ltx-talk – A class for typesetting (accessible) presentations*

Joseph Wright†

Released 2026-11-16

Contents

1 Introduction **2**

2 Engine support **3**

3 Submitting ideas **3**

4 Simple example documents **4**

5 Class structure and design decisions **5**

6 Adjusting settings **5**

7 Differences from beamer **5**

8 Differences from "classical" LaTeX 2ε **6**

9 Creating frames **6**
9.1 The frame environment 6
9.2 Components of a frame 7
9.2.1 The frame title 7
9.3 Frame and margin size 8
9.4 Restricting the slides of a frame 8

10 Creating overlays **8**
10.1 The \pause command 8
10.2 The general concept of overlay specifications 9
10.3 Commands with overlay specifications 10
10.4 Environments with overlay specifications 13
10.5 Dynamically changing text or images 13
10.6 Advanced overlay specifications 14
10.6.1 Mode specifications 14
10.6.2 Action specifications 15
10.6.3 Incremental specifications 16

*This file describes v0.6.13, last revised 2026-11-16.

†E-mail: joseph@texdev.net

11 Structuring a presentation: the global structure **18**
11.1 Adding a title frame . . . 18

12 Structuring a presentation: the local structure **19**
12.1 Itemizations, enumerations and descriptions . . . 19
12.2 Highlighting . . . 20
12.3 Block environments . . . 20
12.4 Figures and tables . . . 21
12.5 Splitting a frame into multiple columns . . . 21
12.6 Footnotes . . . 22
12.7 Repeating a frame . . . 22
12.8 Supplementary frames . . . 23
12.9 Automatic frame breaking . . . 23

13 Creating alternative output formats **23**
13.1 Creating handouts using the `handout` mode . . . 24

14 Creating tagging PDF output **24**
14.1 Enabling tagging . . . 24
14.2 Current tagging approach . . . 24
14.3 Controlling tagging . . . 25

15 Changing the way things look **25**
15.1 Font choice and loading . . . 25
15.2 Template `floatenv` . . . 26
15.3 Template `footer` . . . 26
15.4 Template `header` . . . 27

16 Guidelines for creating presentations **27**
16.1 Structuring a presentation . . . 27
16.1.1 Know the time constraints . . . 27
16.1.2 Global structure . . . 28
16.1.3 Frame structure . . . 31
16.2 Using graphics . . . 33
16.3 Choosing appropriate colors . . . 33

17 Code interfaces **34**

Index **34**

1 Introduction

The beamer class was first released in 2003, and rapidly became the most popular method for producing presentations in LaTeX. As detailed in the beamer manual

> Till Tantau created beamer mainly in his spare time. Many other people have helped by sending him emails containing suggestions for improvement or corrections or patches or whole new themes (by now, this amounts to over a thousand emails concerning beamer). Indeed, most of the development was only initiated by feature requests and bug reports. Without this feedback, beamer would still be what it was originally intended to be: a small private collection of macros that make using the seminar class easier. Till created the first version of beamer for his PhD defense presentation in February 2003. A month later, he put the package on CTAN at the request of some colleagues. After that, things somehow got out of hand.

Despite the very large amount of work Till (and others) put into beamer, there are several challenges which confront us today.

1. The document interface is flexible but in places deviates from normal LaTeX conventions
2. Internally, the code makes use of whatever methods would give the visual results but not necessarily the most idiomatic style
3. Till made few comments in the code or in commit messages in the code history
4. The underlying box structure of a beamer document is very different from the standard LaTeX model, and a lot of material is boxed up multiple times
5. Engine, font and macro development over the past 20 years offers new approaches for some areas

These all feed into an issue addressing a major requirement today: accessibility. Broadly, the internal structure (and to some extent the user interface) of beamer means that it is not possible to "retrofit" PDF tagging into the class.

Instead, the approach is to develop a new class, ltx-talk, which takes ideas from beamer but with tagging and accessibility of structure as a design aim from the beginning. In contrast to work by the LaTeX Project Team on making the core classes accessible, the expectation for ltx-talk is that users *will* need to change their sources. Unlike other documents, presentations tend to be "single use": revised and adjusted each time they are used. The need to edit sources should therefore not be an insurmountable barrier.

At present, this code is experimental: only a (small) subset of beamer functionality is implemented, some things are being done differently, and almost everything is still subject to discussion. Some document commands are stubs: either doing nothing at all or dropping optional arguments. This is to allow testing of existing beamer sources with minimal changes.

2 Engine support

Creating fully tagged PDFs is only possible with suitable engine support; LuaTeX offers the best outcomes here and pdfTeX is usable but with some limitations (for example, more LaTeX runs being required). As such, ltx-talk is *only* tested with these engines. Most notably, XeTeX is *not* supported: it is not suitable for fully accessible tagged PDF production, and transparency support is problematic. Issues arising which are XeTeX-specific will therefore be closed without investigation.

3 Submitting ideas

As noted in Section 1, the class is at present experimental. As such, missing features or restrictions should be expected throughout. Over time, the aim is to address many of the things that beamer can do (though there may be some that are not included).

At present, prioritization of requests will be focussed on the need to provide accessible content. This means that structural issues are likely to be handled before design aspects.

4 Simple example documents

Using ltx-talk *absolutely requires* the use of the `\DocumentMetadata` command. As such, the most basic ltx-talk document is

```
\DocumentMetadata{} % likely with "tagging = on"
\documentclass{ltx-talk}
\begin{document}
\begin{frame}
  Some content here
\end{frame}
\end{document}
```

A slightly more useful version, which generates multiple slides and shows some (current) features, is

```
\DocumentMetadata{}
\documentclass{ltx-talk}
\begin{document}
\begin{frame}
  \frametitle{An example frame}
  \begin{itemize}[action-spec = <+->]
    \item This will be on slide one onward
    \item This will be on slide two onward
    \item<.-> So will this
    \item But this will only appear on slide three
  \end{itemize}
  Back to appearing on all slides
\end{frame}
\end{document}
```

Tagging is activated for the standard (projector) output of ltx-talk, but it is perhaps more useful in `handout` mode, which is activated using a class option.

```
\DocumentMetadata{}
\documentclass[mode = handout]{ltx-talk}
\begin{document}
\begin{frame}
  \frametitle{An example frame}
  \begin{itemize}[action-spec = <+->]
    \item This will be on slide one onward
    \item This will be on slide two onward
    \item<.-> So will this
    \item But this will only appear on slide three
  \end{itemize}
  Back to appearing on all slides
\end{frame}
\end{document}
```

For convenience, this option is also available as simply `handout`, to match the beamer syntax.

