlightenhanced` In a colored document (`color` is set as a package option), displays its argument with enhanced colors.

4.7.2 `\boxedsteps` and `\nonboxedsteps`

By default, ⟨*stepcontents*⟩ part of a `\step` which is not yet "active" are ignored. This allows to include layout specifications such as tabulators (in tables) or line breaks into ⟨*stepcontents*⟩. The presentation of the text may hence vary over time, when ⟨*stepcontents*⟩ become "active".

Sometimes, this behaviour is undesirable as when filling in blanks in a paragraph. In such cases, the desired behaviour is to replace not yet "active" `\step` items with an appropriate amount of blank space. There exists two commands to activate/deactivate this feature:

`\boxedsteps` Makes `\step` create a blank box the size of ⟨*stepcontents*⟩ when inactive.

Example:

```
\boxedsteps
\stepwise{
  A text with \step{something} appearing later\ldots
}
```

`\nonboxedsteps` Makes `\step` ignore ⟨*stepcontents*⟩ when inactive (default).

4.8 How do I. . .

4.8.1 How can I incrementally display a paragraph of text?

Problem: There is a slighty different baseline alignment of the texts displayed incrementally.

Solution: The easiest solution is to use `\parstepwise`, but if the arguments of `\step` are long, you'll get problems with line breaks, as the command `\parstepwise` forces `\step` to put its argument in a box. You can use `\hidetext` like this:

```
\stepwise[\let\hidestepcontents=\hidetext]
{\step{Line breaks} \step{work in here.}}
```

But note that `\hidetext`, being implemented using the `soul` package, is quite fragile[4].

If you are not using structured backgrounds, `\hidevanish` is another alternative which can be used exactly like `\hidetext`, but is much more robust (note that this will fail whenever your text should appear in front of different background colors, for any reason).

In the argument of `\hidevanish`, which uses `\textcolor`, paragraph breaks are not allowed.

[4] Fragile commands, *see* § C.2, p. 73.

4.8.2 How can I incrementally display a table?

Problem: Cells may grow if future contents is widder than the visible contents.

Solution: Ths most robust way of doing this is to create an empty box with the same dimensions as the text to be hidden. Use the command `\boxedsteps` to make `\step` create a blank box the size of ⟨*paramcontents*⟩ when inactive and put ⟨*stepcontents*⟩ into a box when active. The dual command is `\nonboxedsteps` to activate the default behaviour.

```
\stepwise{%
  \boxedsteps
  \begin{tabular}{ll}
    \hline
          1  &       one\\
    \step{2} & \step{two}\\
    \step{3} & \step{three}\\
    \hline
  \end{tabular}%
}
```

4.9 Useful Macros

4.9.1 Incremental Highlight

The *incremental highlight* is a way to step through an enumeration of items and displaying in another color (or *highlighting*) each item as it is introduced.

Requirements: package `ifthen`
File name: `texpower-highlight.tex`
Source: taken from the fulldemo in the TEXPower package

```
% Example:
% \liststepwise*[\let\hidestepcontents=\displaystepcontents]
%{%
%   \begin{stepitemize}
%   \item Item 1
%   \item Item 2
%   \item Item 3
%   \end{stepitemize}
%}
%
% As the highlighting is done by \mystep, we define
% \hidestepcontents to also display its argument, so that all
% items are visible from the outset.
%
% Note that we use the starred version of \liststepwise so that
% the first item is highlighted on the first slide produced by
% \liststepwise.
%
\let\originalitem=\item
\def\myitem{\originalitem[\color{magenta}$\star$]}
%
% We define a macro \mystep which implements the highlighting
% effect.
%
\def\mystep% Note that \mystep takes no argument
{%
  \step
  {%
    \ifthenelse{\boolean{display}}
```

```
      {%
        \ifthenelse{\boolean{firstactivation}}
        {\color{conceptcolor}}%
        {\color{inactivecolor}}
      }{}%
    }%
}%
%
% We define a custom itemize environment which calls to \mystep:
%
\newenvironment{stepitemize}
{%
  \begin{itemize}
    \let\origitem=\item
    \let\origmyitem=\myitem
    % Here, the \mystep command is hidden inside \item
    \renewcommand{\item}{\mystep\origitem}%
    \renewcommand{\myitem}{\mystep\origmyitem}%
}
{%
  \end{itemize}
}
```

4.9.2 Incremental Highlight with Permanent Color Change

This second version of the incremental highlight will display non yet active items using a dimmed color, when they become active for the first time, using a highlight color and using another color when they have been activated.

Requirements: package `ifthen`
File name: `texpower-highlight2.tex`
Source: transformed from the fulldemo in the TEXPower package

```
% Example:
% \liststepwise*[\let\hidestepcontents=\shadowstepcontents]{%
% \begin{mystepitemize}
%   \item Item 1
%   \item Item 2
%   \item Item 3
% \end{mystepitemize}}
%
% As the highlighting is done by \mystep, we define
% \hidestepcontents to also display its argument, so that all
% items are visible from the outset.
%
% Note that we use the starred version of \liststepwise so that
% the first item is highlighted on the first slide produced by
% \liststepwise.
%
\let\originalitem=\item
\def\myitem{\originalitem[\color{magenta}$\star$]}
%
% We define a macro \mystep which implements the highlighting
% effect.
%
\def\shadowstepcontents{\color{inactivecolor}\displaystepcontents}
\def\mynewstep% Note that \mystep takes no argument
{%
  \step
  {%
    \ifthenelse{\boolean{display}}
    {%
      \ifthenelse{\boolean{firstactivation}}
```

```
        {\color{conceptcolor}}%
        {}
      }{}%
    }%
}%
%
% We define a custom itemize environment which calls to \mystep:
%
\newenvironment{mystepitemize}
{%
  \begin{itemize}
    \let\origitem=\item
    \let\origmyitem=\myitem
    % Here, the \mystep command is hidden inside \item
    \renewcommand{\item}{\mynewstep\origitem}%
    \renewcommand{\myitem}{\mynewstep\origmyitem}%
}
{%
  \end{itemize}
}
```

4.10 Identified Bugs

You may browse for a list of identified bugs and sometimes workaround solutions on the TEXPower SourceForge repository:

http://sourceforge.net/tracker/?group_id=60743&atid=495145

5 | Presentation Features

Contents

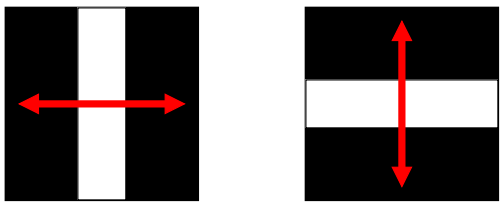
5.1 Adding Slide Transitions

In the following, ⟨*trans*⟩ is a reference to the transition style to use when moving *to* the page *from another* during a presentation. You might be unable to customize the way the transition is performed, according to the presentation package you use. In this case, you will have to use the `\special` tag as described in § 5.1.2 to customize the transition.

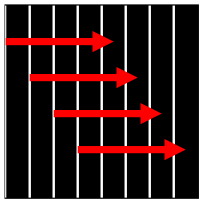
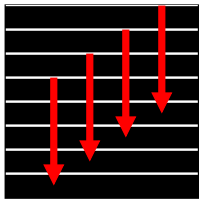
5.1.1 Available Transitions

R: *default option.* The new page simply replaces the old one with no special transition effect.

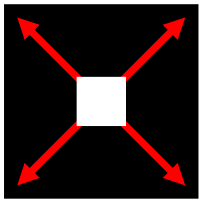
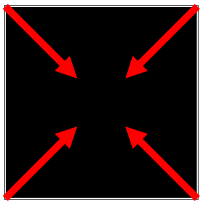
Split: two lines sweep across the screen, revealing the new page. The lines may be either horizontal or vertical and may move inward from the edges of the page or outward from the center.



