

ltx-talk – A class for typesetting presentations*

Joseph Wright[†]

Released 2025-07-12

Contents

1	Introduction	2
2	Submitting ideas	3
3	Simple example documents	3
4	Class structure and design decisions	4
5	Differences from beamer	4
6	Creating frames	5
6.1	The frame environment	5
6.2	Components of a frame	5
6.2.1	The frame title	5
6.3	Frame and margin size	6
6.4	Restricting the slides of a frame	6
7	Creating overlays	6
7.1	The <code>\pause</code> command	6
7.2	The general concept of overlay specifications	8
7.3	Commands with overlay specifications	8
7.4	Environments with overlay specifications	11
7.5	Dynamically changing text or images	11
7.6	Advanced overlay specifications	12
7.6.1	Mode specifications	12
7.6.2	Action specifications	13
7.6.3	Incremental specifications	14
8	Structuring a presentation: the local structure	16
8.1	Itemizations, enumerations and descriptions	16
8.2	Highlighting	17
8.3	Block environments	18
8.4	Splitting a frame into multiple columns	18

*This file describes v0.1.0, last revised 2025-07-12.

[†]E-mail: joseph@texdev.net

9	Creating alternative output formats	18
9.1	Creating handouts using the <code>handout</code> mode	19
10	Creating tagging PDF output	19
	Index	19

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 other) 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` mean 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 *highly* experimental: only a (small) subset of `beamer` functionality is implemented, some things are being done differently, almost everything is still subject to discussion.

2 Submitting ideas

As noted in Section 1, the class is at present *highly* 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.

3 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
  \end{itemize}
\end{frame}
\end{document}
```

```

    \item But this will only appear on slide three
  \end{itemize}
  Back to appearing on all slides
\end{frame}
\end{document}

```

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

4 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 sanserif using established standard implementations (currently `sansmathfonts` for pdfTeX and New Computer Modern for OpenType engines).

5 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* sanserif, in particular `\mathrm` and `\textrm` will give a sanserif 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. `\includgraphics*<...>`: 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`, in particular frame titles are given using `\frametitle` and not as `\begin{frame}{...}`
- New overlay-aware commands and environments should be defined using `ltxcmd` (`\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 L^AT_EX; in particular, the various math-focussed additions which beamer make (such as various theorem environments) are not included

6 Creating frames

6.1 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 **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 L^AT_EX 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.

6.2 Components of a frame

6.2.1 The frame title

`\frametitle`

```
\frametitle<{overlay spec}>[<options>]{<frame title>}
```

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.

`frame-title-arg` For compatibility with `beamer`, 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}
```

This interface is less flexible than using `\frametitle`, which is therefore the recommended approach.

6.3 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.

`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}
```

6.4 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).

7 Creating overlays

7.1 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.

7.2 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 7.3.

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.

7.3 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`, `\textcolor`; `\alert`, `\structure`. 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. For details on how to specify whether the text is invisible or just transparent see Section ??.

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

`\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 7.6.2; 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}

```

7.4 Environments with overlay specifications

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

`visibleenv` (*env.*) 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}

```

7.5 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 L^AT_EX 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}

```

7.6 Advanced overlay specifications

7.6.1 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 9 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.

7.6.2 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 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`.

7.6.3 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 the current value of the L^AT_EX counter `pauses`, which is 1 at the beginning of the frame. Then the counter is increased by 1, though it is only increased once for every overlay specification, even if the specification contains multiple + signs (they are replaced by the same number).

In the above example, the first specification is replaced by `<1->`. Then the second is replaced by `<2->` and so forth. We can now easily insert new entries, without having to change anything 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 L^AT_EX 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 thing as in `<+>` happens *except* that the counter `pauses` is *not* incremented and *except* that you get the value of `pauses` decreased by one. Thus a dot, possibly followed by an offset, just expands to the current value of the counter `pauses` minus one, 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->`.

8 Structuring a presentation: the local structure

L^AT_EX 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.

8.1 Itemizations, enumerations and descriptions

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

`\item` 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,
  \end{enumerate}
\end{frame}
```

```

\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 7.6.3, you can implement, for example, a stepwise uncovering of the items. The `<overlay specification>` is inherited by subenvironments. Naturally, in a subenvironment 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}

```

8.2 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.

8.3 Block environments

8.4 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 L^AT_EX'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 7.6.2. 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 7.6.2. The vertical placement of the enclosing `columns` environment can be overruled by specifying a specific `<placement>`.

9 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.

9.1 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”).

10 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.

`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 can 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.

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	<code>\DocumentMetadata</code>	3
<code>\action</code>	14		
<code>\action</code>	13, 14		
<code>actionenv</code> (env.)	14		
<code>\addtocounter</code>	15		
<code>\alert</code>	17		
<code>\alert</code>	8, 17		
<code>alertenv</code> (env.)	17		
<code>\alt</code>	10		
<code>\alt</code>	10		
<code>aspect-ratio</code> (option)	6		
	B		
<code>\begin</code>	5, 11, 12, 14, 17, 18		
	C		
<code>\color</code>	8		
<code>column</code> (env.)	18		
<code>columns</code> (env.)	18		
	D		
<code>description</code> (env.)	16		
		<code>\emph</code>	8
		<code>\end</code>	5
		<code>enumerate</code> (env.)	16
		environments:	
		<code>actionenv</code>	14
		<code>alertenv</code>	17
		<code>column</code>	18
		<code>columns</code>	18
		<code>description</code>	16
		<code>enumerate</code>	16
		<code>frame</code>	5
		<code>frame*</code>	5
		<code>invisibleenv</code>	11
		<code>itemize</code>	16
		<code>onlyenv</code>	11
		<code>overlayarea</code>	11
		<code>overprint</code>	12
		<code>uncoverenv</code>	11

