ltx-talk – A class for typesetting presentations*

Joseph Wright†

Released 2025-10-10

Contents

*This file describes v0.2.3, last revised 2025-10-10.

†E-mail: joseph@texdev.net

Part I
ltx-talk – Overall set up

1 ltx-talk implementation

Start the DocStrip guards.

```
1 ⟨*class⟩
```

Identify the internal prefix.

```
2 ⟨@@=talk⟩
```

1.1 Set up

Identify the package and give the over all version information.

```
3 \ProvidesExplClass {ltx-talk} {2025-10-10} {0.2.3}
4   {A class for typesetting presentations}
```

Get the right type of message.

```
5 \prop_gput:Nnn \g_msg_module_name_prop { talk } { ltx-talk }
6 \prop_gput:Nnn \g_msg_module_type_prop { talk } { Class }
```

Require the latest LaTeX structures.

```
7 \IfFormatAtLeastF{2025-11-01}
8   {
9     \msg_new:nnnn { ltx-talk } { kernel-too-old }
10      { The~ltx-talk~class~requires~LaTeX~2025-11-01~or~later. }
11      {
12        You~have~tried~to~use~the~ltx-talk~class~with~a~LaTeX~kernel~release~
13        prior~to~2025-11-01;~the~required~functionality~is~missing. \\ \\
14        At~present,~you~may~need~to~use~the~development~release~of~LaTeX,~
15        invoked~using~"lualatex-dev"~or~"pdflatex-dev".
16      }
17    \msg_fatal:nn { ltx-talk } { kernel-too-old }
18  }
19 \NeedsDocumentMetadata
```

1.2 Additions for expl3

Like `\vcoffin_set:Nnn`, so should be an easy enough addition.

```
20 \cs_gset_protected:Npn \vbox_set_to_wd:Nnn #1#2#3
21   {
22     \tex_setbox:D #1 \tex_vbox:D
23       {
24         \tex_hsize:D \__box_dim_eval:n {#2}
25         \color_group_begin: #3 \par \color_group_end:
26       }
27     \box_dp:N #1 \__box_dim_eval:n {#2}
28   }
29 \cs_gset_protected:Npn \vbox_set_to_wd:Nnw #1#2
30   {
31     \cs_set_protected:Npn \__box_set_to_wd:
32       { \box_wd:N #1 \__box_dim_eval:n {#2} }
```

```
    \tex_setbox:D #1 \tex_vbox:D
      \c_group_begin_token
        \tex_hsize:D \__box_dim_eval:n {#2}
        \group_insert_after:N \__box_set_to_wd:
        \color_group_begin:
  }
```

Some things from xbox that would be useful.

```
\cs_gset_protected:Npn \rule:nnn #1#2#3
  {
    \tex_vrule:D
      height \dim_eval:n {#2} \exp_stop_f:
      depth  \dim_eval:n {#3} \exp_stop_f:
      width  \dim_eval:n {#1} \exp_stop_f:
      \scan_stop:
  }
```

1.3 Extra variants

```
\cs_generate_variant:Nn \clist_set:Nn { cv }
\cs_generate_variant:Nn \hook_gput_code:nnn { nne }
\exp_args_generate:n { nVv }
\cs_generate_variant:Nn \color_select:n { V }
\cs_generate_variant:Nn \dim_compare:nNnTF { v }
\cs_generate_variant:Nn \dim_compare_p:nNn { vNv }
\cs_generate_variant:Nn \dim_max:nn { v }
\cs_generate_variant:Nn \str_replace_all:Nnn { NnV }
\cs_generate_variant:Nn \text_purify:n { v }
\cs_generate_variant:Nn \vbox_to_ht:nn { v }
```

1.4 Scratch space

__talk_tmp:w

For one-off processing.

```
\cs_new_protected:Npn \__talk_tmp:w { }
```

(*End of definition for* `\__talk_tmp:w`.)

\l__talk_tmp_box

```
\box_new:N \l__talk_tmp_box
```

(*End of definition for* `\l__talk_tmp_box`.)

\l__talk_tmp_tl

```
\tl_new:N \l__talk_tmp_tl
```

(*End of definition for* `\l__talk_tmp_tl`.)

1.5 Option handling

\l__talk_aspect_ratio_str
\l__talk_fontsize_dim
\l__talk_frame_title_bool
\l__talk_mode_str

```
\keys_define:nn { talk }
  {
    aspect-ratio .str_set:N =
      \l__talk_aspect_ratio_str ,
    font-size .dim_set:N =
      \l__talk_fontsize_dim ,
    frame-title-arg .bool_set:N =
      \l__talk_frame_title_bool ,
    handout .code:n =
      { \str_set:Nn \l__talk_mode_str { handout } } ,
    handout .value_forbidden:n = true ,
    mode .choices:nn =
      { handout , projector }
      { \str_set:NV \l__talk_mode_str \l_keys_choice_tl }
  }
```

(*End of definition for* `\l__talk_aspect_ratio_str` *and others.*)

Scope for options.

```
\keys_define:nn { talk }
  {
    aspect-ratio    .usage:n = load ,
    font-size       .usage:n = load ,
    frame-title-arg .usage:n = load ,
    mode            .usage:n = load
  }
```

Initial values.

```
\keys_set:nn { talk }
  {
    aspect-ratio    = 16:9      ,
    font-size       = 11pt      ,
    frame-title-arg = false     ,
    mode            = projector
  }
\ProcessKeyOptions [ talk ]
```

1.6 Setting up

Load the font size setup if available, otherwise fall back on scaling.

```
\file_if_exist_input:nF { size \dim_to_decimal:n \l__talk_fontsize_dim .clo }
  {
    \file_input:n { size10.clo }
    \RequirePackage { relsize }
    \hook_gput_code:nne { begindocument } { talk }
      { \exp_not:N \relsize { \fp_eval:n { \l__talk_fontsize_dim / 10pt } } }
  }
```

\c__talk_paper_height_dim
\c__talk_paper_width_dim

As geometry is being used to set the paper size with no previous value, we have to use the optional argument rather than waiting to apply `\geometry`.

```
\dim_const:Nn \c__talk_paper_height_dim { 100mm }
```

```
\use:e
  {
    \cs_set_protected:Npn \exp_not:N \__talk_tmp:w
      #1 \tl_to_str:n { : } #2 \tl_to_str:n { : } #3 \exp_not:N \q_stop
      {
        \dim_const:Nn \exp_not:N \c__talk_paper_width_dim
          {
            \exp_not:N \fp_to_dim:n
              { (#1 / #2) * \exp_not:N \c__talk_paper_height_dim }
          }
      }
    \exp_not:N \__talk_tmp:w \l__talk_aspect_ratio_str
      \tl_to_str:n { : } 100 \exp_not:N \q_stop
  }
\use:e
  {
    \exp_not:N \RequirePackage
      [
        papersize =
          {
            \dim_use:N \c__talk_paper_width_dim  ,
            \dim_use:N \c__talk_paper_height_dim
          } ,
        tmargin    = 10mm ,
        bmargin    =  8mm ,
        lmargin    = 10mm ,
        rmargin    = 10mm ,
        headheight = 10mm ,
        headsep    =  2mm ,
        footskip   =  6mm
      ]
      { geometry }
  }
```

(*End of definition for* `\c__talk_paper_height_dim` *and* `\c__talk_paper_width_dim`.)

Turn off justification

```
\raggedright
```

1.7 Math support

We always require amsmath: this is forced anyway by unicode-math for LuaTEX.

```
\RequirePackage { amsmath }
```

1.8 Font selection

The aim here is to minimize change from the standard font setup but at the same time provide a sans-serif default. Since beamer was released, better sans-serif math mode fonts have become available. For OpenType engines, requiring unicode-math is the most sensible approach. The New Computer Modern font provides a reasonable initial set of glyphs. It comes with a wrapper package, but that does various other things: if the user wants these, they can choose to load themselves. For 8-bit engines, switching the text font to be sans-serif is easy. For math mode, the sansmathfonts package does a good job: here, using the package rather than adjusting directly is the sensible option.

```
\sys_if_engine_opentype:TF
  {
    \RequirePackage { unicode-math }
    \setsansfont { NewCMSans10-Regular.otf }
    \setmathfont { NewCMSansMath-Regular.otf }
  }
  {
    \RequirePackage { sansmathfonts }
    \RequirePackage [ nomath ] { lmodern }
  }
\cs_set_eq:NN \rmdefault \sfdefault
```

1.9 Hyperlinks

\thepage We define \thepage here: this is checked for by hyperref so has to come early.

```
\cs_new:Npn \thepage { \@arabic \c@page }
```

(*End of definition for* \thepage*. This variable is documented on page* **??**.)

A requirement.

```
\RequirePackage { hyperref }
\hypersetup { hidelinks }
```

1.10 Tagging

We need to extend the standard tagging model to work with slides and so on.

```
\tagpdfsetup
  {
    role / user-NS = ltx-talk        ,
    role / new-tag = frame / Sect    ,
    role / new-tag = frametitle / H4
  }
```

⟨/class⟩

Part II
ltx-talk-color – Color definitions

1 ltx-talk-color implementation

Start the DocStrip guards.

```
1 ⟨*class⟩
```

Identify the internal prefix.

```
2 ⟨@@=talk⟩
```

The aim here is to *test* how well l3color can support the range of color functions that are needed for a presentation. As such, this is very much experimental, but deliberately so. In particular, there is an important question about the need for global colors: used throughout beamer but otherwise not widely encountered. At the same time, there is a need to work with packages that expect color to be managed in a predictable way: pgf in particular makes use of xcolor internal as part of color management.

Currently, colors defined using xcolor will be passed on to l3color provided `\DocumentMetadata` is active. As that is a requirement in any case for ltx-talk, some of the setup is relatively easy to do.

1.1 Existing definitions

```
3 \RequirePackage { xcolor }
```

`\stdcolor` `\stdmathcolor` `\stdtextcolor` Save the document commands.

```
4 \NewCommandCopy \stdcolor \color
5 \NewCommandCopy \stdmathcolor \mathcolor
6 \NewCommandCopy \stdtextcolor \textcolor
```

(*End of definition for* `\stdcolor`, `\stdmathcolor`, *and* `\stdtextcolor`. *These functions are documented on page* ??.)