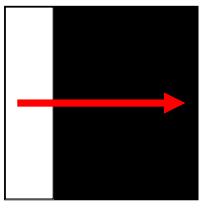
Blinds: multiple lines, evenly spaced across the screen, synchronously sweep in the same direction to reveal the new page. The lines may be either horizontal or vertical. Horizontal lines move downward, vertical lines to the right.

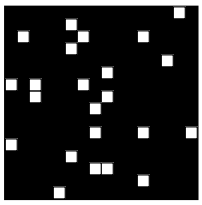
Box: a rectangular box sweeps inward from the edges of the page or outward from the center, revealing the new page.

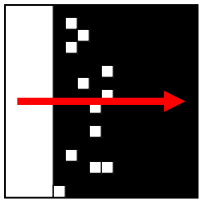
Wipe: a single line sweeps across the screen from one edge to the other.



Dissolve: the old page dissolves gradually to reveal the new one.



Glitter: similar to Dissolve, except the effect sweeps across the page in a wide band moving from one side of the screen to the other in a given direction.



5.1.2 Advanced Options

According to [9, § 8.3.3], there exists several options to customize the transitions. If you know what you do, you may include your own PDF tags using the `\special` LaTeX tag:

```
\special {ps: /pdfmark where {pop} {userdict /pdfmark
/cleartomark load put} ifelse [ {ThisPage} << /Trans << /S
/Wipe /D 5 /Di 90 >> >> /PUT pdfmark }
```

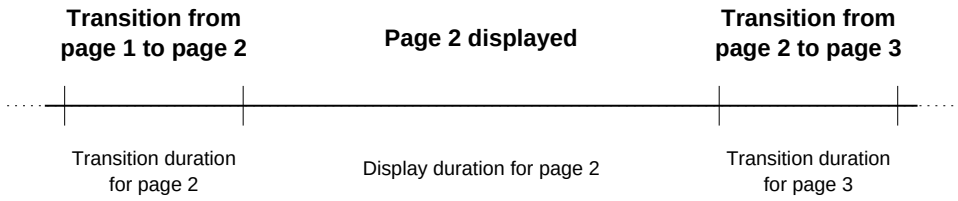


Figure 5.1: Presentation timing

Parameters

D *Optional.* The duration of the transition effect, in seconds. Default value: 1.

S *Optional.* The *transition style* to use when moving to this page from another during a presentation. Values are listed in § 5.1.1.

Dm *Optional; Split and Blinds transition styles only.* The dimension in which the specified transition effect occurs:

H Horizontal
V Vertical

Default value: H.

M *Optional; Split and Box transition styles only.* The direction of motion for the specified transition effect:

I Inward from the edges of the page
O Outward from the center of the page

Default value: I.

Di *Optional; Wipe and Glitter transition styles only.* The direction in which the specified transition effect moves, expressed in degrees counterclockwise starting from a left-to-right direction. Only the following values are valid:

0 Left to right
90 Bottom to top (Wipe only)
180 Right to left (Wipe only)
270 Top to bottom
315 Top-left to bottom-right (Glitter only)

Default value: 0.

Figure 5.1 illustrates the relationship between transition duration (**D**) and display option (**Dur**, *see* § 5.2.1). Note that the transition duration specified for a page (page 2 in the figure) governs the transition *to* that page from another page; the transition *from* the page is governed by the next page's transition duration.

5.1.3 Beamer

Transition commands in BEAMER are inside `frame` environment.

`\transsplithorizontalout<`⟨*duration*⟩`>` Splits horizontally to the outside. ⟨*duration*⟩ is a number specifying the duration in seconds. Default is 1 sec.

`\transsplithorizontalin`<⟨*duration*⟩> Splits horizontally to the inside. ⟨*duration*⟩ is a number specifying the duration in seconds. Default is 1 sec.

`\transsplitverticalout`<⟨*duration*⟩> Splits vertically to the outside. ⟨*duration*⟩ is a number specifying the duration in seconds. Default is 1 sec.

`\transsplitverticalin`<⟨*duration*⟩> Splits vertically to the inside. ⟨*duration*⟩ is a number specifying the duration in seconds. Default is 1 sec.

`\transblindshorizontal`<⟨*duration*⟩> Horizontal blinds. ⟨*duration*⟩ is a number specifying the duration in seconds. Default is 1 sec.

`\transblindsvertical`<⟨*duration*⟩> Vertical blinds. ⟨*duration*⟩ is a number specifying the duration in seconds. Default is 1 sec.

`\transboxout`<⟨*duration*⟩> Growing box. ⟨*duration*⟩ is a number specifying the duration in seconds. Default is 1 sec.

`\transboxin`<⟨*duration*⟩> Shrinking box. ⟨*duration*⟩ is a number specifying the duration in seconds. Default is 1 sec.

`\transwipe`[direction=⟨*angle*⟩] Wipes from one edge of the page to the facing edge. ⟨*angle*⟩ is a number between 0 and 360 which specifies the direction (in degrees) in which to wipe.
According to [9, § 8.3.3], only the values 0, 90, 180 and 270 are supported.

`\transdissolve`<⟨*duration*⟩> Dissolves. ⟨*duration*⟩ is a number specifying the duration in seconds. Default is 1 sec.

`\transglitter`[direction=⟨*angle*⟩] Glitters from one edge of the page to the facing edge. ⟨*angle*⟩ is a number between 0 and 360 giving the direction (in degrees) in which to glitter.
According to [9, § 8.3.3], only the values 0, 270 and 315 are supported.

Auto-Advancing

You may use the following command to animate a range of slide at a given period. Example below animates next slides at every 0.5 sec. *See also* § 2.3.5, p. 9.

```
\transduration<2->{0.5}
```

5.1.4 Prosper

Default Transition

To specify the default transition to be used for all slides of your presentation, use the command `\DefaultTransition`{⟨*trans*⟩}.

Transition for a given slide

You may specify a transition to use for a given slide with the parameter ⟨*trans*⟩ of the slide's definition: `\begin{slide}`[⟨*trans*⟩]{⟨*title*⟩}

Advanced Transition Effects

Insert a `\special` tag as described in § 5.1.2 after the definition of a slide, without any transition parameter.

5.1.5 TEXPower

The commands below work only if the `hyperref` package is loaded.

`\pageTransitionSplitHO` Splits horizontally to the outside.

`\pageTransitionSplitHI` Splits horizontally to the inside.

`\pageTransitionSplitVO` Splits vertically to the outside.

`\pageTransitionSplitVI` Splits vertically to the inside.

`\pageTransitionBlindsH` Horizontal blinds.

`\pageTransitionBlindsV` Vertical blinds.

`\pageTransitionBoxO` Growing box.

`\pageTransitionBoxI` Shrinking box.

`\pageTransitionWipe{`⟨*angle*⟩`}` Wipes from one edge of the page to the facing edge. ⟨*angle*⟩ is a number between 0 and 360 which specifies the direction (in degrees) in which to wipe.
According to [9, § 8.3.3], only the values 0, 90, 180 and 270 are supported.

`\pageTransitionDissolve` Dissolves.

`\pageTransitionGlitter{`⟨*angle*⟩`}` Glitters from one edge of the page to the facing edge. ⟨*angle*⟩ is a number between 0 and 360 giving the direction (in degrees) in which to glitter.
According to [9, § 8.3.3], only the values 0, 270 and 315 are supported.

`\pageTransitionReplace` Simple replace (default).

Remark: Setting page transitions works well with `\pause` but not with `\step`. Here, `\pause` acts as a page break, i.e. a different page transition can be set before every occurence of `\pause`.

5.2 Additional PDF Presentation Features

5.2.1 Presentation Mode

Some PDF viewer application may allow a document to be displayed in the form of a *presentation* or "slide show", advancing from one page to the next either automatically or under user control.