A larger set of examples which can be edited and typeset in the browser are available at https://www.texdev.net/ltx-talk/examples/.

5 Class structure and design decisions

As covered in Section 1, the ltx-talk class is currently highly experimental. Active discussion is ongoing around many aspects of the way that things should work, and very little is therefore at all stable. That said, some decisions have been made: some of this is re-stating ideas which carry forward from beamer.

- The basic structure of a presentation is made up of `frame` environments, which are made up of one or more slides.
- Variable content is indicated by an *overlay specification*, given in optional angle brackets (`< ... >`).
- Slides have a *fixed* height of 100 mm.
- The default font will be sans-serif using established standard implementations (currently sansmathfonts with Latin Modern for text for pdfTEX and New Computer Modern for OpenType engines).

6 Adjusting settings

At present, settings for ltx-talk are not all controlled from a single location. Over time this is likely to evolve and be more clearly documented. Where keys are used for *ad hoc* settings, these can typically be set globally by using `\SetKeys[talk]{`⟨*keys*⟩`}`. The keys are currently in different subtrees. For example, overall frame vertical alignment could be set using

```
\SetKeys[talk]{frame/vertical-alignment = top}
```

Work on both the keys themselves and the documentation is on the active list of tasks to complete.

7 Differences from beamer

The following key differences between beamer and ltx-talk are important to note for the user.

- The default font setup in ltx-talk is *all* sans-serif, in particular `\mathrm` and `\textrm` will give a sans-serif font
- Overlay specifications can only be given as the first argument to commands
- Where a command takes a star as an option, this comes *before* the overlay argument, e.g. `\includegraphics*<...>`: this reflects the fact that the star is usually described as part of the command name rather than as an argument
- There are no optional braced arguments in ltx-talk: frame titles are given using `\frametitle` or, with the class option `frame-title-arg`, using a *mandatory* `{...}` argument (see Section 9.2.1)
- New overlay-aware commands and environments should be defined using ltcmd (`\NewDocumentCommand` and so on): no changes are made to the behavior of `\newcommand` or `\newenvironment`
- There are no commands containing the class name (or `beamer`): rather, methods from the kernel or other standard packages should be used (these are documented where needed)
- The class features are specifically focussed on presentation and avoid as far as possible other changes from standard LaTeX; in particular, the various math-focussed additions which beamer make (such as various theorem environments) are not included

8 Differences from "classical" LaTeX 2$_\varepsilon$

The class builds on ideas developed by the LaTeX team over many years, and introduced along with tagging support in recent years. This includes some revision of how standard LaTeX commands work, in particular

- The optional argument to (sub)sections is treated as a set of key–value entries, and these can be used to have fine control over the processing of the title (for example, whether it appears in the table of contents or PDF bookmarks); this argument is favored over using the starred form.

9 Creating frames

9.1 The frame environment

`frame` (*env.*) A presentation consists of a series of frames. Each frame consists of a series of slides. You create a frame using the `frame` environment. All of the text that is not tagged by overlay specifications is shown on all slides of the frame. (Overlay specifications are explained in more detail in later sections. For the moment, let's just say that an overlay specification is a list of numbers or number ranges in angle brackets that is put after certain commands as in `\uncover<1,2>{Text}`.) If a frame contains commands that

have an overlay specification, the frame will contain multiple slides; otherwise it contains only one slide.

```
\begin{frame}<⟨overlay spec⟩>[⟨options⟩]
⟨environment contents⟩
\end{frame}
```

The ⟨*overlay spec*⟩ dictates which slides of a frame are to be shown. If left out, the number is calculated automatically. The ⟨*environment contents*⟩ can be normal LaTeX text, but may not contain `\verb` commands or `verbatim` environments or any environment that changes the character codes.

```
\begin{frame}
  \frametitle{A title}
  Some content
\end{frame}
```

`frame*` (*env.*) The `frame*` environment works exactly the same way as the `frame` one, *except* that it can contain `\verb` commands or `verbatim` environments or any environment that changes the character codes. The `frame*` environment however *cannot* be used inside the argument of another command.

```
\begin{frame*}
  \frametitle{A title}
  Some content \verb|$%#|.
\end{frame*}
```

In contrast to beamer's `fragile` frame option, the `frame*` environment does not use external files to handle verbatim material. As such, there is very little performance impact processing frames containing category code changes.

The `frame` and `frame*` environments recognise the following options

- `action-spec` Applies the `action-spec` to all nested environments which take a suitable specification: see Section 12.1 for more details.
- `auto-break` and
- `auto-break-coverage` Control automatic breaking of frames into slides: see Section 12.9 for more details.
- `name` Sets the name of a frame for reuse: see Section 12.7.
- `label` An alternative for `name`, available for compatibility with beamer.
- `supplementary-frame` Indicates the frame is supplementary: see Section 12.8.
- `vertical-alignment` Defines the vertical alignment of frame content: choice of `bottom`, `center`, `top` or `stretch`.

For compatibility with beamer, the optional argument will allow the use of an `action-spec` without a key name, for example

```
\begin{frame}[<+->]
```

9.2 Components of a frame

9.2.1 The frame title

\frametitle
\framesubtitle

```
\frametitle<⟨overlay spec⟩>[⟨options⟩]{⟨frame title⟩}
\framesubtitle<⟨overlay spec⟩>[⟨options⟩]{⟨frame subtitle⟩}
```

The frame title is shown prominently at the top of the frame and can be specified with the command `\frametitle`. You should end the ⟨*frame title*⟩ with a period, if the title is a proper sentence. Otherwise, there should not be a period. A frame subtitle may also be given; this is typically used where a series of related frames are given with clearly distinct content.

Currently, the ⟨*frame subtitle*⟩ is not used in output: this will be addressed in later releases. The ⟨*options*⟩ for both commands are currently unused.

`frame-title-arg` The class can be loaded with the option `frame-title-arg`. When this is active, the `frame` environment *requires* an argument in all cases

```
\begin{frame}{Frame title here}
```

Note that this option does *not* influence frame subtitles, which can *only* be given using `\framesubitle`.

At present, the final form of the frame title interface is not decided: feedback on this aspect is therefore *particularly welcome*.