1.2 Document commands

```
7 \cs_generate_variant:Nn \color_select:n { e }
8 \cs_generate_variant:Nn \color_select:nn { ne }
9 \cs_generate_variant:Nn \color_math:nn { e }
10 \cs_generate_variant:Nn \color_math:nnn { ne }
```

`\color` `\mathcolor` `\textcolor` Add the overlay specification and use l3color.

```
11 \RenewDocumentCommand \color { D <> { all } o m }
12   {
13     \__talk_if_overlay:nT {#1}
14       {
15         \IfNoValueTF {#2}
16           { \color_select:e {#3} }
17           { \color_select:ne {#2} {#3} }
18       }
19   }
20 \RenewDocumentCommand \mathcolor { D <> { all } o m +m }
21   {
22     \__talk_if_overlay:nT {#1}
```

```
      {
        \IfNoValueTF {#2}
          { \color_math:en {#3} {#4} }
          { \color_math:nen {#2} {#3} {#4} }
      }
  }
\RenewDocumentCommand \textcolor { D <> { all } o m +m }
  {
    \__talk_if_overlay:nT {#1}
      {
        \mode_leave_vertical:
        \group_begin:
          \IfNoValueTF {#2}
            { \color_select:e {#3} }
            { \color_select:ne {#2} {#3} }
          #4
        \group_end:
      }
  }
```

(*End of definition for* `\color`, `\mathcolor`, *and* `\textcolor`. *These functions are documented on page* ??.)

1.3 Color definition

`\DeclareColor` Provide a single interface here: as the data will be passed to l3color in any case, there is not too much to do.

```
\NewDocumentCommand \DeclareColor { m o m }
  {
    \IfNoValueTF {#2}
      { \colorlet {#1} {#3} }
      { \definecolor {#1} {#2} {#3} }
  }
```

(*End of definition for* `\DeclareColor`. *This function is documented on page* ??.)

1.4 Semantic colors

Pick up the standard colors from beamer.

```
\DeclareColor { alert } [ RGB ] { 200 , 0 , 0 }
\DeclareColor { example } { green!50!black }
\DeclareColor { structure } [ rgb ] { 0.2 , 0.2 , 0.7 }
```

⟨/class⟩

Part III
ltx-talk-decode – Decoding overlay specs

1 ltx-talk-decode implementation

Start the DocStrip guards.

```
1 ⟨*class⟩
```

Identify the internal prefix.

```
2 ⟨@@=talk⟩
```

\l__talk_decode_overlays_bool

The result from decoding: are we on the current slide. This may well be better handled by moving to a `TF` signature: to be explored.

```
3 \bool_new:N \l__talk_decode_overlays_bool
```

(*End of definition for* `\l__talk_decode_overlays_bool`.)

\g__talk_pauses_int
\c@pauses
\thepauses

The automatically-incremented value for the relative overlay value.

```
4 \int_new:N \g__talk_pauses_int
5 \cs_new_eq:NN \c@pauses \g__talk_pauses_int
6 \cs_new:Npn \thepauses { \@arabic \g__talk_pauses_int }
```

(*End of definition for* `\g__talk_pauses_int`, `\c@pauses`, *and* `\thepauses`. *These variables are documented on page* ??.)

\l__talk_decode_pure_bool

Tracks whether only mode specifications were given.

```
7 \bool_new:N \l__talk_decode_pure_bool
```

(*End of definition for* `\l__talk_decode_pure_bool`.)

\l__talk_decode_step_bool

Tracks whether to step `\g__talk_pauses_int`.

```
8 \bool_new:N \l__talk_decode_step_bool
```

(*End of definition for* `\l__talk_decode_step_bool`.)

\l__talk_decode_arg_str

For error usage.

```
9 \str_new:N \l__talk_decode_arg_str
```

(*End of definition for* `\l__talk_decode_arg_str`.)

\l__talk_decode_overlays_clist
\l__talk_decode_overlays_str

The decoded overlay specification: will have only absolute slide numbers present, potentially along with ranges.

```
10 \clist_new:N \l__talk_decode_overlays_clist
11 \str_new:N \l__talk_decode_overlays_str
```

(*End of definition for* `\l__talk_decode_overlays_clist` *and* `\l__talk_decode_overlays_str`.)

\l__talk_decode_action_str

The action which is active, if any.

```
12 \str_new:N \l__talk_decode_action_str
```

(*End of definition for* `\l__talk_decode_action_str`.)

\l__talk_decode_actions_bool
\l__talk_decode_actions_clist
\l__talk_decode_actions_str

For the actions versions of overlay tracking.

```
\bool_new:N \l__talk_decode_actions_bool
\clist_new:N \l__talk_decode_actions_clist
\str_new:N \l__talk_decode_actions_str
```

(*End of definition for* `\l__talk_decode_actions_bool`, `\l__talk_decode_actions_clist`, *and* `\l__talk_decode_actions_str`.)

__talk_decode_parse:n
__talk_decode_parse_aux:n
__talk_decode_parse:w

First a simple check for an entirely blank argument: if that's the case, there is no additional overlay to consider. Then deal with any category code issues before looping over blocks divided by | tokens.

```
\cs_new_protected:Npn \__talk_decode_parse:n #1
  {
    \str_clear:N \l__talk_decode_action_str
    \bool_lazy_or:nnTF
      { \tl_if_blank_p:n {#1} }
      { \str_if_eq_p:nn {#1} { all } }
      { \bool_set_true:N \l__talk_decode_overlays_bool }
      {
        \str_set:Nn \l__talk_decode_arg_str {#1}
        \bool_set_false:N \l__talk_decode_actions_bool
        \bool_set_false:N \l__talk_decode_overlays_bool
        \bool_set_true:N \l__talk_decode_pure_bool
        \str_clear:N \l__talk_decode_overlays_str
        \str_clear:N \l__talk_decode_actions_str
        \exp_args:No \__talk_decode_parse_aux:n { \l__talk_decode_arg_str }
      }
  }
\cs_new_protected:Npn \__talk_decode_parse_aux:n #1
  { \__talk_decode_parse:w #1 | \q_recursion_tail | \q_recursion_stop }
```

The end-of-loop test here covers the case where the active mode is not mentioned at all in the specification.

```
\cs_new_protected:Npn \__talk_decode_parse:w #1 |
  {
    \quark_if_recursion_tail_stop_do:nn {#1}
      {
        \bool_lazy_and:nnT
          { \str_if_empty_p:N \l__talk_decode_overlays_str }
          { ! \l__talk_decode_pure_bool }
          { \bool_set_true:N \l__talk_decode_overlays_bool }
      }
    \exp_args:Ne \__talk_decode_mode:n
      { \tl_trim_spaces:n {#1} }
    \__talk_decode_parse:w
  }
```

(*End of definition for* `\__talk_decode_parse:n`, `\__talk_decode_parse_aux:n`, *and* `\__talk_decode_parse:w`.)

\c__talk_modes_clist

The possible modes: detokenized as that is applied up-front in decoding.

```
\clist_const:Ne \c__talk_modes_clist
  {
    \tl_to_str:n { handout }   ,
    \tl_to_str:n { projector }
  }
```

(*End of definition for* `\c__talk_modes_clist`.)

`\__talk_decode_mode:n` `\__talk_decode_mode:w` `\__talk_decode_mode_aux:n`

Check if the mode is known and current. If we find an action but have no overlay details, they are filled in with a `*`.

```
\cs_new_protected:Npe \__talk_decode_mode:n #1
  {
    \clist_if_in:NnTF \exp_not:N \c__talk_modes_clist {#1}
      {
        \exp_not:N \str_if_eq:VnT
          \exp_not:N \l__talk_mode_str {#1}
          { \bool_set_true:N \exp_not:N \l__talk_decode_overlays_bool }
      }
      {
        \exp_not:N \__talk_decode_mode:w #1 \tl_to_str:n { : : }
          \exp_not:N \q_stop
      }
  }
\use:e
  {
    \cs_new_protected:Npe \exp_not:N \__talk_decode_mode:w
      #1 \token_to_str:N :
      #2 \token_to_str:N :
      #3 \exp_not:N \q_stop
  }
  {
    \exp_not:N \tl_if_blank:nTF {#2}
      {
        \exp_not:N \__talk_decode_mode:nn
          { \tl_to_str:n { projector } } {#1}
      }
      { \exp_not:N \__talk_decode_mode:nn {#1} {#2} }
  }
\cs_new_protected:Npn \__talk_decode_mode:nn #1#2
  {
    \str_if_eq:VnTF \l__talk_mode_str {#1}
      {
        \__talk_decode_action:n {#2}
        \str_if_empty:NT \l__talk_decode_overlays_str
          { \__talk_decode_overlays:nn { overlays } { * } }
      }
      {
        \tl_if_blank:nF {#2}
          { \bool_set_false:N \l__talk_decode_pure_bool }
      }
  }
```

(*End of definition for* `\__talk_decode_mode:n`, `\__talk_decode_mode:w`, *and* `\__talk_decode_mode_aux:n`.)

`\__talk_decode_action:n` `\__talk_decode_action:w`

Here, we have two valid possibilities: no action specification at all, or from the known list. If we don't find one of those outcomes, we can issue an error.

```
\cs_new_protected:Npe \__talk_decode_action:n #1
  {
    \exp_not:N \__talk_decode_action:w
```

```
      #1 \tl_to_str:n { @ @ } \exp_not:N \q_stop
  }
\use:e
  {
    \cs_new_protected:Npn \exp_not:N \__talk_decode_action:w
      #1 \tl_to_str:n { @ } #2 \tl_to_str:n { @ } #3 \exp_not:N \q_stop
  }
  {
    \tl_if_blank:nTF {#2}
      { \__talk_decode_overlays:nn { overlays } {#1} }
      {
        \cs_if_exist:cTF { __talk_action_ #1 :N }
          {
            \bool_set_false:N \l__talk_decode_pure_bool
            \str_set:Nn \l__talk_decode_action_str {#1}
            \tl_if_blank:nF {#2}
              { \__talk_decode_overlays:nn { actions } {#2} }
          }
          {
            \msg_error:nnV { talk } { bad-action-spec }
              \l__talk_decode_arg_str
           }
      }
  }
```

(*End of definition for* `\__talk_decode_action:n` *and* `\__talk_decode_action:w`.)