Windows XP	
Version	FPS
4.05	60.3
5.05	19.7
5.1	19.8
6.0.1	15.4

Linux 2.4.26	
Version	FPS
5.08	5.63

Table 5.1: Adobe Acrobat Reader's Performances on a Pentium IV 1.8 GHz, 512 MB RAM

Usage with Package `hyperref`

`hyperref` provides a command to specify the duration for a given slide of the presentation. This duration n is expressed in seconds and should be specified right after the start of the corresponding slide in the presentation package you use[1].

```
\hypersetup{pdfpageduration=n}
```

Advanced Usage

The **Dur** entry in the page object specifies the page's *display duration* (also called *advance timing*): the maximum length of time, in seconds, that the page will be displayed before the presentation automatically advances to the next page. (The user can advance the page manually before the specified time has expired.) Example below shows how you may specify a timeout of 4 seconds before automatically advancing to the next slide.

```
\special {ps: /pdfmark where {pop} {userdict /pdfmark
/cleartomark load put} ifelse [ {ThisPage} << /Dur 4 >>
/PUT pdfmark }
```

Performances

Table 5.1 shows the performances of a few Adobe Acrobat Reader versions with automatically advancing a huge document containing only text with a timeout of 0 seconds, which means *as fast as possible*.

[1] BEAMER: inside a `frame` definition but before any contents.

6 Creation of the PDF Files

Contents

6.1 To pdflatex or Not?

There exists several ways to produce a PDF file out of a TEX file:

1. `pdflatex` *file.tex*
2. `latex` *file.tex*
   `dvipdf` *file.dvi*
3. `latex` *file.tex*
   `dvipdfm` *file.dvi*
4. `latex` *file.tex*
   `dvips` *file.dvi* `-o`
   `ps2pdf` *file.ps*

Method 1: One of the largest advantages of pdfLATEX (a.k.a. pdfTEX [1]) over the conventional LATEX is the fact that `pdflatex` merges easily pictures in PDF, PNG, JPEG or TIFF format instead of the standard EPS format. However, it cannot deal with typesetting involving the package `pstricks`. As the other methods support all PDF features *and* `pstricks`, you should consider using them (however, package Beamer is not well suited for compiling files with other method than `pdflatex`).

[1] Actually, pdfTEX is the real name but as you compile your TEX file using `pdflatex`, we will only refer to this method using the less generic name pdfLATEX.

LaTeX and pdfLaTeX behave by default differently concerning the character-spacing. As a result, the same TeX file compiled with LaTeX and pdfLaTeX may show different line-breaks, paragraphs, page-breaks, ... Fortunately, there is a switch in pdfLaTeX to ensure LaTeX spacing.[2]

Methods 2 – 4: These are the usual ways to create PDF file.

- `dvipdf` essentially runs `dvips` then pumps that output through `ps2pdf`, a script which uses Ghostscript to generate PDF output. You may add security as if you used `ps2pdf` (*see* § 6.3);
- `dvipdfm` is written from the ground-up to create PDF directly from DVI files and generally does a better job;
- Method 4 allows you to add security to your PDF files (*see* § 6.3). In fact, the PDF output will be slightly smaller than with `dvipdf`.

6.1.1 I still want to use `pdflatex` & `pstrick`!

psTricks macros cannot be used with `pdflatex` since psTricks uses PostScript arithmetic, which is not in PDF. As such, a package pdfTricks has been written to circumvent this limitation, so that the extensive facilities offered by the powerful psTricks package can be made use of in a `pdflatex` document. This is brought by making use of the shell escape function available in the web2c TeX compiler,[3] while this package is of no use to other commercial implementations.

Shell escape allows this package to suspend a TeX compilation whenever a psTricks block is encountered, to compile the corresponding block into an EPS picture, to convert it to PDF format (needed by `pdflatex`) and to return to the TeX compilation and finish the job.

http://sarovar.org/projects/pdftricks/

6.2 Producing Nice Looking PDF

A problem that might appear when producing PDF documents is that they are rendered to the screen using ugly bitmapped (type 3) fonts instead of vectorial or *outline* (type 1) fonts (*see* Figure 6.1).

> *"Bitmapped fonts are to typesetting what punch-card machines are to digital storage. They were necessary at one time, when no other viable technology was available, but they have long since been made obsolete. That they are still the default... is at best a sign of laziness and conservatism among the LaTeX crowd, or at worst an inept expression of adoration for Knuth."*
>
> Gordon Kindlmann

Explanation: The default installation of dvips uses fonts with a fixed resolution (.pk fonts) encoded as 300dpi (dots per inch) bitmaps. This is unnoticeable for printing; however, the resulting PDF files are barely legible when scaled down to today's screen resolutions (typically 72dpi). These fonts are embedded in PostScript output as Type 3 fonts. Acrobat Distiller

[2] Include `\pdfadjustspacing=1` in the preamble.
[3] The teTeX distribution works well.

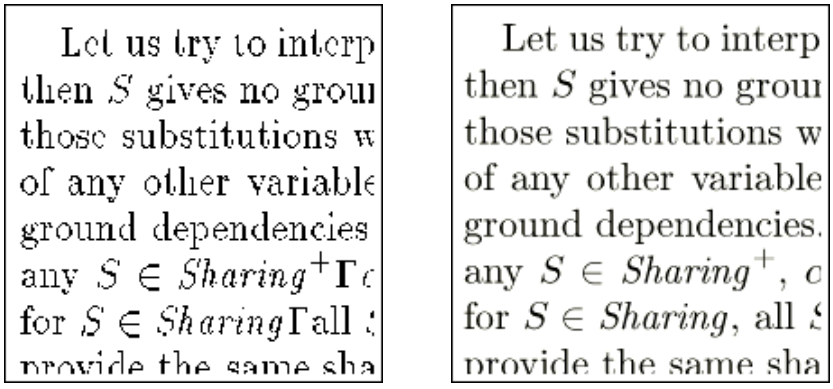


Figure 6.1: PDF using Type 3 bitmapped fonts (left) and Type 1 scalable fonts (right).

cannot handle those fonts, because there are no font descriptors available. It leaves them embedded in PDF files and renders them very badly, although printing those documents does not make too many differences, if the original resolution was high enough.
There is two solutions to fix this misbehaviour:

1. Include `\usepackage{pslatex}` in your document preamble. This is a small package that makes LaTeX default to "standard" PostScript fonts. It is basically a hacked merger of `times.sty` and `mathptm.sty`. You must have installed standard LaTeX (LaTeX 2_ε) and PSNFSS PostScript fonts to use this package.
   The main novel feature is that the pslatex package (unlike `times.sty`) tries to compensate for the visual differences between the Adobe fonts by scaling Helvetica by 90%, and "condensing" Courier (i.e. scaling horizontally) by 85%.

2. Use `dvips -Ppdf` to compile your PostScript file. This will include the PostScript (Type 1) versions of the Computer Modern and $\mathcal{AMS}$ fonts, which must be installed. They are not included by default in the teTeX distribution but they are in the MikTeX distribution.[4] In addition, you may append the `-G0` (zero) option to `dvips` that turns off dvips default behaviour (which breaks ligatures in many fonts).

NB: There is a bug in older version of `dvips` that makes `-Ppdf` flag incompatible with the `\usepackage{times}` package. If in doubt, it is safest to use the `-Pcmz -Pamz` flags to `dvips`.

6.2.1 Checking Fonts in PDF Files

One way to find out what type of fonts you've actually ended up with in a PostScript file is to open the file using Adobe Acrobat Reader. From the "File" drop down menu, select "Document Info→Fonts". Click on the "List all Fonts" button. Make sure you either scroll the resulting list or expand the popup window sufficiently to see everything that is listed. If you see "Type 3" anywhere in the list, you have done something wrong!

[4] Go to http://www.tex.ac.uk/tex-archive/systems/win32/miktex/1.20 and get the files `amsps.zip` and `cmps.zip`.