9.3 Frame and margin size

The size of a frame is actually the "paper size" of a presentation, and it is variable. The standard size of a frame is 100 mm height with an aspect ratio of 16 : 9. It is the job of the presentation program to display the slides at full screen size. The main advantage of using a small "paper size" is that you can use all your normal fonts at their natural sizes. In particular, inserting a graphic with 11 pt labels will result in reasonably sized text during the presentation.

`font-size` You can change the standard font size for the frames using the `font-size` option. This is initially set to `11pt`, and if a non-standard size is requested will use the relsize package to produce the output requested. To support translation from beamer, the classical options `10pt`, `11pt` and `12pt` are also supported.

`aspect-ratio` The aspect ratio of the slides can be adjusted by setting the `aspect-ratio` load-time option, which takes two integer values separated by a colon. These values are the relative width and height of the slide: in contrast to beamer, the absolute height is a fixed value of 100 mm.

Aside from using these options, you should refrain from changing the "paper size". However, you *can* change the size of the left and right margins, which default to 10 mm. These should be set using the interfaces from the geometry package, for example

```
\geometry{lmargin = 15mm, rmargin = 15mm}
```

9.4 Restricting the slides of a frame

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

10 Creating overlays

10.1 The \pause command

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

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

The \pause command takes an optional argument

\pause[⟨*overlay spec*⟩]

This causes the text following it to be shown only from the next slide on, or, if the optional ⟨*overlay spec*⟩ is given, from the slide with the number ⟨*overlay spec*⟩. If the optional ⟨*overlay spec*⟩ is given, the counter pauses is set to this number. This command uses the \onslide command internally. The effect of \pause lasts till the next \pause, \onslide, or the end of the frame.

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

  Shown from fourth slide on.

  \begin{itemize}
  \onslide
  \item
    Shown from first slide on.
```

```
  \pause
  \item
    Shown from fifth slide on.
  \end{itemize}
\end{frame}
```

This command does *not* work correctly inside math mode environments like `align` and pgf environments like `tikzpicture` or `tcolorbox`, since these do really wicked things.

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

10.2 The general concept of overlay specifications

Whilst the `\pause` command is easy to understand, it is quite limited and so is best suited only to simple cases. The ltx-talk class therefore supports a different approach. The idea is to add *overlay specifications* to commands. These specifications are always given in pointed brackets and follow the command as the first argument. In the simplest case, the specification contains just a number. A command with an overlay specification following it will only have "effect" on the slide(s) mentioned in the specification. What exactly "having an effect" means, depends on the command. Consider the following example.

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

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

For a second example, consider the following frame:

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

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

Overlay specifications can only be written behind certain commands, not every command. Which commands you can use and which effects this will have is explained in the next section. However, it is quite easy to redefine an existing command such that it becomes "overlay specification aware", see also Section 10.3.

The syntax of (basic) overlay specifications is the following: they are comma-separated lists of slides and ranges. Ranges are specified like this: `2-5`, which means slide two through to five. The start or the end of a range can be omitted. For example, `3-` means "slides three, four, five, and so on" and `-5` means the same as `1-5`. A complicated example is `-3,6-8,10,12-15`, which selects the slides 1, 2, 3, 6, 7, 8, 10, 12, 13, 14 and 15.

10.3 Commands with overlay specifications

For the following commands, adding an overlay specification causes the command to be simply ignored on slides that are not included in the specification: `\textbf`, `\textit`, `\textmd`, `\textnormal`, `\textrm`, `\textsc`, `\textsf`, `\textsl`, `\texttt`, `\textup`, `\emph`; `\color`, `\mathcolor`, `\textcolor`; `\alert`; `\includegraphics`; `\label`.

If a command takes several arguments, like `\color`, the specification should directly follow the command as in the following example:

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

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

`\onslide`

`\onslide<`⟨*overlay spec*⟩`>`

All text following this command will only be shown (uncovered) on the specified slides. On non-specified slides, the text still occupies space. If no slides are specified, the following text is always shown. You need not call this command in the same TEX group, its effect transcends block groups.

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

`\only`

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

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

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

`\uncover`

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

If the ⟨*overlay spec*⟩ is present, the ⟨*text*⟩ is shown ("uncovered") only on the specified slides. On other slides, the text still occupies space and it is still typeset, but it is not shown or only shown as if transparent.

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

To adjust the transparency, use

```
\EditInstance{hidden}{std}{opacity = ⟨value⟩}
```

where ⟨*value*⟩ ranges from 0 (totally invisible) to 1 (completely visible).

`\visible`

```
\visible<⟨overlay spec⟩>{⟨text⟩}
```

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

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

`\invisible`

```
\invisible<⟨overlay spec⟩>{⟨text⟩}
```

This command does the opposite of `\visible`.

```
\invisible<2->{Text hidden from slide~2 on.}
```

`\alt`

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

The default text is shown on the specified slides, otherwise the alternative text.

```
\alt<2>{On slide~2}{Not on slide~2.}
```

`\temporal`

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

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

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

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

```
\NewDocumentCommand\colorize{D<>{all}}{%
  \temporal<#1>{\color{red!50}}{\color{black}}{\color{black!50}}}
\begin{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}
\end{frame}
```

`\item`

`\item<`⟨*action spec*⟩`>[`⟨*item label*⟩`]`

The effect of ⟨*action spec*⟩ is described in Section 10.6.2; this extends the ⟨*overlay spec*⟩ to include the potential to "alert" items.

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

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

10.4 Environments with overlay specifications

`onlyenv` (*env.*)
`invisibleenv` (*env.*)
`uncoverenv` (*env.*)
`visibleenv` (*env.*)

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

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

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

10.5 Dynamically changing text or images

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

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

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

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

`overlayarea` (*env.*)

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

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` (*env.*)

`\begin{overprint}[`⟨*area width*⟩`]`

The ⟨*area width*⟩ defaults to the text width. Inside the environment, use `\only` or `onlyenv` to specify different things that should be shown for this environment on different slides. Everything within the environment will be placed in a rectangular area of the specified width. The height and depth of the area are chosen large enough to accommodate the largest contents of the area. Two compilations will be needed to allow LaTeX to track the tallest version of the frame contents.

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

A similar need for dynamical changes arises when you have, say, a series of pictures named `first`, `second`, and `third` that show different stages of some process. To make a frame that shows these pictures on different slides, the following code might be used:

```
\begin{frame}
  \frametitle{The Three Process Stages}

  \includegraphics<1>{first}
  \includegraphics<2>{second}
  \includegraphics<3>{third}
\end{frame}
```

10.6 Advanced overlay specifications

10.6.1 Mode specifications

This section is only important if you use the mode mechanism to create different versions of your presentation. If you are not familiar with modes, please skip this section or read Section 13 first.

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

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

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

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

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

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

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

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

This will cause the text `Riddle` to be inserted in `article` mode and on the first slide of a frame in `projector` mode, but not at all in `handout` mode.

10.6.2 Action specifications

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

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

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

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

Not all commands can take an action specification. Currently, only `\item`, `\action`, the environment `actionenv`, and the block environments (like `block`) handle them.

By default, the following actions are available:

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

`actionenv` (*env.*) The whole action mechanism is based on the following environment:

```
\begin{actionenv}<⟨action spec⟩>
```

This environment extracts all actions from the ⟨*action spec*⟩ for the current mode. For each action of the form ⟨*action*⟩`@`⟨*slide numbers*⟩, it inserts the equivalent internal code to: `\begin{`⟨*action*⟩`env}<`⟨*slide number*⟩`>` at the beginning of the environment and the text `\end{`⟨*action*⟩`env}` at the end. An ⟨*overlay spec*⟩ without an action is promoted to `uncover@`⟨*overlay spec*⟩.

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

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

`\action`

```
\action<⟨action spec⟩>{⟨text⟩}
```

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

Note that the syntax for action specifications in ltx-talk is currently more tightly defined than that for beamer. Only a single action is allowed, and this must come *after* any overlay or mode part.

10.6.3 Incremental specifications

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

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

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

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

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

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

The effect of the `+` sign is the following. You can use it in any overlay specification at any point where you would usually use a number. If a `+` sign is encountered, it is replaced by one more than the current value of the LaTeX counter `pauses`; that is 0 at the beginning of the frame, so the first possible replacement value is 1. At the end of the overlay specification, the `pauses` counter is incremented. This only happens once even if the specification contains multiple `+` signs (they are replaced by the same number, which is also the new value of `pauses`).

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

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

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

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

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

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

```
\begin{itemize}
  \item<2-> Apple
  \item<3-> Peach
  \item<4-> Plum
  \item<5-> Orange
\end{itemize}
```

and

```
\begin{itemize}[<+(1)->]
  \item Apple
  \item Peach
  \item Plum
  \item Orange
\end{itemize}
```

are equivalent.

There is another special sign you can use in an overlay specification that behaves similarly to the `+` sign: a dot. When you write `<.->`, a similar process occurs to that for `<+->`, *except* that the counter `pauses` is used unchanged. Thus a dot, possibly followed by an offset, just expands to the current value of the counter `pauses`, possibly offset. This dot notation can be useful in case like the following:

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

In the example, the second item is shown at the same time as the first one since it does not update the counter.

In the following example, each time an item is uncovered, the specified text is alerted. When the next item is uncovered, this altering ends.

```
\begin{itemize}[<+->]
  \item This is \alert<.>{important}.
  \item We want to \alert<.>{highlight} this and \alert<.>{this}.
  \item What is the \alert<.>{matrix}?
\end{itemize}
```

The replacements of the `+` sign and the `.` sign are no less than zero. This prevents errors when encountering large negative offsets, for example `<+(-7)->` is converted to `<0->` rather than `<-6->`.

11 Structuring a presentation: the global structure

This section lists the commands that are used for structuring a presentation "globally" using commands like `\section` or `\part`. These commands are used to create a *static* structure, meaning that the resulting presentation is normally presented one slide after the other in the order the slides occur.

11.1 Adding a title frame

You can use the standard `\maketitle` command to insert a title frame. As standard, it will arrange the following elements on the title page: the document title and subtitle, the authors' names, their affiliation, and the date.

`\maketitle`

```
\maketitle[⟨settings⟩]
```

The `\maketitle` command may be inserted either inside or outside of a `frame` environment; the result will be a full frame in either case. The optional argument to the command is used to specify the following ⟨*settings*⟩

- `element-order` which determines which entries are printed and the order they appear: a comma-list
- `frame-style` sets the frame style
- `horizontal-alignment`
- `vertical-alignment`

`\author` `\date` `\institute` `\subtitle` `\title`

```
\author[⟨options⟩]{⟨author⟩}
\date[⟨options⟩]{⟨date⟩}
\institute[⟨options⟩]{⟨institute⟩}
\subtitle[⟨options⟩]{⟨subtitle⟩}
\title[⟨options⟩]{⟨title⟩}
```

As well as the standard LaTeX elements `\author`, `\title` and `\date`, ltx-talk supports `\subtitle` and `\institute`. These working the same way as the standard commands for providing metadata. The ⟨*options*⟩ may be used to set the `short-` version of each item: if no key name is given, the entire optional argument is used as the short version. If multiple ⟨*options*⟩ are given, they are interpreted as key-values.

12 Structuring a presentation: the local structure

LaTeX provides different commands for structuring text "locally", for example, *via* the `itemize` environment. These environments are also available in the ltx-talk class, although their appearance has been slightly changed.

12.1 Itemizations, enumerations and descriptions

`description` (*env.*)
`enumerate` (*env.*)
`itemize` (*env.*)
`\item`

There are three predefined environments for creating lists, namely `enumerate`, `itemize`, and `description`. The first two can be nested to depth three, but nesting them to this depth creates totally unreadable slides.

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

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

The list environments have syntax

```
\begin{⟨list type⟩}[⟨options⟩]
```

If the option `action-spec` is given, in every occurrence of an `\item` command that does not have an overlay specification attached to it, the ⟨*overlay specification*⟩ is used. By setting this specification to be an incremental overlay specification, see Section 10.6.3, you can implement, for example, a stepwise uncovering of the items. The ⟨*overlay specification*⟩ is inherited by sub-environments. Naturally, in a sub-environment you can reset it locally by setting it to `<1->` (the subitems will be shown on all slides) or `<.->` (the subitems will be shown starting from the same slide as the parent item).

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

If you do not need to give any other options for the list environment, you may use the shortened format `[<...>]`, which matches the beamer syntax:

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

12.2 Highlighting

`\alert`

`\alert<`⟨*overlay spec*⟩`>{`⟨*text*⟩`}`

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

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

`alertenv` (*env.*)

`\begin{alertenv}<`⟨*overlay spec*⟩`>`

Environment version of the `\alert` command.

12.3 Block environments

`block` (*env.*)

`\begin{block}<`⟨*overlay spec*⟩`>{`⟨*block title*⟩`}`

Inserts a block, like a definition or a theorem, with the title ⟨*block title*⟩. If the ⟨*action specification*⟩ is present, the given actions are taken on the specified slides, see Section 10.6.2. In the example, the definition is shown only from slide 3 onward.

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

12.4 Figures and tables

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

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

  \begin{figure}
```

```
    \caption{This caption is placed above the figure.}
    \includegraphics{myotherfigure}
  \end{figure}
\end{frame}
```

Note that at present the standard setting for ltx-talk place the float content left-aligned. This is similar to the LaTeX base classes and contrasts with beamer. See the example directory for the supported method to center content in floats.