`\__talk_decode_overlays:nn`
`\__talk_decode_overlays:nN`
`\@_decode_overlay_+:nw`
`\__talk_decode_overlay_.:nw`
`\__talk_decode_overlay_aux:nNN`
`\__talk_decode_overlay_offset:nNnN`
`\__talk_decode_overlay_offset:nNn`

The loop here needs to replace all `+` and `.` characters by the current automatic value, allowing for any offsets. This step also needs to track whether to increment the automatic value: true if a `+` is seen, false otherwise.

```
\cs_new_protected:Npn \__talk_decode_overlays:nn #1#2
  {
    \bool_set_false:N \l__talk_decode_step_bool
    \__talk_decode_overlays:nN {#1} #2 \q_recursion_tail \q_recursion_stop
    \bool_if:NT \l__talk_decode_step_bool
      { \int_gincr:N \g__talk_pauses_int }
    \__talk_decode_check:n {#1}
  }
\cs_new_protected:Npn \__talk_decode_overlays:nN #1#2
  {
    \quark_if_recursion_tail_stop:N #2
    \cs_if_exist_use:cF { __talk_decode_overlay_ #2 :nw }
      {
        \str_put_right:cn { l__talk_decode_ #1 _str } {#2}
        \__talk_decode_overlays:nN
      }
        {#1}
  }
\cs_new_protected:cpn { __talk_decode_overlay_+:nw } #1
  {
    \bool_set_true:N \l__talk_decode_step_bool
    \__talk_decode_overlay_aux:nNN {#1} 1
  }
\cs_new_protected:cpn { __talk_decode_overlay_.:nw } #1
```

```
  { \__talk_decode_overlay_aux:nNN {#1} 0 }
```

The look-ahead for an offset to a relative specification. If the end-of-loop is reached, the value still needs to be inserted: to share auxiliaries, that is done by using the same function as elsewhere, so the end-of-loop markers are re-inserted. Otherwise, there is a check to see if the next token is a `(`.

```
\cs_new_protected:Npn \__talk_decode_overlay_aux:nNN #1#2#3
  {
    \quark_if_recursion_tail_stop_do:Nn #3
      {
        \__talk_decode_overlay_offset:nNn {#1} #2 { 0 }
        \q_recursion_tail \q_recursion_stop
      }
    \token_if_eq_meaning:NNTF #3 ( % )
      { \__talk_decode_overlay_offset:nNnN {#1} #2 { } }
      { \__talk_decode_overlay_offset:nNn {#1} #2 { 0 } #3 }
  }
```

For the end of an offset, any valid overlay specification must have a closing `)`, so this time the end-of-loop case is an error. Otherwise simply collect up tokens until the closing `)` is found.

```
\cs_new_protected:Npn \__talk_decode_overlay_offset:nNnN #1#2#3#4
  {
    \quark_if_recursion_tail_stop_do:Nn #4
      {
        \msg_error:nnV { talk } { bad-action-spec }
          \l__talk_decode_arg_str
      } % (
    \token_if_eq_meaning:NNTF #4 )
      { \__talk_decode_overlay_offset:nNn {#1} #2 {#3} }
      { \__talk_decode_overlay_offset:nNnN {#1} #2 {#3#4} }
  }
```

Overlay values can never be negative: this is enforced here.

```
\cs_new_protected:Npn \__talk_decode_overlay_offset:nNn #1#2#3
  {
    \str_put_right:ce { l__talk_decode_ #1 _str }
      { \int_max:nn { 0 } { #3 + \g__talk_pauses_int + #2 } }
    \__talk_decode_overlays:nN {#1}
  }
```

(*End of definition for* `\__talk_decode_overlays:nn` *and others. This function is documented on page* ??.)

`\__talk_decode_check:n`
`\__talk_decode_check:nw`
`\__talk_decode_check_single:nn`
`\__talk_decode_check_range:nnn`

At this stage we have a fully "written out" overlay specification, and need to work out if the current slide is included. We need to look at each entry in the comma-separated list to sort this out. First we filter out the case of a `*`, then it's a question of working out whether each entry is a single number or a range, and if the latter, whether it's open at either the start or the end.

```
\cs_new_protected:Npn \__talk_decode_check:n #1
  {
    \clist_set:cv { l__talk_decode_ #1 _clist } { l__talk_decode_ #1 _str }
    \clist_if_in:cnTF { l__talk_decode_ #1 _clist } { * }
      { \bool_set_true:c { l__talk_decode_ #1 _bool } }
      {
```

```
        \clist_map_inline:cn { l__talk_decode_ #1 _clist }
          { \__talk_decode_check:nw {#1} 0 ##1 - - \q_stop }
      }
  }
```

If `#4` is empty, both of the "filler" `-` tokens were consumed: we have a single value. Otherwise there is a range: the setup above ensures that there will be a starting value in all cases due to the leading `0`, but there may not be an end one.

```
\cs_new_protected:Npn \__talk_decode_check:nw #1#2 - #3 - #4 \q_stop
  {
    \tl_if_empty:nTF {#4}
      { \__talk_decode_check_single:nn {#1} {#2} }
      {
        \tl_if_blank:nTF {#3}
          { \__talk_decode_check_range:nnn {#1} {#2} { \c_max_int } }
          { \__talk_decode_check_range:nnn {#1} {#2} {#3} }
      }
  }
\cs_set_protected:Npn \__talk_decode_check_single:nn #1#2
  {
    \int_compare:nNnTF \g__talk_slide_int = {#2}
      {
        \bool_set_true:c { l__talk_decode_ #1 _bool }
        \clist_map_break:
      }
      {
        \int_compare:nNnT {#2} > \g__talk_slide_int
          { \bool_gset_true:N \g__talk_slide_continue_bool }
      }
  }
```

TODO: In the following we might want to add a check whether the range was given with `#2` being smaller than `#3`, to be decided upon.

```
\cs_set_protected:Npn \__talk_decode_check_range:nnn #1#2#3
  {
    \int_compare:nNnF \g__talk_slide_int > {#3}
      {
        \int_compare:nNnTF \g__talk_slide_int < {#2}
          { \bool_gset_true:N \g__talk_slide_continue_bool }
          {
            \bool_set_true:c { l__talk_decode_ #1 _bool }
            \bool_lazy_and:nnT
              { \int_compare_p:nNn \g__talk_slide_int < {#3} }
              { \int_compare_p:nNn {#3} < \c_max_int }
              { \bool_gset_true:N \g__talk_slide_continue_bool }
            \clist_map_break:
          }
      }
  }
```

(*End of definition for* `\__talk_decode_check:n` *and others.*)

```
\msg_new:nnnn { talk } { bad-action-spec }
  { Bad~overlay~specification~"#1". }
  {
    The~overlay~specification~given~doesn't~follow~the~pattern~described~in~
```

```
    the~ltx-talk~manual:~it~has~been~ignored.
  }
```
⟨/class⟩

Part IV
ltx-talk-frame – The structure of frames

1 ltx-talk-frame implementation

Start the DocStrip guards.

```
1 ⟨*class⟩
```

Identify the internal prefix.

```
2 ⟨@@=talk⟩
```

1.1 Slides in frames

Currently, each slide in a frame will produce a separate page in the output (unless the slide is suppressed entirely). Material is then hidden on some pages by using opacity. An alternative approach would be to use Optional Content Groups to have a similar effect on one page per frame. However, whilst that would be relatively clear for appear/disappear effects, it would be much less straight-forward for partial transparency, *etc.*, plus would depend more heavily on viewer support. At a future stage we may wish to revisit this.

\g__talk_slide_continue_bool

Tracks whether the frame continues after the current slide.

```
3 \bool_new:N \g__talk_slide_continue_bool
```

(*End of definition for* `\g__talk_slide_continue_bool`.)

\l__talk_slide_box

```
4 \box_new:N \l__talk_slide_box
```

(*End of definition for* `\l__talk_slide_box`.)

\g__talk_slide_int
\c@slide
\theslide

The slide number inside the current frame: needed to know which overlays are active. We also provide LaTeX counter-style access.

```
5 \int_new:N \g__talk_slide_int
6 \cs_new_eq:NN \c@slide \g__talk_slide_int
7 \cs_new:Npn \theslide { \@arabic \c@slide }
```

(*End of definition for* `\g__talk_slide_int`, `\c@slide`, *and* `\theslide`. *These variables are documented on page* ??.)

Required to know which is the last slide in a frame for tagging.

```
8 \property_new:nnnn { slides } { now } { 1 } { \int_use:N \g__talk_slide_int }
```

__talk_slide:nn
__talk_slide_aux:n

Each slide is parsed inside simple set up, the only complexity being if we are handling fragile frames. There, all `\obeyedline` in the grabbed tokens need to be turned back into `^^M` before rescanning: this ensures that any verbatim grabbing in the frame still works. The strange business with setting the continuation boolean is needed as otherwise we get an infinite loop if there is an overlay specification for the frame itself. Tagging should not of itself force slide continuation, so the global boolean is reset for the tagged slide.

```
9 \cs_new_protected:Npn \__talk_slide:nn #1#2
10  {
```

```
    \group_begin:
      \tl_set:Ne \l__talk_tmp_tl
        {
          \property_ref:ee { frame . \int_use:N \g__talk_frame_int }
            { slides }
        }
      \str_if_eq:VnTF \l__talk_frame_tagging_str { n }
        { \str_set:NV \l__talk_frame_tagging_str \l__talk_tmp_tl }
        {
          \str_replace_all:NnV \l__talk_frame_tagging_str { ,n }
            \l__talk_tmp_tl
          \str_replace_all:NnV \l__talk_frame_tagging_str { ,~n }
            \l__talk_tmp_tl
        }
      \int_gzero:N \g__talk_slide_int
      \RenewCommandCopy \frame \__talk_latexe_frame:n
      \bool_do_while:Nn \g__talk_slide_continue_bool
        {
          \int_gincr:N \g__talk_slide_int
          \bool_gset_false:N \g__talk_slide_continue_bool
          \__talk_if_overlay:nT {#1}
            {
              \__talk_slide_begin:
              \__talk_if_overlay:VTF \l__talk_frame_tagging_str
                {
                  \bool_gset_false:N \g__talk_slide_continue_bool
                  \__talk_frame_tag:n
                }
                {
                  \bool_gset_false:N \g__talk_slide_continue_bool
                  \__talk_frame_notag:n
                }
                  {
                    \bool_if:NTF \l__talk_frame_verb_bool
                      { \__talk_slide_aux:n }
                      { \use:n }
                        {#2}
                  }
               \__talk_slide_end:
            }
        }
      \property_record:ee { frame . \int_use:N \g__talk_frame_int }
        { slides }
    \group_end:
  }
\cs_new_protected:Npn \__talk_slide_aux:n #1
  {
    \group_begin:
      \cs_set:Npn \obeyedline { ^^J }
      \use:e
        {
          \group_end:
          \tl_retokenize:n {#1}
        }
```

```
  }
```

(*End of definition for* `\__talk_slide:nn` *and* `\__talk_slide_aux:n`.)