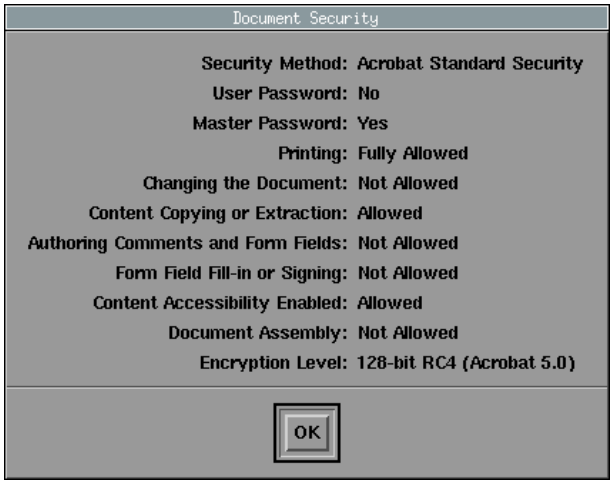


Figure 6.2: Document Security in Adobe Acrobat Reader

6.3 PDF Encryption

PDF Encryption is available with Ghostscript's macros `ps2pdf`. This allows you to restrict the use of the final PDF file. Figure 6.2 shows an example of a few restrictions applied to a document. There are two revisions of the security handler:

Revision 2: Revision 2 does not work with GS 8.14[5]. It is intended to be used with `KeyLength` equal to 40 bits and produces PDF 1.3 compliant documents; that is, intended to be used with Adobe Acrobat 4.0;

Revision 3: Revision 3 is able to use `KeyLength` up to 128 bits and produces PDF 1.4 compliant documents; that is, intended to be used with Adobe Acrobat 5.0.

6.3.1 Common Command Line Parameters

-sOwnerPassword Password allowing full non restrictive access to the document. Mandatory to use PDF restrictions.

-sUserPassword Password needed for opening the document. This parameter may be left empty.

-dEncryptionR Revision to use. May be either 2 or 3.

-dKeyLength Length of the encryption key. With revision 2, must be equal to 40, with revision 3, should be equal to 128.[6]

-dPermissions Permissions to apply to the document. See description below.

6.3.2 Revision 2

Document Security can be set with the Permissions flag. For `EncryptionR=2`, subtract these values from -4 to disable an access.

[5] http://www.ghostscript.com/doc/AFPL/index.htm

[6] With revision 3 it must actually be a multiple of 8 in the interval [40, 128].

4 = Print document
8 = Modify contents of document
16 = Copy text and graphics from document
32 = Add or modify text annotations

To allow printing and copying, but disable modifying the contents and annotations, the value is $-4-8-32$ so use `-dPermissions=-44`. To enable all, use `-dPermissions=-4`. To disable all, use `-dPermissions=-64`.

Command line

```
ps2pdf13 -sOwnerPassword=OWNER -sUserPassword=USER \
         -dEncryptionR=2 -dKeyLength=40 \
         -dPermissions=PERMISSIONS in.ps out.pdf
```

6.3.3 Revision 3

See [9, Table 3.15, p. 77] for full details of the user access permission values.

4 = Print document (possibly not at the highest quality level, depending on whether 2048 is also set)
8 = Modify contents of document, except as controlled by 32, 256 and 1024
16 = Copy text and graphics from document other than that controlled by 512
32 = Add or modify text annotations, fill in interactive form fields, and if 256 is set, create or modify interactive form fields
256 = Fill in existing interactive form fields, even if 32 is clear
512 = Extract text and graphics (in support of accessibility to disabled users or for other purposes)
1024 = Assemble the document (insert, rotate, or delete pages and create bookmarks or thumbnail images), even if 16 is clear
2048 = Print the document to a representation from which a faithful digital copy of the PDF contents could be generated. When this is clear (and 4 is set), printing is limited to a low-level representation of the appearance, possibly of degraded quality.

To enable all, use `-dPermissions=-4`. To disable everything apart from viewing, combine the following -4 (base) -4 (print) -8 (modify) -16 (copy) -32 (annotate) -256 (interactive fields) -512 (copy for disability access) -1024 (assemble) -2048 (high quality print), so `-dPermisions=-3904`.

Command line

```
ps2pdf14 -sOwnerPassword=OWNER -sUserPassword=USER \
         -dEncryptionR=3 -dKeyLength=128 \
         -dPermissions=PERMISSIONS in.ps out.pdf
```

6.3.4 Shell Script Automating the Procedure

This script allows you to quickly get the whole list of arguments with a dialog base shell script, as shown on Figure 6.3.

Requirements: sh shell, dialog and dc (an arbitrary precision calculator)
File name: `pdf_security.sh`

```
#!/bin/sh
title="PDF Security Options"

# Message Box as Greating
dialog --title "$title" \
       --shadow \
       --msgbox "This script allows you to set permissions and \
password for a PDF file created from a given PostScript file.\n\n \
  - Choose Permissions\n \
  - Enter (mandatory) Owner Password\n \
  - Enter (optional) User Password\n \
  - Use the output command line to create your PDF file." 0 0

# Permission Check-List
permissions=`
  dialog --stdout \
         --title "$title" \
         --shadow \
         --checklist 'Please select features permissions' \
         0 0 0 \
         print       'Print document (normal quality)'           on \
         modify      'Modify contents of document'               on \
         copy        'Copy text and graphics'                    on \
         annotations 'Add/modify annotations'                    on \
         forms       'Fill in interactive form fields'           on \
         extract     'Extract text and graphics (accessibility)' on \
         assemble    'Insert/rotate pages, create bookmarks'     on \
         quality     'Print document (high quality)'             on \
  |
  sed    -e 's/print/4+/'        \
         -e 's/modify/8+/'       \
         -e 's/copy/16+/'        \
         -e 's/annotations/32+/' \
         -e 's/forms/256+/'      \
         -e 's/extract/512+/'    \
         -e 's/assemble/1024+/'  \
         -e 's/quality/2048+/'   \
         -e 's/^0 3904- /'       \
         -e 's/$/p/'             \
  |
  dc`

if [ -z "$permissions" ] ; then
  permissions=-4
fi

# Ask for Owner Password
ownerpassword=`
  dialog --stdout \
         --title "$title" \
         --shadow \
         --passwordbox 'Enter the owner password (mandatory)' 0 0`

# Ask for User Password
userpassword=`
  dialog --stdout \
         --title "$title" \
         --shadow \
         --passwordbox 'Enter the user password (optional)' 0 0`

# Output command line
echo -n "ps2pdf14 -sOwnerPassword='$ownerpassword'"
if [ -n "$userpassword" ] ; then
  echo -n " -sUserPassword='$userpassword'"
fi
echo " -dEncryptionR=3 -dKeyLength=128 -dPermissions=$permissions \
in.ps out.pdf"
```

6.4 Other PDF Features

PDF documents are now the de facto standard for the secure and reliable distribution and exchange of electronic documents and forms around the

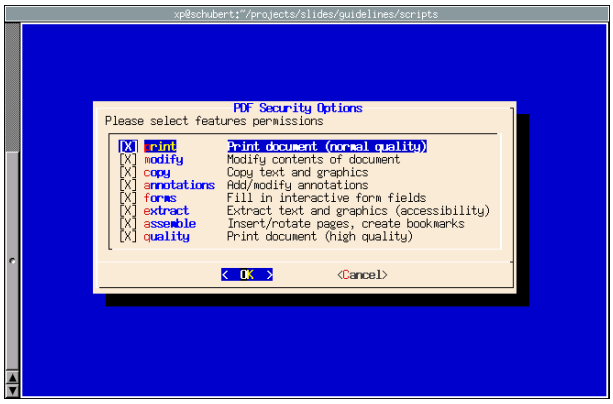


Figure 6.3: Shell script to set the security of a PDF file.

world. However creating a PDF document should not be associated as only being a solution to provide an easy way to let other people print your LaTeX documents or for you as being the solution for getting animated presentations. One should think of it as a whole entity and give it more informations than just what is "seen".