The floating environments, like other blocks, accept a first action argument. This can be used to apply an overlay to the entire content, most obviously a graphics and the associated caption.

12.5 Splitting a frame into multiple columns

The class offers several commands and environments for splitting (perhaps only part of) a frame into multiple columns. These commands have nothing to do with LaTeX's commands for creating columns. Columns are especially useful for placing a graphic next to a description/explanation.

The main environment for creating columns is called `columns`. Inside this environment, you can place several `column` environments, each of which creates a new column.

`columns` (*env.*)

```
\begin{columns}<⟨action spec⟩>[⟨options⟩]
```

A multi-column area. If the ⟨*action spec*⟩ is present, the given actions are taken on the specified slides, see Section 10.6.2. The following ⟨*options*⟩ may be given:

- `vertical-alignment` a choice of one of `bottom`, `center` or `top`; this causes the content of the columns to be aligned as specified. The standard setting is `center`. For compatibility with beamer, the short options `b`, `c` and `t` are available.
- `width` will cause the columns to occupy ⟨*width*⟩: the standard setting is `\textwidth`.

`column` (*env.*)

```
\begin{column}<⟨action spec⟩>[⟨placement⟩]{⟨column width⟩}
```

To create a column, you can use the `column` environment.

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

Creates a single column of width ⟨*column width*⟩. If the ⟨*action specification*⟩ is present, the given actions are taken on the specified slides, see Section 10.6.2. The vertical placement of the enclosing `columns` environment can be overruled by specifying a specific ⟨*placement*⟩.

12.6 Footnotes

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

\footnote

\footnote<⟨*overlay spec*⟩>[⟨*mark-number*⟩]{⟨*text*⟩}

Inserts a footnote into the current frame. Footnotes will always be shown on the current frame; they will never be "moved" to other frames. As usual, one can give a ⟨*mark-number*⟩, which will cause the footnote to use that number. Footnotes are placed at the bottom of the content area of the frame: unlike beamer, this is true for footnotes in columns, *etc.*: for beamer, this behavior is similar to the `framefootnotes` class option or `\footnote[frame]{`⟨*text*⟩`}`.

If an ⟨*overlay spec*⟩ is given, this causes the footnote ⟨*text*⟩ to be shown only on the specified slides. The footnote symbol in the text is shown on all slides.

12.7 Repeating a frame

\reuseframe

\reuseframe<⟨*overlay spec*⟩>[⟨*options*⟩]{⟨*name*⟩}

Reuses a frame that was previously created using `frame` or `frame*` environment with the option `name =`⟨*name*⟩. You can use this command to "continue" a frame that has been interrupted by another frame. The effect of this command is to insert the `frame` environment with the given ⟨*overlay specification*⟩ and ⟨*options*⟩ with the original contents.

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

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

\reuseframe<3>{myframe}
```

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

\againframe

To allow conversion of beamer slides, the alternative name `\againframe` is available, and the use of the `label` key in place of `name` is currently supported.

12.8 Supplementary frames

Sometimes, more frames are available than are needed for the presentation. One approach to handling these supplementary frames is to collect them after the `\appendix` command in the source, so that they appear at the end of the presentation. This may mean they are out of logical order, and also means it is more difficult to provide both a "short" and "long" version of the same content.

To address this, ltx-talk provides the class option `supplementary-frames` and frame key `supplementary-frame`. Frames marked as as supplementary then have three different handling possibilities, set using the class option:

- `appendix`: Move to the appendix automatically (after any explicit appendix)
- `in-place`: Output in place in the source, *i.e.* the same as if the frames where not marked as supplementary
- `omit`: Simply omit the the frames entirely

When frames are automatically moved to the appendix, they are names `appendix.`⟨*n*⟩, where ⟨*n*⟩ is a serial number; these therefore can be accessed using `\reuseframe`.

12.9 Automatic frame breaking

In the *vast* majority of cases, frames should be split by the author when there is too much content to appear in one go. However, in the case of automatically-generated material, most commonly bibliographies, this may not be feasible. To allow for this, the frame option `auto-break` is available.

When auto-breaking is activated, ltx-talk will split the content of the frame into different slides based on the vertical height. The split will be activated if the natural height of the content is greater than `\textheight`, and each slide with use `auto-break-coverage` percent of the `\textheight` for the split.

Note that overlays are disabled when breaking is active, and that formal support here is specifically targetded at auto-generated content.

13 Creating alternative output formats

`mode` The ltx-talk class allows you to select different output formats from the same source using the option `mode`. This works with the overlay specification to typeset some of the input selectively.

13.1 Creating handouts using the `handout` mode

During a presentation it is very much desirable that the audience has a *handout* or even *lecture notes* available to it. A handout allows everyone in the audience to individually go back to things they have not understood.

The easiest way of creating a handout for your audience is to set the `mode` option set to ⟨*handout*⟩. This will cause the document to be typeset in `handout` mode. It will look like a presentation, but it can be printed or annotated more easily (the overlays are "flattened").

14 Creating tagging PDF output

As detailed in the introduction (Section 1), the ltx-talk class has been written from the start with tagging in mind. As such, where possible, tagging is generated automatically. This works best with predictable frame structure, for example the classical "uncover a list one item at a time" approach.

14.1 Enabling tagging

Tagging can be enabled using the standard LaTeX interface

```
\DocumentMetadata{tagging = on}
```

as the first line of the source.

In most cases, you will also need to state which standard(s) you are targeting: validators in particular require that standards are explicitly stated. For accessibility, the UA-2 standard is the most likely target, which would mean using