The very last frame will not be recorded by the above, so we have to add to the hook at the very end of the run.

```
\AddToHook { enddocument / afterlastpage }
  {
    \property_record:ee { frame . \int_use:N \g__talk_frame_int }
      { slides }
  }
```

`\g__talk_frame_struct_int` The tagging structure number for the slide: needed by the content placed outside of the current box (for example the frame title).

```
\int_new:N \g__talk_frame_struct_int
```

(*End of definition for* `\g__talk_frame_struct_int`.)

`\__talk_slide_begin:`
`\__talk_slide_end:`

```
\cs_new_protected:Npn \__talk_slide_begin:
  {
    \int_gzero:N \g__talk_pauses_int
    \tl_gclear:N \g__talk_frame_title_tl
    \tl_gclear:N \g__talk_frame_subtitle_tl
    \__talk_cnt_save:
    \vbox_set:Nw \l__talk_slide_box
      \tl_gclear:N \g__talk_onslide_tl
  }
\cs_new_protected:Npn \__talk_slide_end:
  {
      \tl_use:N \g__talk_onslide_tl
    \vbox_set_end:
    \bool_if:NT \g__talk_slide_continue_bool
      { \__talk_cnt_restore: }
    \vbox_to_ht:nn { \textheight }
      {
        \use:c { __talk_slide_align_ \l__talk_frame_alignment_tl :n }
          { \vbox_unpack_drop:N \l__talk_slide_box }
      }
    \clearpage
  }
```

(*End of definition for* `\__talk_slide_begin:` *and* `\__talk_slide_end:`.)

`\__talk_slide_align_bottom:n` A pretty standard abstraction: we make sure there are always two skips.
`\__talk_slide_align_center:n`
`\__talk_slide_align_stretch:n`
`\__talk_slide_align_top:n`

```
\cs_new_protected:Npn \__talk_slide_align_bottom:n #1
  {
    \skip_vertical:n { 0pt~plus~1fil }
    #1
    \skip_vertical:n { 0pt }
  }
\cs_new_protected:Npn \__talk_slide_align_center:n #1
  {
    \skip_vertical:n { 0pt~plus~0.5fil }
    #1
```

```
    \skip_vertical:n { 0pt~plus~0.5fil }
  }
\cs_new_protected:Npn \__talk_slide_align_stretch:n #1
  {
    \skip_vertical:n { 0pt }
    #1
    \skip_vertical:n { 0pt }
  }
\cs_new_protected:Npn \__talk_slide_align_top:n #1
  {
    \skip_vertical:n { 0pt }
    #1
    \skip_vertical:n { 0pt~plus~1fil }
  }
```

(*End of definition for* `\__talk_slide_align_bottom:n` *and others.*)

1.2 Counters

`\l__talk_cnt_reset_seq`

As `\stepcounter`, *etc.*, will increment at each overlay, there is a need to save and reset. The list will be finalized at the end of the preamble, so the data storage is created at that point. The starting point is counters created before the class is loaded (other than those for lists, which reset "naturally"). Other cases are handled by adding to `\newcounter`.

```
\seq_new:N \l__talk_cnt_reset_seq
\seq_set_from_clist:Nn \l__talk_cnt_reset_seq
  {
    equation       ,
    footnote       ,
    mpfootnote     ,
    parentequation
  }
\seq_map_inline:Nn \l__talk_cnt_reset_seq
    {
      \int_new:c { g__talk_saved_ #1 _int }
      \int_gset_eq:cc { g__talk_saved_ #1 _int } { c@ #1 }
    }
```

(*End of definition for* `\l__talk_cnt_reset_seq`.)

`\__talk_cnt_save:`
`\__talk_cnt_restore:`

A simple save-and-restore pair.

```
\cs_new_protected:Npn \__talk_cnt_save:
  {
    \seq_map_inline:Nn \l__talk_cnt_reset_seq
      { \int_gset_eq:cc { g__talk_saved_ ##1 _int } { c@ ##1 } }
  }
\cs_new_protected:Npn \__talk_cnt_restore:
  {
    \seq_map_inline:Nn \l__talk_cnt_reset_seq
      { \int_gset_eq:cc { c@ ##1 } { g__talk_saved_ ##1 _int } }
  }
```

(*End of definition for* `\__talk_cnt_save:` *and* `\__talk_cnt_restore:`.)

\@definecounter
\std@definecounter

Track all counters for resetting.

```
\cs_new_eq:NN \std@definecounter \@definecounter
\cs_gset_protected:Npn \@definecounter #1
  {
    \std@definecounter {#1}
    \int_new:c { g__talk_saved_ #1 _int }
    \seq_gput_right:Nn \l__talk_cnt_reset_seq {#1}
  }
```

(*End of definition for* `\@definecounter` *and* `\std@definecounter`*. These functions are documented on page* ??.)

1.3 Frame options

\l__talk_frame_alignment_tl

```
\tl_new:N \l__talk_frame_alignment_tl
```

(*End of definition for* `\l__talk_frame_alignment_tl`.)

\l__talk_action_spec_str
\l__talk_frame_tagging_str

```
\keys_define:nn { talk / frame }
  {
    action-spec .str_set:N
      = \l__talk_action_spec_str ,
    tag-slides .str_set:N
      = \l__talk_frame_tagging_str ,
    vertical-alignment .choices:nn =
      { bottom , center , stretch , top }
      {
        \tl_set_eq:NN \l__talk_frame_alignment_tl
          \l_keys_value_tl
      }
  }
\keys_set:nn { talk / frame }
  {
    action-spec        =        ,
    tag-slides         = n      ,
    vertical-alignment = center
  }
```

(*End of definition for* `\l__talk_action_spec_str` *and* `\l__talk_frame_tagging_str`.)

1.4 Tagging for headers

__talk_header_tag_begin:n
__talk_header_tag_begin:e
__talk_header_tag_end:

Generalized control for inserting material into the header area (which is otherwise outside of tagging).

```
\cs_new_protected:Npn \__talk_header_tag_begin:n #1
  {
    \tag_resume:n { header }
    \tag_mc_end:
    \tag_struct_begin:n {#1}
    \tag_mc_begin:n { }
  }
\cs_generate_variant:Nn \__talk_header_tag_begin:n { e }
```

```
\cs_new_protected:Npn \__talk_header_tag_end:
  {
    \tag_mc_end:
    \tag_struct_end:
    \tag_mc_begin:n { artifact }
    \tag_suspend:n { header }
  }
```

(*End of definition for* `\__talk_header_tag_begin:n` *and* `\__talk_header_tag_end:`.)

1.5 Wallpaper

`\l__talk_footelem_left_skip`
`\l__talk_footelem_right_skip`
`\l__talk_footelem_color_tl`
`\l__talk_footelem_font_tl`

```
\NewTemplateType { footer-element } { 1 }
\DeclareTemplateInterface { footer-element } { talk } { 1 }
  {
    color      : tokenlist ,
    font       : tokenlist = ,
    left-skip  : length = 0em ,
    right-skip : length = 0em
  }
\DeclareTemplateCode { footer-element } { talk } { 1 }
  {
    color      = \l__talk_footelem_color_tl ,
    font       = \l__talk_footelem_font_tl ,
    left-skip  = \l__talk_footelem_left_skip ,
    right-skip = \l__talk_footelem_right_skip
  }
  {
    \tl_if_empty:nF {#1}
      {
        \hspace { \l__talk_footelem_left_skip }
        \group_begin:
          \tl_if_empty:NF \l__talk_footelem_color_tl
            { \color_select:V \l__talk_footelem_color_tl }
          \l__talk_footelem_font_tl
          #1
        \group_end:
        \hspace { \l__talk_footelem_right_skip }
      }
  }
\DeclareInstance { footer-element } { date } { talk } { }
\DeclareInstance { footer-element } { author } { talk } { }
\DeclareInstance { footer-element } { title } { talk } { }
\DeclareInstance { footer-element } { institute } { talk } { }
\DeclareInstance { footer-element } { framenumber } { talk } { }
```

(*End of definition for* `\l__talk_footelem_left_skip` *and others.*)

`\l__talk_header_bg_tl`
`\l__talk_header_fg_tl`
`\l__talk_header_font_tl`
`\l__talk_header_ht_dim`
`\l__talk_header_left_skip`
`\l__talk_header_frametitle_bool`
`\l__talk_header_right_skip`