For this purpose, the PDF file format offers meta-tags for identifying and indexing the documents.

6.4.1 Document's Informations

```
\usepackage[
    pdfauthor       = {Xavier~Perseguers},
    pdftitle        = {Making~Presentations~with~LaTeX},
    pdfsubject      = {Guidelines},
    pdfcreator      = {LaTeX},
    pdfproducer     = {dvips + ps2pdf}
  ]{hyperref}
```

6.4.2 Using Common PDF Features

Adding more features like links recognition, bookmarks, thumbnails is easily achieved with other options of the package `hyperref`. We will only show some common features in the rest of this section. A full reference for this package is available at

http://www.tug.org/applications/hyperref/manual.html

Bookmarks and Links

The `hyperref` package extends the functionality of LaTeX cross references, which in turn can be converted to hyperlinks. Since `hyperref` overwrites some LaTeX commands, it is often included as the final package, to make sure that nothing can overwrite it. The `backref` or `pagebackref` options in `hyperref` create so-called back reference to the text passage, and also add links into the bibliography.

```
\usepackage[
    colorlinks      = true,
    urlcolor        = rltblue,    % \href{...}{...} external (URL)
    filecolor       = rltgreen,   % \href{...} local file
    linkcolor       = rltred,     % \ref{...} and \pageref{...}
    bookmarks       = true,
```

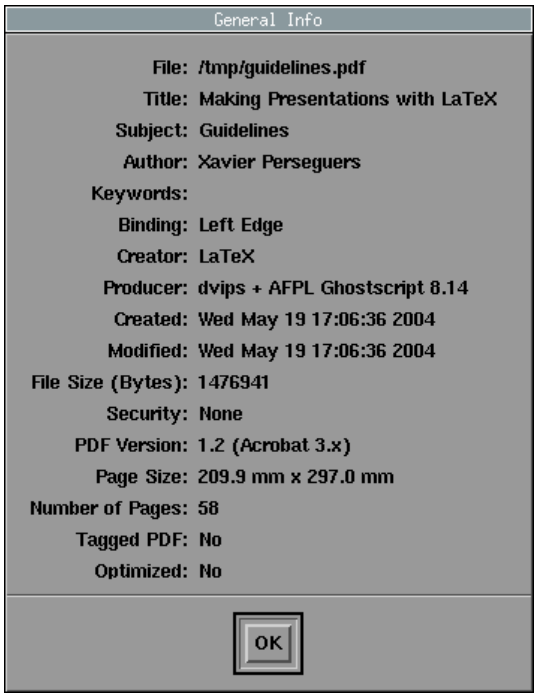


Figure 6.4: Document Informations in Adobe Acrobat Reader

```
7       bookmarksnumbered = true,
8       bookmarksopen     = true,
9       pdfpagelabels     = true
10    ]{hyperref}
11
12  \definecolor{rltred}{rgb}{0.75,0,0}
13  \definecolor{rltgreen}{rgb}{0,0.5,0}
14  \definecolor{rltblue}{rgb}{0,0,0.75}
```

Including Thumbnails

A PDF document may define *thumbnail images* representing the contents of its pages in miniature form. A viewer application can then display these images on the screen, allowing the user to navigate to a page by clicking its thumbnail image with the mouse (*see* Figure 6.6).

Recent versions of Adobe Acrobat Reader are able to automatically create thumbnails for an open document. However, these thumbnails will not be saved in the PDF file itself unless you use the full, commercial, version of Adobe Acrobat.

To create thumbnails from LaTeX, include one of the commands below in the preamble of the document, according to the method you use to produce the PDF.

```
\usepackage[ps2pdf]{thumbpdf}
\usepackage[pdftex]{thumbpdf}
```

Then, after having created a first version of the PDF file, run `thumbpdf`[7] with the name of the PDF file as argument. This will create the thumbnails.

You then have to redo the whole production process (starting from `latex`) to permanently include the generated thumbnails. Figure 6.5 shows the overall process to be done.

[7] With option `--modes=dvips` if you use dvips in your production process.

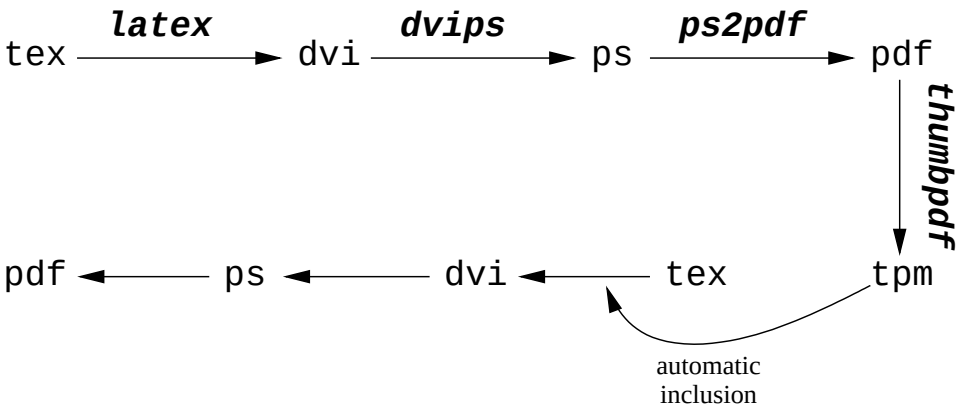


Figure 6.5: Process for including thumbnails in a PDF file.

Naming Thumbnails: The thumbnails, one for each page of the publication, are normally automatically numbered sequentially. However, you may ensure a logical numbering scheme if you use the `hyperref` package with options

```
plainpages    = false
pdfpagelabels = true
```

6.4.3 Navigation Aids

`\Acrobatmenu{`⟨*MenuOption*⟩`}{`⟨*Text*⟩`}` allows Adobe Acrobat Reader menu commands to be invoked when clicking on the element ⟨*Text*⟩ in the PDF document.

This command is provided with the package `hyperref`. Do not forget to specify the driver you use to produce the PDF file as a package option.

E.g., `\usepackage[ps2pdf]{hyperref}`

Common ⟨*MenuOption*⟩ Items

File Open, Close, Print, Quit

View ActualSize, FitVisible, FitWidth, FitPage, FullScreen, FirstPage, PrevPage, NextPage, LastPage, GoToPage, GoBack, GoForward, ShowBookmarks, ShowThumbs

Window ShowHideToolBar, ShowHideMenuBar

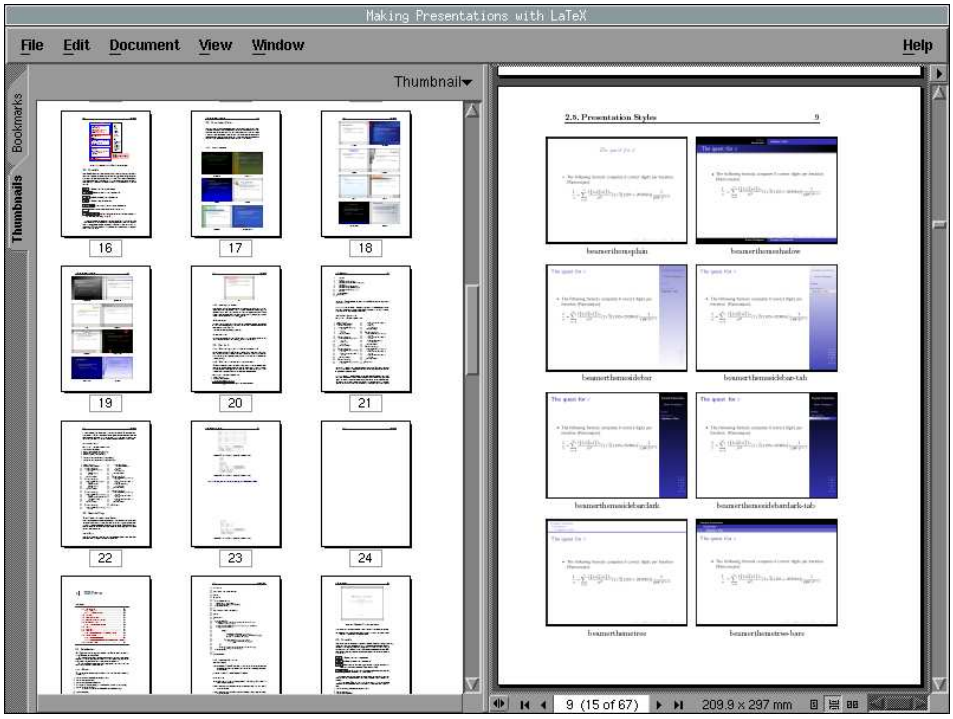


Figure 6.6: Thumbnails in Adobe Acrobat Reader.