```
\DocumentMetadata{tagging = on, pdfstandard = UA-2}
```

as the first line of the source.

14.2 Current tagging approach

Whilst some aspects of tagging are straight-forward, there are specific challenges in tagging presentations that mean that some parts of the approach are experimental. In particular, the relationship between frame titles and sectioning is still very much open. At present, frame titles are tagged as sections at the level `H4`, *i.e.*, deliberately well below sections and subsections. This may lead to warning from some accessibility checkers.

Feedback here is needed from users of assistive technology tools, to allow adjustment to the standard tagging to best aid them. It is likely that in the future, a richer range of options will be provided in this area. At present, if you want to adjust the tagging you could use for example

```
\tagpdfsetup{role / new-tag = frametitle / H1}
```

14.3 Controlling tagging

`tag-slides` To enable control tagging of slides, the frame option `tag-slides` is available. This can be used to list which slides in a frame are tagged: those omitted are marked as artifacts. The setting should be a comma-list of slide numbers, which an include `n` as a marker to mean the last slide of the frame. The standard setting is `tag-slides = n`, meaning only the last slide of the frame is tagged. You may use mode specifications in the `tag-slides` option, for example `tag-slides = n||handout:1,n` would tag the last slide in the frame in `projector` mode and the first and last for `handout` mode.

15 Changing the way things look

The design of frames in ltx-talk is controlled by *templates*, a feature introduced in the LaTeX kernel in the 2024-06-01 release. The general concept is currently documented in `lttemplates.pdf` with some coverage in `clsguide.pdf`.

Several of the templates are illustrated in the examples, available along with this document or at https://www.texdev.net/ltx-talk/examples/ as live demonstrations.

15.1 Font choice and loading

For pdfTEX users, font choice will typically be determined by the available LATEX font packages. For LuaTEX users, the font choice will be partly determined by the system fonts installed. To get consistent appearance for text and math mode material, LuaTEX users are strongly encouraged to choose a font available with Unicode math support. A demonstration of the majority of current options in this area is available at https://mathfonts.github.io/pdf-test-expression, where you can also try out the appearance of expressions you choose.

At present, ltx-talk loads the fontspec package when used with suitable engines (XeTEX and LauTEX).[1] This means that the interfaces `\setmainfont` and `\setsansfont` can be used to change the body font. This *likely will change* in time: whilst fontspec is convenient for users, there are technical issues that mean that alternatives are being examined. As such, we *strongly* recommend that you load fontspec explicitly if you make use of its functionality.

Note that for LuaTEX, the current font loading approach is using lua-unicode-math with the font then loaded directly. For some use cases, for example bold math usage, you may wish to adjust the font setup using a lua-unicode-math font-specific package such as

```
\usepackage{lum-newcomputermodernsans}
```

This area is very experimental and users may need to manually adjust loading if they have specific font requirements. If you want to set the math font directly, you may also need to adjust the one use for `\mathrm`, for example