Templates for the header area. The background always covers the full width, but the text area may be narrower. The setup here aims to avoid repeated kerns but also dealing with complex conditionals, hence we always move to the edge of the paper first then adjust as required.

```
\NewTemplateType { header } { 0 }
```

```
\DeclareTemplateInterface { header } { talk } { 0 }
  {
    background-color  : tokenlist,
    color             : tokenlist = structure ,
    font              : tokenlist = \normalfont ,
    height            : length = \Gm@tmargin + \headsep ,
    left-hspace       : skip = \Gm@lmargin ,
    print-frame-title : boolean = true ,
    right-hspace      : skip = \Gm@rmargin
  }
\DeclareTemplateCode { header } { talk } { 0 }
  {
    background-color  = \l__talk_header_bg_tl ,
    color             = \l__talk_header_fg_tl ,
    font              = \l__talk_header_font_tl ,
    height            = \l__talk_header_ht_dim ,
    left-hspace       = \l__talk_header_left_skip ,
    print-frame-title = \l__talk_header_frametitle_bool ,
    right-hspace      = \l__talk_header_right_skip
  }
  {
    \noindent
    \__talk_wallpaper_hrule:Nnn
      \l__talk_header_bg_tl
      { \l__talk_header_ht_dim - \headsep }
      { \headsep }
    \skip_horizontal:n { \l__talk_header_left_skip }
    \group_begin:
      \tl_if_empty:NF \l__talk_header_fg_tl
        { \color_select:V \l__talk_header_fg_tl }
      \l__talk_header_font_tl
      \bool_if:NT \l__talk_header_frametitle_bool
        {
          \ExpandArgs { nnV }
            \UseInstance { frametitle } { header }
              \g__talk_frame_title_tl
        }
    \group_end:
  }
\DeclareInstance { header } { std } { talk } { }
\AddToHook { begindocument }
  {
    \DeclareInstanceCopy { header } { wallpaper } { std }
    \EditInstance { header } { wallpaper }
      { print-frame-title = false }
  }
```

(*End of definition for* `\l__talk_header_bg_tl` *and others.*)

`\l__talk_footer_bg_tl`
`\l__talk_footer_fg_tl`
`\l__talk_footer_font_tl`
`\l__talk_footer_order_clist`
`\l__talk_footer_sep_tl`
`\l__talk_footer_left_skip`
`\l__talk_footer_right_skip`

Templates for the footer area. Again the margins are handled in stages: here we do have a box for the content so the right skip is used, and we avoid an overfull box by including consideration of the right margin of the page layout.

```
\NewTemplateType { footer } { 0 }
\DeclareTemplateInterface { footer } { talk } { 0 }
```

```
  {
    background-color : tokenlist ,
    color            : tokenlist ,
    element-order    : commalist ,
    font             : tokenlist = \tiny ,
    left-skip        : length = \Gm@lmargin ,
    right-skip       : length = \Gm@rmargin ,
    separator        : tokenlist = \hfil
  }
\DeclareTemplateCode { footer } { talk } { 0 }
  {
    background-color = \l__talk_footer_bg_tl ,
    color            = \l__talk_footer_fg_tl ,
    element-order    = \l__talk_footer_order_clist ,
    separator        = \l__talk_footer_sep_tl ,
    font             = \l__talk_footer_font_tl ,
    left-skip        = \l__talk_footer_left_skip ,
    right-skip       = \l__talk_footer_right_skip
  }
  {
    \noindent
    \__talk_wallpaper_hrule:Nnn
      \l__talk_footer_bg_tl
      { \footskip }
      { \Gm@bmargin - \footskip }
    \skip_horizontal:n { \l__talk_footer_left_skip }
    \vbox_set_to_wd:Nnn \l__talk_tmp_box
      {
          \paperwidth
        - \l__talk_footer_left_skip
        - \l__talk_footer_right_skip
      }
      {
        \tl_if_empty:NF \l__talk_footer_fg_tl
          { \color_select:V \l__talk_footer_fg_tl }
        \l__talk_footer_font_tl
        \clist_pop:NNT \l__talk_footer_order_clist \l__talk_tmp_tl
          {
            \ExpandArgs { nVv } \UseInstance { footer-element } \l__talk_tmp_tl
              { @ \l__talk_tmp_tl }
            \clist_map_inline:Nn \l__talk_footer_order_clist
              {
                \tl_if_empty:cF { @ ##1 }
                  {
                    \l__talk_footer_sep_tl
                    \ExpandArgs { nnv }
                      \UseInstance { footer-element } {##1} { @ ##1 }
                  }
              }
          }
        \hfil
      }
    \box_use_drop:N \l__talk_tmp_box
    \skip_horizontal:n { \l__talk_footer_right_skip - \Gm@rmargin }
```

```
  }
\DeclareInstance { footer } { std } { talk } { }
\AddToHook { begindocument }
  {
    \DeclareInstanceCopy { footer } { wallpaper } { std }
    \EditInstance { footer } { wallpaper }
      { element-order = }
  }
```

(*End of definition for* `\l__talk_footer_bg_tl` *and others.*)

`\__talk_wallpaper_hrule:Nnn` A simple abstraction for the top and bottom rules on the page.

```
\cs_new_protected:Npn \__talk_wallpaper_hrule:Nnn #1#2#3
  {
    \skip_horizontal:n { -\Gm@lmargin }
    \tl_if_empty:NF #1
      {
        \group_begin:
          \color_select:V #1
          \rule:nnn { \paperwidth } {#2} {#3}
          \skip_horizontal:n { -\paperwidth }
        \group_end:
      }
  }
```

(*End of definition for* `\__talk_wallpaper_hrule:Nnn`.)

`\ps@plain` `\ps@wallpaper` `\ps@talk` Install a standard header and footer template, and redefine the `plain` one as this will be used for frames without "wallpaper" which still need core links, *etc.* We also provide a version that only shows the visual elements: this is deliberately using the same settings as the main templates.

```
\cs_set_nopar:Npn \ps@plain
  {
    \cs_set_nopar:Npn \@oddhead
      {
        \__talk_section_tagged:
        \hfil
      }
    \cs_set_nopar:Npn \@oddfoot { }
    \cs_set_eq:NN \@evenhead \@oddhead
    \cs_set_eq:NN \@evenfoot \@oddfoot
  }
\cs_set_nopar:Npn \ps@wallpaper
  {
    \cs_set_nopar:Npn \@oddhead
      {
        \__talk_section_tagged:
        \UseInstance { header } { wallpaper }
        \hfil
      }
    \cs_set_nopar:Npn \@oddfoot
      {
        \UseInstance { footer } { wallpaper }
        \hfil
```

```
      }
    \cs_set_eq:NN \@evenhead \@oddhead
    \cs_set_eq:NN \@evenfoot \@oddfoot
  }
\cs_new_nopar:Npn \ps@talk
  {
    \cs_set_nopar:Npn \@oddhead
      {
        \__talk_section_tagged:
        \UseInstance { header } { std }
        \hfil
      }
    \cs_set_nopar:Npn \@oddfoot { \UseInstance { footer } { std } }
    \cs_set_eq:NN \@evenhead \@oddhead
    \cs_set_eq:NN \@evenfoot \@oddfoot
  }
\pagestyle { talk }
```

(*End of definition for* `\ps@plain`, `\ps@wallpaper`, *and* `\ps@talk`. *These functions are documented on page* **??**.)

1.6 The `frame` environment

`\l__talk_frame_bool` To track whether we are inside a frame or not.

```
\bool_new:N \l__talk_frame_bool
```

(*End of definition for* `\l__talk_frame_bool`.)

`\g__talk_frame_tag_bool` To track when a frame is being tagged: mainly needed for the header (and as a result global).

```
\bool_new:N \g__talk_frame_tag_bool
```

(*End of definition for* `\g__talk_frame_tag_bool`.)

`\l__talk_frame_verb_bool` Indicates that material was collected verbatim (and thus needs rescanning).

```
\bool_new:N \l__talk_frame_verb_bool
```

(*End of definition for* `\l__talk_frame_verb_bool`.)

`\g__talk_frame_int` The overall frame number, including LaTeX counter-like access.

`\c@frame` `\theframe` `\@framenumber`

```
\int_new:N \g__talk_frame_int
\cs_new_eq:NN \c@frame \g__talk_frame_int
\cs_new:Npn \theframe { \@arabic \c@frame }
\cs_new:Npn \@framenumber { \arabic { frame } }
```

(*End of definition for* `\g__talk_frame_int` *and others. These variables are documented on page* **??**.)

The total frames can be handled using the kernel properties.

```
\property_new:nnnn { totalframes } { shipout } { -1 }
  { \int_use:N \g__talk_frame_int }
\AddToHook { enddocument / afterlastpage }
  { \property_record:nn { lastpage } { totalframes } }
```

`\__talk_latexe_frame:n` As we will need to re-define `\frame` but have it available inside frames, a copy is made here.

```
\NewCommandCopy \__talk_latexe_frame:n \frame
```

(*End of definition for* `\__talk_latexe_frame:n`.)

`\__talk_frame_process:nn` Here, the frame content is received as the argument.

```
\cs_new_protected:Npn \__talk_frame_process:nn #1#2
  {
    \int_gincr:N \g__talk_frame_int
    \bool_set_true:N \l__talk_frame_bool
    \__talk_slide:nn {#1} {#2}
  }
```

(*End of definition for* `\__talk_frame_process:nn`.)

`\__talk_frame_tag:n` Wraps some content in tagging for a frame: we may have multiple of these in one logical frame, but that is non-standard.

```
\cs_new_protected:Npn \__talk_frame_tag:n #1
  {
    \tag_struct_begin:n { tag = frame }
    \int_gset:Nn \g__talk_frame_struct_int { \tag_get:n { struct_num } }
    \bool_gset_true:N \g__talk_frame_tag_bool
    #1
    \tag_struct_end:
  }
```

(*End of definition for* `\__talk_frame_tag:n`.)

`\__talk_frame_notag:n` The alternative: turn off tagging and suppress the function that would tag the frame title.

```
\cs_new_protected:Npn \__talk_frame_notag:n #1
  {
    \tag_mc_begin:n { artifact }
    \tag_suspend:n { frame }
    \bool_gset_false:N \g__talk_frame_tag_bool
    #1
    \par
    \tag_resume:n { frame }
    \tag_mc_end:
  }
```

(*End of definition for* `\__talk_frame_notag:n`.)