A Common Problems Resolution

Contents

A.1 Invalid Page Format/Orientation

The PDF file is shown in portrait format while having been prepared for a landscape orientation. Figure A.1 shows the problem. This problem may occur if you use

- `latex` + `dvips` + `ps2pdf`,
- `latex` + `dvipdf` *or*
- `latex` + `dvipdfm`

If you use `pdflatex` instead, you should not get this misbehaviour.

Preliminaries

Make sure `dvips`, `pdflatex` and other compilation tools involved in a LaTeX compilation process are well configured for your system. In particular, default page format should be set to "A4". Each LaTeX distribution has its own configuration system so please consult the associated documentation.[1]

A.1.1 If you do not use package `hyperref`

If you did not specify both "landscape" and "a4paper" as document class options, do it yet.

`dvips` compile your document with option `-t landscape`.

`dvipdf` insert `\special{landscape}` at the beginning of your document and compile it as usual.

`dvipdfm` compile your document with option `-l`.

[1] If you use teTeX, you may run the command `texconfig` to configure the whole system.

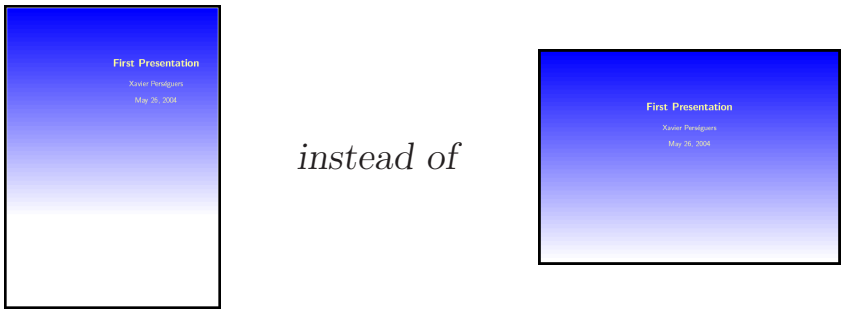


Figure A.1: Invalid Page Format

A.1.2 If you use package `hyperref`

You probably noticed that package `hyperref` only creates working links and other dynamic PDF features if you specify the driver you use to produce the PDF file as one of its option; that is either `dvips` or `ps2pdf` or even `dvipdfm`.

The problem is that as soon as you specify the driver, the page orientation is always set to portrait as shown on Figure A.1.

This is a well-known problem of using package `hyperref` in landscape page orientation mode. The code below fixes the bug if you use `dvips` + `ps2pdf` or `dvipdf`. You should copy it in the preamble of your document.

```
1 \makeatletter
2   \def\special@paper{297mm,210mm}
3 \makeatother
```

`dvips` compile your document without any additional option.

`dvipdf` compile your document as usual.

If you use `dvipdfm` instead, add option `dvipdfm` to the package `hyperref` and compile your document as usual with option `-l`.

Beware: It is also possible to give the option `dvips` in addition to the option `landscape` to the document class declaration to get a landscape paper orientation but you may not be able to define `a4paper` as well resulting in a PDF in landscape orientation but not using the A4 paper format.

A.2 Multiple Inclusion of a Picture

Each time you include a picture in your document, the driver processing the DVI file generates code to encapsulate the image's bitstream into the resulting PostScript file. If for instance you use a logo on each slide, it will be included as many times as there are slides, although it is always the same file. It is even worse if you use overlays as each overlay specification results in a new slide!

Problem. The generated PostScript file's size grows linearly with the number of use of the corresponding picture. If, for instance, you use a 35 KB logo with a 100 slide presentation, your output, as PostScript, will be 3500 KB worth just for the logo!

If you convert the PostScript file to PDF, the presentation will be extremly smaller but its weight will still grow linearly.

Workaround Solution. The PDF specification has a concept of *external object* (commonly called a *XObject*). It is a graphics object which contents are defined by a self-contained content stream, separate from the content stream in which it is used.

The idea would be to include pictures this way, giving them a name and then using them with a reference instead of including them multiple times.

A.2.1 What you need to know

Actually, using the latest distribution of teTEX with Linux solves this problem automatically. A few tests have been made with the file below. Using either `pdflatex` or `latex` + `dvips` + `ps2pdf` for producing the PDF file gave always a file as big as if the image had been included only once. The latest distribution of MikTEX under Windows was not able to create small PDF files. However a Windows PostScript generated file can result in a small PDF file when converted with `ps2pdf` under Linux, although the version of Ghostscript is equivalent!

```
\documentclass{article}
\usepackage{graphics}
\begin{document}
\includegraphics{foo} \includegraphics{foo} \includegraphics{foo}
\end{document}
```

Versions of the Programs[2]

teTEX	
`dvips`	5.92b
`pdflatex`	3.14159-1.10b
`Ghostscript`	AFPL 8.14

MikTEX	
`dvips`	5.90a
`pdflatex`	2.4.1542 (1.20a-rc1)
`Ghostscript`	AFPL 8.14

A.2.2 Using `dvipdfm`

`dvipdfm` includes command to create XObjects. The following code shows how you may declare and then reuse a picture. Please note that:

- The XObject definition does not display the picture;
- You should encapsulate use of XObject within a box to let LATEX create a decent layout. The problem is that LATEX is not aware of the dimensions of the picture as this is purely done by the DVI driver;
- XObject dimensions should be manually adjusted or left bigger than what is actually needed.

Requirements: packages `graphics` and `ifthen`, `dvipdfm` to compile the DVI file

File name: `logo-dvipdfm.tex`

```
\documentclass{article}

\usepackage{graphics}
\usepackage{ifthen}

\DeclareRobustCommand*{\myincludegraphics}[2][NOFILE]{%
  \ifthenelse{\equal{#1}{NOFILE}}{
```

[2] As available on June, 4th 2004.

```
    \special{pdf: uxobj @#2}
  }{%
    \special{pdf: bxobj @#2 width 6.0in height -6.0in}%
    \makebox[0cm]{\includegraphics{#1}}%
    \special{pdf: exobj}%
  }%
}

\begin{document}

% Definition of the logo (without inclusion)
\myincludegraphics[logo-filename]{myLogo}%

Some text

% Use of the logo
\myincludegraphics{myLogo}

\end{document}
```

A.2.3 Using `pdflatex`

The syntax of picture inclusion seems somewhat odd at first sight but authorizes the picture to be included only once even if you need to "include" it multiple times. pdfLaTeX creates a unique identifier for each picture allowing you to reference it for a later use.

```
\pdfximage{logo-filename}
\pdfrefximage \pdflastximage
```

`\pdfximage` creates a new object, containing the specified picture and holds it in memory. Hence, this object is not written to the PDF output unless:

1. the image is referenced by saying `\pdfrefximage` ⟨*object number*⟩;
2. `\pdfximage` is preceded by `\immediate`.