```
\setmathfont{STIX Two Math}
\setmathrm{STIX Two Text}
```

15.2 Template `floatenv`

This template is used to create environments which map to the floats in a starndard LATEX document. Note that they do *not* float in an ltx-talk document. The template takes two arguments: options to apply to an individual environment, and the type of "float". The standard template is `talk`, which recognizes the keys

- `float-placement` Ignored at present
- `horizontal-alignment` Takes a choice of `left`, `center` or `right`

15.3 Template `footer`

This template is used to populate the footer area of the slide: this takes no arguments. The standard template is `talk`, which recognizes the keys

- `background-color` The color of the background of the entire footer area
- `color` The color of the text in the footer area
- `element-order` A comma-separated list of metadata elements to print: this can include the `framenumber`. Where available, "short" versions of the metadata items are printed.

[1] This is skipped for some very specialized development cases: you know if this applies to you!

- `font` Font for printing elements
- `left-hspace` Margin on left of footer
- `right-hspace` Margin on right of footer
- `separator` Tokens inserted between elements.

The current list of supported metadata entries is

- `appendixframenumber`
- `appendixframes`
- `author`
- `date`
- `framenumber`
- `frames`
- `institute`
- `subtitle`
- `title`
- `totalframes`

Two instances of this template are pre-defined: `std` and `wallpaper`. The `wallpaper` version omits any elements, meaning that it only applies a colored background to the footer.

15.4 Template `header`

This template is used to populate the header area of the slide: this takes no arguments. The standard template is `talk`, which recognizes the keys

- `background-color` The color of the background of the entire header area
- `color` The color of the text in the header area
- `font` Font for printing elements
- `height` The overall height of the header background area
- `left-hspace` Margin on left of header
- `print-frame-title` A boolean to turn title printing on and off
- `right-hspace` Margin on right of header

Two instances of this template are pre-defined: `std` and `wallpaper`. The `wallpaper` version sets `print-frame-title` to `false`, and thus only adds the background color to the header.

16 Guidelines for creating presentations

In this section we sketch the guidelines that we try to stick to when we create presentations. These guidelines either arise out of experience, out of common sense, or out of recommendations by other people or books. These rules are certainly not intended as commandments that, if not followed, will result in catastrophe. The central rule of typography also applies to creating presentations: *Every rule can be broken, but no rule may be ignored.*

16.1 Structuring a presentation

16.1.1 Know the time constraints

When you start to create a presentation, the very first thing you should worry about is the amount of time you have for your presentation. Depending on the occasion, this can be anything between 2 minutes and two hours.

- A simple rule for the number of frames is that you should have at most one frame per minute.
- In most situations, you will have less time for your presentation than you would like.
- *Do not try to squeeze more into a presentation than time allows for.* No matter how important some detail seems to you, it is better to leave it out, but get the main message across, than getting neither the main message nor the detail across.

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

16.1.2 Global structure

To create the "global structure" of a presentation, with the time constraints in mind, proceed as follows:

- Make a mental inventory of the things you can reasonably talk about within the time available.
- Categorize the inventory into sections and subsections.
- For very long talks (like a 90 minute lecture), you might also divide your talk into independent parts (like a "review of the previous lecture part" and a "main part") using the `\part` command. Note that each part has its own table of contents.
- Do not feel afraid to change the structure later on as you work on the talk.

Parts, section, and subsections

- Do not use more than four sections and not less than two per part.

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

- Ideally, a table of contents should be understandable by itself. In particular, it should be comprehensible *before* someone has heard your talk.
- Keep section and subsection titles self-explaining.
- Both the sections and the subsections should follow a logical pattern.
- Begin with an explanation of what your talk is all about. (Do not assume that everyone knows this. The *Ignorant Audience Law* states: Someone important in the audience always knows less than you think everyone should know, even if you take the Ignorant Audience Law into account.)
- Then explain what you or someone else has found out concerning the subject matter.
- Always conclude your talk with a summary that repeats the main message of the talk in a short and simple way. People pay most attention at the beginning and at the end of talks. The summary is your "second chance" to get across a message.
- You can also add an appendix part using the `\appendix` command. Put everything into this part that you do not actually intend to talk about, but that might come in handy when questions are asked.
- Do not use subsubsections, they are evil.

Giving an abstract In papers, the abstract gives a short summary of the whole paper in about 100 words. This summary is intend to help readers appraise whether they should read the whole paper or not.

- Since your audience is unlikely to flee after the first slide, in a presentation you usually do not need to present an abstract.
- However, if you can give a nice, succinct statement of your talk, you might wish to include an abstract.
- If you include an abstract, be sure that it is *not* some long text but just a very short message.
- *Never, ever* reuse a paper abstract for a presentation.

Numbered theorems and definitions A common way of globally structuring (math) articles and books is to use consecutively numbered definitions and theorems. Unfortunately, for presentations the situation is a bit more complicated and we would like to discourage using numbered theorems in presentations. The audience has no chance of remembering these numbers. *Never* say things like "now, by Theorem 2.5 that I showed you earlier, we have …" It would be much better to refer to, say, Kummer's Theorem instead of Theorem 2.5. If Theorem 2.5 is some obscure theorem that does not have its own name (unlike Kummer's Theorem or Main Theorem or Second Main Theorem or

Key Lemma), then the audience will have forgotten about it anyway by the time you refer to it again.

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

If you do number theorems and definitions, number everything consecutively. Thus if there are one theorem, one lemma, and one definition, you would have Theorem 1, Lemma 2, and Definition 3. Some people prefer all three to be numbered 1. We would *strongly* like to discourage this. The problem is that this makes it virtually impossible to find anything since Theorem 2 might come after Definition 10 or the other way round. Papers and, worse, books that have a Theorem 1 and a Definition 1 are a pain.

- Do not inflict pain on other people.

Bibliographies You may also wish to present a bibliography at the end of your talk, so that people can see what kind of "further reading" is possible. When adding a bibliography to a presentation, keep the following in mind:

- It is a bad idea to present a long bibliography in a presentation. Present only very few references. (Naturally, this applies only to the talk itself, not to a possible handout.)
- If you present more references than fit on a single slide you can be almost sure that none of them will be remembered.
- Present references only if they are intended as "further reading." Do not present a list of all things you used like in a paper.
- You should not present a long list of all your other great papers *except* if you are giving an application talk.
- Using the `\cite` commands can be confusing since the audience has little chance of remembering the citations. If you cite the references, always cite them with full author name and year like "[Nobacon, 2003]" instead of something like "[2, 4]" or "[Nob01, ND02]".
- If you want to be modest, you can abbreviate your name when citing yourself as in "[Nickelsen and N., 2003]" or "[Nickelsen and N, 2003]". However, this can be confusing for the audience since it is often not immediately clear who exactly "N." might be. We recommend using the full name.

Adding an appendix You can add an appendix to your talk by using the `\appendix` command. You should put frames and perhaps whole subsections into the appendix that you do not intend to show during your presentation, but which might be useful to answer a question. The `\appendix` command essentially just starts a new part.

`\appendix`

`\appendix<`⟨*mode spec*⟩`>`

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