`frame` `frame*` The definition for the `frame` and `frame*` environments: the exact interface at both the document and code levels is still open.

```
\bool_if:NTF \l__talk_frame_title_bool
  {
    \RenewDocumentEnvironment { frame }
      { D <> { all } = { action-spec } O { } +m +b }
      {
        \keys_set:nn { talk / frame } {#2}
        \bool_set_false:N \l__talk_frame_verb_bool
        \__talk_frame_process:nn {#1} { \frametitle {#3} #4 }
      }
      { }
    \NewDocumentEnvironment { frame* }
      { D <> { all } = { action-spec } O { } +m c }
```

```
      {
        \keys_set:nn { talk / frame } {#2}
        \bool_set_true:N \l__talk_frame_verb_bool
        \tl_gset:Nn \g__talk_frame_title_tl {#3}
        \exp_args:Nne \__talk_frame_process:nn {#1}
          { \tl_to_str:n { \frametitle } \exp_not:n { {#3} #4 } }
      }
      { }
  }
  {
    \RenewDocumentEnvironment { frame }
      { !D <> { all } = { action-spec } !O { } +b }
      {
        \keys_set:nn { talk / frame } {#2}
        \bool_set_false:N \l__talk_frame_verb_bool
        \__talk_frame_process:nn {#1} {#3}
      }
      { }
    \NewDocumentEnvironment { frame* }
      { !D <> { all } = { action-spec } !O { } c }
      {
        \keys_set:nn { talk / frame } {#2}
        \bool_set_true:N \l__talk_frame_verb_bool
        \__talk_frame_process:nn {#1} {#3}
      }
      { }
  }
```

(*End of definition for* `frame` *and* `frame*`. *These functions are documented on page* **??**.)

```
⟨/class⟩
```

Part V
ltx-talk-frame – The structure of frames

1 ltx-talk-frame-structure implementation

Start the DocStrip guards.

```
⟨*class⟩
```

Identify the internal prefix.

```
⟨@@=talk⟩
```

1.1 Columns

```
\keys_define:nn { talk }
  { columns .inherit:n = talk / column }
```

`\l__talk_columns_wd_tl` We store the requested width for columns in a `tl` as this means that the key value will make sense even if it depends on the current `\textwidth`.

```
\keys_define:nn { talk / columns }
  { width .tl_set:N = \l__talk_columns_wd_tl }
\keys_set:nn { talk / columns }
  { width = \textwidth }
```

(*End of definition for* `\l__talk_columns_wd_tl`.)

`columns` (*env.*) Columns are block-like environments so we start and end with a `\par` to ensure correct tagging.

```
\NewDocumentEnvironment { columns } { D <> { all } O { } }
  {
    \__talk_action_begin:n {#1}
    \par
    \keys_set:nn { talk / columns } {#2}
    \hbox_set_to_wd:Nnw \l__talk_tmp_box { \l__talk_columns_wd_tl }
      \dim_set:Nn \textwidth { \l__talk_columns_wd_tl }
      \dim_set_eq:NN \columnwidth \textwidth
      \hfil
      \ignorespaces
  }
  {
      \unskip
      \hfil
    \hbox_set_end:
    \box_use_drop:N \l__talk_tmp_box
    \par
    \__talk_action_end:
  }
```

\l__talk_column_alignment_tl

```
\keys_define:nn { talk / column }
  {
    b .meta:n =
      { vertical-alignment = bottom } ,
    b .value_forbidden:n = true ,
    c .meta:n =
      { vertical-alignment = center } ,
    c .value_forbidden:n = true ,
    t .meta:n =
      { vertical-alignment = top } ,
    t .value_forbidden:n = true ,
    vertical-alignment .choices:nn =
      { bottom , center , top }
      {
        \tl_set_eq:NN \l__talk_column_alignment_tl
          \l_keys_value_tl
      }
  }
\keys_set:nn { talk / column }
  {
    vertical-alignment = center
  }
```

(*End of definition for* `\l__talk_column_alignment_tl`.)

__talk_column_align_bottom:n
__talk_column_align_center:n
__talk_column_align_top:n

Based on ideas in the highly experimental xbox.

```
\cs_new_protected:Npn \__talk_column_align_bottom:n #1
  { \vbox:n {#1} }
\cs_new_protected:Npn \__talk_column_align_center:n #1
  {
    \vbox:n
      {
        \hbox:n
          {
            \box_move_down:nn
              {
                  0.5 \box_ht:N \l__talk_tmp_box
                - \tex_fontdimen:D 22 ~ \tex_textfont:D 2 ~
              }
              { \vbox:n {#1} }
          }
      }
  }
\cs_new_protected:Npn \__talk_column_align_top:n #1
  { \vbox_top:n {#1} }
```

(*End of definition for* `\__talk_column_align_bottom:n`, `\__talk_column_align_center:n`, *and* `\__talk_column_align_top:n`.)

`column` (*env.*) A cut-down version of a minipage: we want to be clear on the semantic meaning. the action is applied inside the box after starting horizontal mode to avoid spacing issues when switching whatsits in and out.

```
\NewDocumentEnvironment { column } { D <> { all } O { } m }
```

```
  {
    \par
    \keys_set:nn { talk / column } {#2}
    \vbox_set_to_wd:Nnw \l__talk_tmp_box {#3}
      \dim_set:Nn \textwidth {#3}
      \dim_set_eq:NN \columnwidth \textwidth
      \@parboxrestore
      \leavevmode
      \raggedright
      \__talk_action_begin:n {#1}
      \ignorespaces
  }
```

The `\@ignore` here means that any spaces after `\end{column}` are suppressed by a `\ignorespaces` inserted by the kernel.

```
  {
      \__talk_action_end:
    \vbox_set_end:
    \use:c { __talk_column_align_ \l__talk_column_alignment_tl :n }
      { \vbox_unpack_drop:N \l__talk_tmp_box }
    \hfil
    \par
    \@ignoretrue
  }
```

1.2 Floats

Well really "not floats at all" but the idea is clear.

`\l__talk_float_alignment_tl`

We only worry about horizontal alignment here.

```
\tl_new:N  \l__talk_float_alignment_tl
```

(*End of definition for* `\l__talk_float_alignment_tl`.)

A bit similar to the current approach to lists: we need a template at the start but a common function at the end. The `float-placement` key is at present just there to allow mopping up of any argument that is given by accident, hence maps to a temporary variable.

```
\NewTemplateType { floatenv } { 2 }
\DeclareTemplateInterface { floatenv } { talk } { 2 }
  {
    float-placement : tokenlist ,
    horizontal-alignment : choice { left , center ,  right } = left
  }
\DeclareTemplateCode { floatenv } { talk } { 2 }
  {
    float-placement = \l__talk_tmp_tl ,
    horizontal-alignment =
      {
        left = \tl_set:Nn \l__talk_float_alignment_tl { flushleft } ,
        center = \tl_set:Nn \l__talk_float_alignment_tl { center } ,
        right = \tl_set:Nn \l__talk_float_alignment_tl { flushright }
      }
  }
  {
```

```
    \SetTemplateKeys { floatenv } { talk } {#1}
    \begin { minipage } { \columnwidth }
      \begin { \l__talk_float_alignment_tl }
        \cs_set_nopar:Npn \@captype {#2}
  }
\DeclareInstance { floatenv } { std } { talk } { horizontal-alignment = left }
```

`\endfloatenv` And the common end function.

```
\cs_new_protected:Npn \endfloatenv
  {
      \end { \l__talk_float_alignment_tl }
    \end { minipage }
  }
```

(*End of definition for* `\endfloatenv`*. This function is documented on page* ??.)

`figure` (*env.*) `table` (*env.*) Unlike beamer, we allow for overlays for the environments as a whole.

```
\clist_map_inline:nn { figure , table }
  {
    \NewDocumentEnvironment {#1} { D <> { all } = { float-placement } O { } }
      {
        \__talk_action_begin:n {##1}
        \UseInstance { floatenv } { std } {##2} {#1}
      }
      {
        \endfloatenv
        \__talk_action_end:
      }
```

`\c@figure` `\thefigure` `\c@table` `\thetable` `\figurename` `\tableename` `\fnum@figure` `\fnum@table` The standard variables needed to make captions work (nothing for list of floats, as at present those are not offered).

```
    \newcounter {#1}
    \tl_new:c { #1 name }
    \tl_set:ce { #1 name } { \text_titlecase_first:n {#1} }
    \tl_new:c { fnum@ #1 }
    \tl_set:ce { fnum@ #1 }
      { \exp_not:c { #1 name } \exp_not:N \nobreakspace \exp_not:c { the #1 } }
  }
```

(*End of definition for* `\c@figure` *and others. These variables are documented on page* ??.)

The spacing values needed for the standard function.

```
\newlength \abovecaptionskip
\newlength \belowcaptionskip
\setlength \abovecaptionskip { 7pt }
\setlength \belowcaptionskip { 7pt }
```

`\@caption` This is a copy of the kernel version of the function, but with writing to the list of whatever file removed. It is very likely this needs to be reworked as a template, but that will likely come from the kernel.

```
\cs_set_protected:Npn \@caption #1 [ #2 ] #3
  {
    \par
    \begingroup
      \@parboxrestore
```

```
      \if@minipage \@setminipage \fi
      \normalsize
      \@makecaption { \csname fnum@ #1 \endcsname } { \ignorespaces #3 }
      \par
    \endgroup
  }
```

(*End of definition for* `\@caption`. *This function is documented on page* **??**.)

⟨/class⟩

Part VI
ltx-talk-mode – Modes

1 ltx-talk-mode implementation

Start the DocStrip guards.

```
1 ⟨*class⟩
```

Identify the internal prefix.

```
2 ⟨@@=talk⟩
```

`\__talk_mode:nT` A simplified version of `\mode`: only deal with the argument form, only check the entire overlay spec as a string.

```
3 \prg_new_protected_conditional:Npnn \__talk_mode:n #1 { T }
4   {
5     \bool_lazy_or:nnTF
6       { \str_if_eq_p:nn {#1} { all } }
7       { \str_if_eq_p:Vn \l__talk_mode_str {#1} }
8       \prg_return_true:
9       \prg_return_false:
10  }
```

(*End of definition for* `\__talk_mode:nT`.)