`\pdflastximage` returns the ID of the picture object produced last. Once a picture object has been created, you may use the associated object arbitrarily often again.

It's highly recommended to wrap `\pdfrefximage` by a box in order to ensure spacing will be correct:

```
\hbox{\pdfrefximage \pdflastximage}
```

Size and Resolution of the Image

The following examples are taken from [13] and associated documentation.[3]

If `\pdfimageresolution` is set, the image will be inserted at that resolution (or 72 if `\pdfimageresolution` is set to zero) given in dots per inch in the output file, but some images may contain data specifying the image resolution, and in such a case the image will be scaled to the original resolution. The dimensions of the image can be accessed by enclosing the `\pdfrefximage` command to a box and checking the dimensions of the box:

```
\pdfximage width 3cm {logo.jpg} % set the image width and keep
\pdfrefximage \pdflastximage    % the proportion width/height
```

or with both dimensions...

[3] The documenation is part of teTeX and MikTeX distributions.

```
\pdfximage width 3cm height 2cm % set both width and height;
   {logo.jpg}                   % the final proportion W/H
                                % may be different from the
\pdfrefximage \pdflastximage    % original one
```

and how an image may be inserted at a given resolution...

```
\pdfimageresolution=72          % to open an image at 72 dpi
\pdfximage {logo.jpg}
% get dimensions of the image in order to include the image
% at a specific resolution
\setbox0=\hbox{\pdfrefximage\pdflastximage}
% calculate the image width at 1200 dpi (0.06 = 72/1200)
\dimen0=.06\wd0
% include the image at resolution 1200 dpi by setting image
% width to the calculated value
\pdfximage width \dimen0 {logo.jpg}
\pdfrefximage \pdflastximage
```

A.2.4 Quicker Picture Inclusion

If you include multiple times the same graphics and use each time the `includegraphics` command, LaTeX will read the picture file for each inclusion, resulting in a longer process.

Manipulating boxed material

Material can be typeset once and then stored inside a named box, whose contents can later be retrieved. Include the declaration in the preamble:

```
\newsavebox{\myLogo}
\sbox{\myLogo}{\includegraphics{logo-filename}}
```

and reuse the box in the document, to include the picture:

```
\usebox{\myLogo}
```

B | Additional Material

Contents

B.1 Text following a Sinus Curve

This example is written for packages PROSPER and TEXPower. It was not extended to BEAMER as this package best works with `pdflatex` and `pdflatex` cannot handle package psTricks as said in § 6.1, p. 53.

B.1.1 Package Source Code

Requirements: packages `pst-text` and `pst-plot`
File name: `wavetext.sty`

```
%% WaveText for Prosper
%% Created by Xavier Perseguers
%% See the GNU General Public License
\def\fileversion{1.1}
\def\filedate{2004/06/11}
\def\docdate{2004/06/11}
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{wavetext}
[\filedate\space\fileversion\space WaveText for Prosper package]
\RequirePackage{pst-text,pst-plot}
\newcommand*{\WT@pkgname}{wavetext}

\newif\if@prosper    \@prosperfalse
\newif\if@texpower   \@texpowerfalse
\newif\if@nopackage  \@nopackagetrue
\DeclareOption{prosper}{
  \@prospertrue
  \@texpowerfalse
  \@nopackagefalse
}
\DeclareOption{texpower}{
  \@prosperfalse
  \@texpowertrue
  \@nopackagefalse
}
\DeclareOption*{
  \PackageWarning{\WT@pkgname}{Unknown Option \CurrentOption}
}
\ProcessOptions

\if@nopackage
  \PackageError{\WT@pkgname}{No presentation package specified}
\fi
```

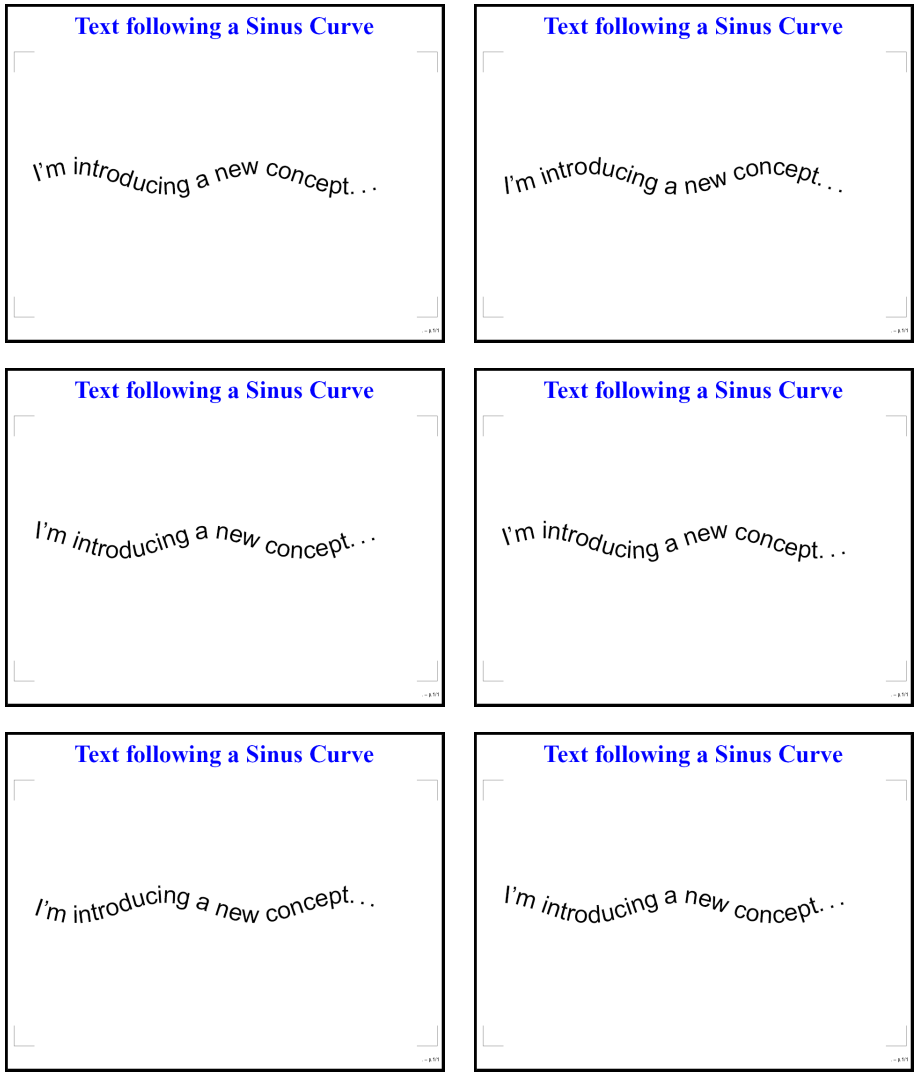
```
34
35 \def\textsin#1 from #2 length #3 #4\par{%
36   \begin{pspicture}(0,-#1)(10,#1)
37     %\psframe(0,-#1)(10,#1)
38     \rput[l](0,0){%
39       \pstextpath%
40       {\psplot[linestyle=none,plotpoints=300,xunit=.015,yunit=#1]%
41               {0}{#3}{x #2 add sin}}
42       {#4}
43     }
44   \end{pspicture}
45   \par
46 }
47
48 \newcount\WT@n
49 \newcount\WT@m
50 \def\animtextsin#1 #2\par{\WT@n=0%
51   %% Prosper code
52   \if@prosper
53   \loop\ifnum\WT@n<#1
54     \WT@m=\WT@n\multiply\WT@m by10%
55     \onlySlide*{\number\WT@n}
56                {\textsin .3 from {\number\WT@m} length 1000 #2\par}
57   \advance\WT@n by1 \repeat
58   \fi
59   %% TeXPower code
60   \if@texpower
61   \loop\ifnum\WT@n<#1
62     \WT@m=\WT@n\multiply\WT@m by10%
63     \steponce{\textsin .3 from {\number\WT@m} length 1000 #2\par}
64   \advance\WT@n by1 \repeat
65   \fi
66 }
67
68 \def\autoadvance{
69  \special {ps: /pdfmark where {pop} {userdict /pdfmark
70    /cleartomark load put} ifelse [ {ThisPage} << /Dur 0 >>
71    /PUT pdfmark }
72 }
73
74 \endinput
75 %%
76 %% End of file wavetext.sty
```