```
\begin{document}
\begin{frame}
  \maketitle
\end{frame}
\section*{Outline}
\begin{frame}
  \tableofcontents
\end{frame}
\section{Main Text}
\begin{frame}
  Some text
\end{frame}
\section*{Summary}
\begin{frame}
  Summary text
\end{frame}
\appendix
\section{Supplementary material}
\begin{frame}
  \tableofcontents
\end{frame}
\subsection{Additional material}
\begin{frame}
  Details
\end{frame}
\begin{frame}
  Text omitted in main talk.
\end{frame}
\subsection{Even more additional material}
\begin{frame}
  More details
\end{frame}
\end{document}
```

16.1.3 Frame structure

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

The frame title

- Put a title on each frame. The title explains the contents of the frame to people who did not follow all details on the slide.
- The title should really *explain* things, not just give a cryptic summary that cannot be understood unless one has understood the whole slide. For example, a title like "The Poset" will have everyone puzzled what this slide might be about. Titles

like “Review of the Definition of Partially Ordered Sets (Posets)” or “A Partial Ordering on the Columns of the Genotype Matrix” are *much* more informative.

- Ideally, titles on consecutive frames should “tell a story” all by themselves.
- In English, you should *either always* capitalize all words in a frame title except for words like “a” or “the” (as in a title), *or* you *always* use the normal lowercase letters. Do *not* mix this; stick to one rule. The same is true for block titles. For example, do not use titles like “A short Review of Turing machines.” Either use “A Short Review of Turing Machines.” or “A short review of Turing machines.” (Turing is still spelled with a capital letter since it is a proper name).
- In English, the title of the whole document should be capitalized, regardless of whether you capitalize anything else.
- In German and other languages that have lots of capitalized words, always use the correct upper-/lowercase letters. Never capitalize anything in addition to what is usually capitalized.

How much can I put on a frame?

- A frame with too little on it is better than a frame with too much on it. A usual frame should have between 20 and 40 words. The maximum should be at about 80 words.
- Do not assume that everyone in the audience is an expert on the subject matter. Even if the people listening to you should be experts, they may last have heard about things you consider obvious several years ago. You should always have the time for a quick reminder of what exactly a “semantical complexity class” or an “ω-complete partial ordering” is.
- Never put anything on a slide that you are not going to explain during the talk, not even to impress anyone with how complicated your subject matter really is. However, you may explain things that are not on a slide.
- Keep it simple. Typically, your audience will see a slide for less than 50 seconds. They will not have the time to puzzle through long sentences or complicated formulas.

Structuring a frame

- Use block environments like `block`, `theorem`, `proof`, `example`, and so on.
- Prefer `enumerate` and `itemize` environments over plain text.
- Use `description` when you define several things.
- Do not use more than two levels of “subitemizing.”
- Do not create endless `itemize` or `enumerate` lists.
- Do not uncover lists piecewise.

- Emphasis is an important part of creating structure. Use `\alert` to highlight important things. This can be a single word or a whole sentence. However, do not overuse highlighting since this will negate the effect.
- Use columns.
- *Never* use footnotes. They needlessly disrupt the flow of reading. Either what is said in the footnote is important and should be put in the normal text; or it is not important and should be omitted (*especially* in a presentation).
- Use `quote` or `quotation` to typeset quoted text.
- Do not use long bibliographies.

Writing the text

- Use short sentences.
- Prefer phrases over complete sentences. For example, instead of "The figure on the left shows a Turing machine, the figure on the right shows a finite automaton." try "Left: A Turing machine. Right: A finite automaton." Even better, turn this into an itemize or a description.
- Punctuate correctly: no punctuation after phrases, complete punctuation in and after complete sentences.
- *Never* use a smaller font size to "fit more on a frame."
- Do not hyphenate words. If absolutely necessary, hyphenate words "by hand", using the command `\-`.
- Break lines "by hand" using the command `\\`. Do not rely on automatic line breaking. Break where there is a logical pause. For example, good breaks in "the tape alphabet is larger than the input alphabet" are before "is" and before the second "the." Bad breaks are before either "alphabet" and before "larger."
- Text and numbers in figures should have the *same* size as normal text. Illegible numbers on axes usually ruin a chart and its message.

16.2 Using graphics

Graphics often convey concepts or ideas much more efficiently than text: A picture can say more than a thousand words. (Although, sometimes a word can say more than a thousand pictures.)

- Put (at least) one graphic on each slide, whenever possible. Visualizations help an audience enormously.
- Usually, place graphics to the left of the text. (Use the `columns` environment.) In a left-to-right reading culture, we look at the left first.
- Graphics should have the same typographic parameters as the text: Use the same fonts (at the same size) in graphics as in the main text. A small dot in a graphic should have exactly the same size as a small dot in a text. The line width should be the same as the stroke width used in creating the glyphs of the font. For example, an 11 pt non-bold Computer Modern font has a stroke width of 0.4 pt.

- While bitmap graphics, like photos, can be much more colorful than the rest of the text, vector graphics should follow the same "color logic" as the main text (like black = normal lines, red = highlighted parts, green = examples, blue = structure).
- Like text, you should explain everything that is shown on a graphic. Unexplained details make the audience puzzle whether this was something important that they have missed. Be careful when importing graphics from a paper or some other source. They usually have much more detail than you will be able to explain and should be radically simplified.
- Sometimes the complexity of a graphic is intentional and you are willing to spend much time explaining the graphic in great detail. In this case, you will often run into the problem that fine details of the graphic are hard to discern for the audience.

16.3 Choosing appropriate colors

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

17 Code interfaces

Some interfaces for customization are provided using the L3 programming layer; these are intended for programmers, and are currently somewhat experimental.

`\l_talk_frame_properties_clist`

Contains names of properties which should be recorded at the end of each frame. The properties should be defined using `\property_new:nnnn`. The class does not add any entries to this list.

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

A

\action *16*
\action *15, 16*
actionenv (env.) *15*
\addtocounter *17*
\againframe *23*
\againframe *23*
\alert *20*
\alert *10, 20, 32*
alertenv (env.) *20*
\alt *12*
\alt *12*
\appendix *30*
\appendix *23, 28, 30*
aspect-ratio (option) *8*
\author *19*
\author *19*

B

\begin *6, 13, 15, 19–21*
block (env.) *20*

C

\cite *30*
\color *10*
column (env.) *21*
columns (env.) *21*

D

\date *19*
\date *19*
description (env.) *19*
\DocumentMetadata *4*

E

\EditInstance *11*
\emph *10*
\end *6*
enumerate (env.) *19*
environments:
  actionenv *15*
  alertenv *20*
  block *20*
  column *21*
  columns *21*
  description *19*
  enumerate *19*
  frame *6*
  frame* *6*
  invisibleenv *13*
  itemize *19*
  onlyenv *13*
  overlayarea *13*
  overprint *13*
  uncoverenv *13*
  visibleenv *13*

F

font-size (option) *8*
\footnote *22*
\footnote *22*
\footnote[frame] *22*
frame (env.) *6*
frame* (env.) *6*
frame-title-arg (option) *7*
\framesubitle *7*
\framesubtitle *7*
\framesubtitle *7*
\frametitle *7*
\frametitle *5, 7*

I

\includegraphics *10*
\institute *19*
\institute *19*
\invisible *11*
\invisible *11, 13*
invisibleenv (env.) *13*
\item *12, 19*
\item *12, 15, 19, 20*
itemize (env.) *19*

L

\label *10*

M

\maketitle *18*
\maketitle *18*
\mathcolor *10*
\mathrm *5, 25*
mode (option) *23*

N

\newcommand *5*
\NewDocumentCommand *5*
\newenvironment *5*

O

\only *11*
\only *10, 11, 13, 14*
onlyenv (env.) *13*
\onslide *10*
\onslide *8–10*
options:
aspect-ratio *8*
font-size *8*
frame-title-arg *7*
mode *23*
tag-slides *25*
overlayarea (env.) *13*
overprint (env.) *13*

P

\part *18, 28*
\pause *8*
\pause *8, 9, 16, 17*
property commands:
\property_new:nnnn *34*

R

\reuseframe *22*
\reuseframe *22, 23*

S

\section *18, 30*
\setcounter *17*
\SetKeys *5*
\setmainfont *25*
\setsansfont *25*
\subsection *30*
\subtitle *19*
\subtitle *19*

T

tag-slides (option) *25*
talk commands:
\l_talk_frame_properties_clist . . *34*
\temporal *12*
\temporal *12*
\textbf *10*
\textcolor *10*
\textheight *23*
\textit *10*
\textmd *10*
\textnormal *10*
\textrm *5, 10*
\textsc *10*
\textsf *10*
\textsl *10*
\texttt *10*
\textup *10*
\textwidth *21*
\title *19*
\title *19*

U

\uncover *11*
\uncover *11, 13, 15*
uncoverenv (env.) *13*

V

\visible *11*
\visible *11, 13, 15*
visibleenv (env.) *13*