`\mode`

```
11 \NewDocumentCommand \mode { D <> { all } +m }
12   { \__talk_mode:nT {#1} {#2} }
```

(*End of definition for* `\mode`. *This function is documented on page* **??**.)

```
13 ⟨/class⟩
```

Part VII
ltx-talk-overlay – Overlays

1 ltx-talk-overlay implementation

Start the DocStrip guards.

```
1 ⟨*class⟩
```

Identify the internal prefix.

```
2 ⟨@@=talk⟩
```

1.1 Utilities

__talk_if_overlay:nTF
__talk_if_overlay:VTF
__talk_overlay_arg:n

```
3 \prg_new_protected_conditional:Npnn \__talk_if_overlay:n #1 { T , F , TF }
4   {
5     \__talk_decode_parse:n {#1}
6     \bool_if:NTF \l__talk_decode_overlays_bool
7       \prg_return_true:
8       \prg_return_false:
9   }
10 \prg_generate_conditional_variant:Nnn \__talk_if_overlay:n { V } { T , F , TF }
```

A macro processor variant of the check that always results in an N-type bool.

```
11 \cs_new_protected:Npn \__talk_overlay_arg:n #1
12   {
13     \__talk_if_overlay:nTF {#1}
14       { \cs_set:Npn \ProcessedArgument { \c_true_bool } }
15       { \cs_set:Npn \ProcessedArgument { \c_false_bool } }
16   }
```

(*End of definition for* `\__talk_if_overlay:nTF` *and* `\__talk_overlay_arg:n`.)

1.2 Action commands and environments

Commands that can be used as actions all have a common form (with one exception). The common internal structure is used to enable them to be used as actions by looking for the name `\__talk_action_`⟨*name*⟩`:N`. This is set up such that the inactive versions insert a whatsit equal to that which would be present if they were active: that's needed for spacing.

__talk_action_:N

The fallback action. At present, we need to create a whatsit here to avoid spacing issues. (In LuaTEX, if we can move to attributes, this can be removed.)

```
17 \cs_new_protected:Npn \__talk_action_:N #1 { \opacity_select:n { 1 } }
```

(*End of definition for* `\__talk_action_:N`.)

__talk_action_alert:N

At present a color selection.

```
18 \cs_new_protected:Npn \__talk_action_alert:N #1
19   {
20     \bool_if:NTF #1
21       { \color_select:n { alert } }
```

```
      { \color_select:n { . } }
  }
```

(*End of definition for* `\__talk_action_alert:N`.)

`\__talk_action_invisible:N`
`\__talk_action_visible:N`

Simply hide unconditionally.

```
\cs_new_protected:Npn \__talk_action_invisible:N #1
  {
    \bool_if:NTF #1
      { \opacity_select:n { 0 } }
      { \opacity_select:n { 1 } }
  }
\cs_new_protected:Npn \__talk_action_visible:N #1
  {
    \bool_if:NTF #1
      { \opacity_select:n { 1 } }
      { \opacity_select:n { 0 } }
  }
```

(*End of definition for* `\__talk_action_invisible:N` *and* `\__talk_action_visible:N`.)

`\__talk_action_only_begin:N`
`\__talk_action_only_end:N`

Here, we simply throw away the content we do not want: this is done by typesetting in a disposable box.

```
\cs_new_protected:Npn \__talk_action_only:N #1
  {
    \bool_if:NF #1
      { \vbox_set:Nw \l__talk_tmp_box }
  }
\cs_new_protected:Npn \__talk_action_only_end:N #1
  {
    \bool_if:NF #1
      { \vbox_set_end: }
  }
```

(*End of definition for* `\__talk_action_only_begin:N` *and* `\__talk_action_only_end:N`.)

`\l__talk_uncover_hidden_fp`

Currently just an on-off, but that will change.

```
\NewTemplateType { hidden } { 0 }
\DeclareTemplateInterface { hidden } { talk } { 0 }
  { opacity : real = 0 }
\DeclareTemplateCode { hidden } { talk } { 0 }
  { opacity = \l__talk_uncover_hidden_fp }
  { \opacity_select:n { \l__talk_uncover_hidden_fp } }
\DeclareInstance { hidden } { std } { talk } { }
```

(*End of definition for* `\l__talk_uncover_hidden_fp`.)

`\__talk_action_uncover:N`

Use the template

```
\cs_new_protected:Npn \__talk_action_uncover:N #1
  {
    \bool_if:NTF #1
      { \opacity_select:n { 1 } }
      { \UseInstance { hidden } { std } }
  }
```

(*End of definition for* `\__talk_action_uncover:N`.)

`\only` `\invisible` `\uncover`

Commands and environments where the payload applies when the material is not active on the slide.

```
\clist_map_inline:nn { only , invisible , uncover }
  {
    \ExpandArgs { cne } \NewDocumentCommand {#1}
      { > { \__talk_overlay_arg:n } D <> { all } +m }
      {
        \group_begin:
          \exp_not:c { __talk_action_ #1 :N } ##1
          ##2
          \cs_if_exist:cT { __talk_action_ #1 _end:N }
            { \exp_not:c { __talk_action_ #1 _end:N } ##1 }
        \group_end:
      }
```

(*End of definition for* `\only`, `\invisible`, *and* `\uncover`. *These functions are documented on page* **??**.)

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

```
    \ExpandArgs { nnee } \NewDocumentEnvironment { #1 env }
      { > { \__talk_overlay_arg:n } D <> { all } }
      { \exp_not:c { __talk_action_ #1 :N } ##1 }
      {
        \cs_if_exist:cT { __talk_action_ #1 _end:N }
          { \exp_not:c { __talk_action_ #1 _end:N } ##1 }
      }
  }
```

`\alert` `\visible`

And those where the action applies when we are on the slide.

```
\clist_map_inline:nn { alert , visible }
  {
    \ExpandArgs { cne } \NewDocumentCommand {#1}
      { > { \__talk_overlay_arg:n } D <> { all } +m }
      {
        \group_begin:
          \exp_not:c { __talk_action_ #1 :N } ##1
            ##2
         \group_end:
      }
```

(*End of definition for* `\alert` *and* `\visible`. *These functions are documented on page* **??**.)

`alertenv` (*env.*) `visibleenv` (*env.*)

```
    \ExpandArgs { nnee } \NewDocumentEnvironment { #1 env }
      { > { \__talk_overlay_arg:n } D <> { all } }
      { \exp_not:c { __talk_action_ #1 :N } ##1 }
      { }
  }
```

`\only` This code needs to be done manually as for the command version the content must be entirely discarded. That can't work for the environment version, which has to deal with for example single items in a list (and so cannot be collected up verbatim and must use a box).

```
\RenewDocumentCommand \only { D <> { all } +m }
  {
    \__talk_if_overlay:nT {#1}
      {#2}
  }
```

(*End of definition for* `\only`. *This function is documented on page* **??**.)

`\l__talk_saved_overlays_bool`
`\l__talk_saved_action_str`
`\l__talk_saved_actions_bool`

```
\bool_new:N \l__talk_saved_overlays_bool
\str_new:N \l__talk_saved_action_str
\bool_new:N \l__talk_saved_actions_bool
```

(*End of definition for* `\l__talk_saved_overlays_bool`, `\l__talk_saved_action_str`, *and* `\l__talk_saved_actions_bool`.)

`actionenv` `\action` `\__talk_action_begin:n` `\__talk_action_end:`

As we need data on not just overlays but also actions at the end of the environment, this has to be done manually. To allow working with environments but also items, the code needs to save data for the end function. The group is needed for cases where we are not in a LaTeX environment group.

```
\NewDocumentCommand \action { D <> { all } +m }
  {
    \group_begin:
      \__talk_action_begin:n {#1}
      #2
      \__talk_action_end:
    \group_end:
  }
\NewDocumentEnvironment { actionenv } { D <> { all } }
  { \__talk_action_begin:n {#1} }
  { \__talk_action_end: }
\cs_new_protected:Npn \__talk_action_begin:n #1
  {
    \group_begin:
      \__talk_decode_parse:n {#1}
      \bool_set_eq:NN \l__talk_saved_overlays_bool
        \l__talk_decode_overlays_bool
      \str_set_eq:NN \l__talk_saved_action_str
        \l__talk_decode_action_str
      \bool_set_eq:NN \l__talk_saved_actions_bool
        \l__talk_decode_actions_bool
      \bool_if:NTF \l__talk_decode_overlays_bool
        {
          \use:c { __talk_action_ \l__talk_decode_action_str :N }
            \l__talk_decode_actions_bool
        }
        { \UseInstance { hidden } { std } }
  }
\cs_new_protected:Npn \__talk_action_end:
  {
      \bool_if:NT \l__talk_saved_overlays_bool
        {
          \cs_if_exist_use:cF
            { __talk_action_ \l__talk_saved_action_str _end:N }
```

```
            { \use_none:n }
              \l__talk_saved_actions_bool
        }
    \group_end:
  }
```

(*End of definition for* `\action`, `\__talk_action_begin:n`, *and* `\__talk_action_end:`. *This function is documented on page* ??.)

1.3 Non-action commands and environments

This section contains commands and environments that do *not* need to be made available as actions.

`\alt` Simple wrappers around the internal switch.

```
\NewDocumentCommand \alt { D <> { all } +m +m }
  {
    \__talk_if_overlay:nTF {#1}
      {#2}
      {#3}
  }
```

(*End of definition for* `\alt`. *This function is documented on page* ??.)

`\onslide` `\__talk_onslide:n` `\__talk_onslide_reset:` Simply make transparent: we will likely need to save the original opacity level. To allow us to apply independent of group level, a little work is needed.

```
\NewDocumentCommand \onslide { D <> { all } }
  { \__talk_onslide:n {#1} }
\cs_new_protected:Npn \__talk_onslide:n #1
  {
    \tl_use:N \g__talk_onslide_tl
    \__talk_if_overlay:nTF {#1}
      { \__talk_onslide_reset: }
      {
        \opacity_select:n { 0 }
        \tl_gset:Nn \g__talk_onslide_escape_tl
          {
            \opacity_select:n { 0 }
            \group_insert_after:N \g__talk_onslide_escape_tl
          }
        \group_insert_after:N \g__talk_onslide_escape_tl
        \tl_gset:Nn \g__talk_onslide_tl
          {
            \tl_gclear:N \g__talk_onslide_tl
            \tl_gclear:N \g__talk_onslide_escape_tl
            \__talk_onslide_reset:
          }
      }
  }
\cs_new_protected:Npn \__talk_onslide_reset: { \opacity_select:n { 1 } }
```

(*End of definition for* `\onslide`, `\__talk_onslide:n`, *and* `\__talk_onslide_reset:`. *This function is documented on page* ??.)