B.1.2 Examples of Use

Prosper

File name: `prosper-wavetext.tex`

```
1  \documentclass[slideColor,pdf,default]{prosper}
2
3  \usepackage[prosper]{wavetext}
4
5  \begin{document}
6
7  \overlays{99}{%$
8   \begin{slide}{Text following a Sinus Curve}
9     \autoadvance
10    \vspace{3cm}
11    \animtextsin 100 \Large I'm introducing a new concept\ldots\par
12  \end{slide}
13 }
14
15 \end{document}
```



TEXPower

File name: `texpower-wavetext.tex`

```
\documentclass[landscape,a4paper]{foils}

\usepackage{color,soul,fixseminar,hyperref}
\usepackage[display]{texpower}
\usepackage[texpower]{wavetext}

\LogoOff
\rightfooter{}

\begin{document}

\foilhead{Text following a Sinus Curve}

\vspace{3cm}

\stepwise*{
  \autoadvance
  \animtextsin 100 \Large I'm introducing a new concept\ldots\par
}

\end{document}
```

C | A few LaTeX Explanations

Contents

C.1 Fonts and Sizes

The default font at `\normalsize` is a "sans serif font" at size 20pt, unless one of the [17pt], [25pt], [30pt] or shortform[1] options have been declared in the `\documentclass` command. Table C.1 shows the control sequences for other accessible text fonts and the name of the font in a sample of its type. These control sequences give the font at the current size. Font size changing commands for each of the normal point size options are described by Table C.2. Note that `\bf` and `\sl` yield sans serif fonts, not the usual variations on roman.

Command	Font Name
`\sf`	Sans Serif
`\it`	*Text Italic*
`\sl`	*Slanted Sans Serif*
`\bf`	**Bold Sans Serif**
`\tt`	`Typewriter`
`\rm`	Roman
`\sc`	Small Caps

Table C.1: Available fonts and their name

C.2 Fragile and Robust Commands

LaTeX commands are either fragile or robust. Fragile commands need special care if they are part of a moving argument.

Arguments to some LaTeX commands are called moving arguments because they may be "moved" to other places in the document. For example, the argument of Sectioning commands may be moved to the Table of Contents. Other examples of moving arguments include:

[1] This class option can be used to generate a document which takes up much less space (paper, mostly).

	Document Option				
Size	20pt	17pt	25pt	30pt	shortform
\tiny	12pt	12pt	12pt	14pt	12pt
\scriptsize	12pt	12pt	14pt	17pt	12pt
\footnotesize	14pt	12pt	17pt	20pt	12pt
\small	17pt	14pt	20pt	25pt	12pt
\normalsize	20pt	17pt	25pt	30pt	12pt
\large	25pt	20pt	30pt	36pt	14pt
\Large	30pt	25pt	36pt	43pt	17pt
\LARGE	36pt	30pt	43pt	51pt	20pt
\huge	43pt	36pt	51pt	51pt	25pt
\Huge	51pt	43pt	51pt	51pt	25pt

Table C.2: Type sizes for size-changing commands for the different document style options

- arguments of `\caption` commands;
- terminal input and output, `\typeout` and `\typein`;
- commands that produce page headings;
- the Letter Environment;
- the `\thanks` command;
- an @ expression in the Array or Tabular environment.

A fragile command that appears in a moving argument must be preceded by a `\protect` command. The `\protect` applies only to the immediately following command; if arguments of this command also contain fragile commands, the latter must be protected with their own `\protect`.

C.2.1 Some Fragile Commands

The following list is not exhaustive, but illustrates the kind of commands which are fragile:

- all commands that have an optional argument are fragile;
- environments delimited by `\begin` ... `\end` are fragile;
- display math environment delimited by `\[` ... `\]`;
- Math environment `\(` ... `\)`;
  however, `$` ... `$` is robust;
- line breaks, `\\`;
- `\item` commands;
- `\footnote` commands.

C.2.2 Some Robust Commands

In general, commands which change Type face or Type style are robust. Length commands are robust and should not be preceded by a `\protect` command. Nor should a `\protect` command be used in the argument to `\addtocounter` or `\setcounter` command.

References

[1] Sascha Beuermann. Erstellung von leistungfähigen PDF-Dokumenten mit LaTeX und den Paketen `hyperref` sowie `thumbpdf`. *Institut für Baumechanik und Numerische Mechanik Universität Hannover*, January 2002.
http://www.ibnm.uni-hannover.de/Mitarbeiter/beuerman/LaTeX2PDF.pdf.

[2] Aladdin Enterprises. *PostScript-to-PDF converter*, 2004.
http://cs.wisc.edu/ ghost/doc/cvs/Ps2pdf.htm.

[3] Elena Fraboschi. Marking up PDFs. *Indiana University Mathematics Journal*, September 1999.
http://inca.math.indiana.edu/iumj/Authors/pdfmark/pdfmark.pdf.

[4] Frédéric Goualard and Peter Møller Neergaard. Making slides in LaTeX with Prosper. February 2003.
http://www.tug.org/tex-archive/macros/latex/contrib/prosper/doc/
▷ prosper-doc.pdf.

[5] Sheldon Green. Fragile and robust commands. *Hypertext Help with LaTeX*, September 1995.
http://www-h.eng.cam.ac.uk/help/tpl/textprocessing/teTeX/latex/latex2e-html/
▷ fragile.html.

[6] Rajarshi Guha. Making presentations with LaTeX and Prosper. *freshmeat.net*, December 2002. http://freshmeat.net/articles/view/667.

[7] Jim Hafner. The FoilTeX class package. *IBM Research Division, Almaden Research Center*, October 1998. http://tex.loria.fr/classes/foiltex.pdf.

[8] Adobe Systems Incorporated. *pdfmark Reference Manual*, November 1999. http://partners.adobe.com/asn/acrobat/docs/pdfmark.pdf.

[9] Adobe Systems Incorporated. *PDF Reference*, version 1.5, 4th edition, August 2001.
The specification of Adobe's Portable Document Format (PDF). The document introduces and explains all aspects of the PDF format, including its architecture and imaging model (allowing transparancy and opacity for text, images, and graphics), the command syntax, the graphics operators, fonts and rendering, and the relation between PostScript and PDF.
http://partners.adobe.com/asn/tech/pdf/specification.jsp.

[10] Ki-Joo Kim. Beamer guide. April 2004.
http://faq.ktug.or.kr/wiki/uploads/beamer_guide.pdf.

[11] Stephan Lehmke. Dynamic online and beamer presentations in PDF using LaTeX and TeXPower. *Universität Dortmund*, September 2003.

[12] Till Tantau. *User's Guide to the Beamer Class*, April 2004. http://www.tug.org/tex-archive/macros/latex/contrib/beamer/doc/▷ beameruserguide.pdf.

[13] Hàn Thế Thành. *The pdfTEX manual*, January 2000.

[14] William B. Thompson. Fonts in LaTeX. *www.cs.utah.edu*. http://www.cs.utah.edu/support/contrib/thompson/latex/latex-fonts.html.

[15] Paul Walmsley. QuickRep — A quick report writing class. *Signal Processing Group, Cambridge University Engineering Department*, November 1999. http://www-h.eng.cam.ac.uk/help/tpl/textprocessing/quickrep.pdf.

[16] Michael Wiedmann. Screen presentations. *miwie.org*, April 2004. http://www.miwie.org/presentations.

Index

U

V

X