\g__talk_onslide_tl
\g__talk_onslide_escape_tl

```
\tl_new:N \g__talk_onslide_tl
\tl_new:N \g__talk_onslide_escape_tl
```

(*End of definition for* `\g__talk_onslide_tl` *and* `\g__talk_onslide_escape_tl`.)

`\temporal` A tricky one: to separate the not-on-current-slide cases, the flag to continue is used.

```
\NewDocumentCommand \temporal { D <> { all } +m +m +m }
  {
    \__talk_if_overlay:nTF {#1}
      {#3}
      {
        \bool_if:NTF \g__talk_slide_continue_bool
          {#4}
          {#2}
      }
  }
```

(*End of definition for* `\temporal`. *This function is documented on page* **??**.)

`\pause` A thin wrapper.

```
\NewDocumentCommand \pause { o }
  {
    \IfNoValueTF {#1}
      { \int_gincr:N \g__talk_pauses_int }
      { \int_gset:Nn \g__talk_pauses_int {#1} }
    \exp_args:Ne \__talk_onslide:n { \int_eval:n { \g__talk_pauses_int + 1 } - }
  }
```

(*End of definition for* `\pause`. *This function is documented on page* **??**.)

1.4 Fixed-size areas

`\__talk_overprint_begin:n` A common auxiliary for overprinting, which starts off much the same for both `overlayarea` and `overprint`.

```
\cs_new_protected:Npn \__talk_overprint_begin:n #1
  {
    \par
    \vbox_set_to_wd:Nnw \l__talk_tmp_box {#1}
      \raggedright
      \ignorespaces
  }
```

(*End of definition for* `\__talk_overprint_begin:n`.)

`overlayarea` (*env.*) An initial approach: quite similar to a column.

```
\NewDocumentEnvironment { overlayarea } { m m }
  { \__talk_overprint_begin:n {#1} }
  {
    \vbox_set_end:
    \vbox_to_ht:nn {#2}
      {
        \box_use_drop:N \l__talk_tmp_box
        \vfil
```

```
      }
    \par
  }
```

`\l__talk_overprint_int` Track the overprints on a slide: as the slide forms a group, we do not need to worry about resetting.

```
\int_new:N \l__talk_overprint_int
```

(*End of definition for* `\l__talk_overprint_int`.)

`\__talk_frame_overprint:` To refer to the current overprint environment within the document: needed in the `.aux` so avoids using non-letters.

```
\cs_new:Npn \__talk_frame_overprint:
  {
    \int_to_Roman:n \g__talk_frame_int
    \int_to_roman:n \l__talk_overprint_int
  }
```

(*End of definition for* `\__talk_frame_overprint:`.)

`\__talk_overprint_begin:n` (*env.*) `\__talk_overprint_check_ht:n` For overprinting, in contrast to beamer we use a two-pass approach to save the size at the end of the run: this means you can use `\only` for example in overprinting.

```
\NewDocumentEnvironment { overprint } { O { \textwidth } }
  { \__talk_overprint_begin:n {#1} }
  {
    \vbox_set_end:
    \int_incr:N \l__talk_overprint_int
    \__talk_overprint_save_ht:
    \cs_if_exist:cTF
      { overprint@ \__talk_frame_overprint: }
      {
        \dim_compare:vNnTF
          { overprint@ \__talk_frame_overprint: }
            > { \box_ht:N \l__talk_tmp_box }
          {
            \vbox_to_ht:vn
              { overprint@ \__talk_frame_overprint: }
              {
                \box_use_drop:N \l__talk_tmp_box
                \vfil
              }
          }
          { \box_use_drop:N \l__talk_tmp_box }
      }
      { \box_use_drop:N \l__talk_tmp_box }
    \par
  }
```

As there is no clear end-point for overprinting, we need to be careful to keep the current width separate from the saved one. The rest is then about saving to the `.aux` file and helping out the user.

```
\cs_new_protected:Npn \__talk_overprint_save_ht:
  {
    \tl_if_exist:cF { g__talk_overprint_ \__talk_frame_overprint: _tl }
```

```
      {
        \tl_new:c { g__talk_overprint_ \__talk_frame_overprint: _tl }
        \tl_gset:cn { g__talk_overprint_ \__talk_frame_overprint: _tl }
          { 0pt }
      }
    \tl_gset:ce { g__talk_overprint_ \__talk_frame_overprint: _tl }
      {
        \dim_max:vn { g__talk_overprint_ \__talk_frame_overprint: _tl }
          { \box_ht:N \l__talk_tmp_box }
      }
    \legacy_if:nT { @filesw }
      {
        \iow_now:Ne \@auxout
          {
            \gdef \exp_not:c { overprint@ \__talk_frame_overprint: }
              {
                \exp_not:v { g__talk_overprint_ \__talk_frame_overprint: _tl }
              }
          }
      }
    \hook_gput_code:nne { enddocument / afterlastpage } { talk }
       { \__talk_overprint_check_ht:n { \__talk_frame_overprint: } }
  }
\cs_new_protected:Npn \__talk_overprint_check_ht:n #1
  {
    \bool_lazy_and:nnF
      { \exp_not:N \cs_if_exist_p:c { overprint@ #1 } }
      {
        \dim_compare_p:vNv { overprint@ #1 } = { g__talk_overprint_ #1 _tl }
      }
      {
        \msg_warning:nn { talk } { overprint-ht }
        \cs_gset_protected:Npn \__talk_overprint_check_ht:n ##1 { }
      }
  }
\msg_new:nnn { talk } { overprint-ht }
  {
    Overprint~area~height~has~changed:\\
    rerun~LaTeX.
  }
```

(*End of definition for* `\__talk_overprint_save_ht:` *and* `\__talk_overprint_check_ht:n`.)

1.5 Adding overlays to existing commands

`\textbf` `\textit` `\textmd` `\textnormal` `\textrm` `\textsc` `\textsf` `\textsl` `\texttt` `\textup` `\emph` `\stdtextbf` `\stdtextit` `\stdtextmd` `\stdtextnormal` `\stdtextrm` `\stdtextsc` `\stdtextsf` `\stdtextsl`

Make the standard text commands overlay-aware. To keep the spacing unchanged when the command is not active, we use the same approach as the kernel does for inserting the right grouping.

```
\tl_map_inline:nn
  {
    \textbf
    \textit
    \textmd
    \textnormal
```

```
    \textrm
    \textsc
    \textsf
    \textsl
    \texttt
    \textup
    \emph
  }
  {
    \ExpandArgs { c } \NewCommandCopy { std \cs_to_str:N #1 } #1
    \ExpandArgs { Nne } \RenewDocumentCommand #1
      { D <> { all } +m }
      {
        \exp_not:N \__talk_if_overlay:nTF {##1}
          { \exp_not:c { std \cs_to_str:N #1 } }
          { \exp_not:N \__talk_textcmd_eqiv:n }
            {##2}
      }
  }
\cs_new_protected:Npn \__talk_textcmd_eqiv:n #1
  {
    \mode_if_math:TF
      { { \mbox {#1} } }
      {
        \mode_leave_vertical:
        {#1}
      }
  }
```

(*End of definition for* `\textbf` *and others. These functions are documented on page* **??**.)

`\includegraphics` `\stdincludegraphics`

Just wrap up the args and forward if appropriate. The star is `#1` here as that matches the documented behavior of starred commands generally.

```
\RequirePackage { graphicx }
\NewCommandCopy \stdincludegraphics \includegraphics
\RenewDocumentCommand \includegraphics { s D <> { all } o o m }
  {
    \__talk_if_overlay:nT {#2}
      {
        \use:e
          {
            \exp_not:N \stdincludegraphics
              \IfBooleanT #1 { * }
              \IfNoValueF {#3} { [ \exp_not:n { {#3} } ] }
              \IfNoValueF {#4} { [ \exp_not:n { {#4} } ] }
          }
            {#5}
      }
  }
```

(*End of definition for* `\includegraphics` *and* `\stdincludegraphics`*. These functions are documented on page* **??**.)

`\label` `\__talk_label:n`

Here, we can't wrap the existing command up as we need the space hack, so it has to be declared from scratch. There is also a non-standard overlay default. At present, no

special tricks as seen in beamer.

```
\RenewDocumentCommand \label { D <> { 1 } m }
  {
    \@bsphack
    \__talk_if_overlay:nT {#1}
      { \__talk_label:n {#2} }
    \@esphack
  }
\cs_new_protected:Npn \__talk_label:n #1
  {
    \begingroup
      \UseHookWithArguments { label } { 1 } {#1}
      \protected@write \@auxout { }
        {
          \string \newlabel {#1}
            {
              { \@currentlabel }
              { \thepage }
              { \@currentlabelname }
              { \@currentHref }
              { \@kernel@reserved@label@data }
            }
        }
    \endgroup
  }
```

(*End of definition for* `\label` *and* `\__talk_label:n`*. This function is documented on page* **??**.)

⟨/class⟩

Part VIII
ltx-talk-required – "Required" definitions

1 ltx-talk-required implementation

Start the DocStrip guards.

```
1 ⟨*class⟩
```

Identify the internal prefix.

```
2 ⟨@@=talk⟩
```

Here we collect up things that are more-or-less required to create a useful class but are not defined by the LaTeX kernel for historical reasons. They are therefore largely copies from `article.cls` and contain "classical" definitions so that they follow the expectations of third-party code.

`\today` This is the definition as done in the standard classes.

```
3 \cs_new_nopar:Npn \today
4   {
5     \ifcase \month \or
6       January \or
7       February \or
8       March \or
9       April \or
10      May \or
11      June \or
12      July \or
13      August \or
14      September \or
15      October \or
16      November \or
17      December
18    \fi
19    \space
20    \number \day ,
21    \space
22    \number \year
23  }
```

(*End of definition for* `\today`*. This function is documented on page* ??.)

1.1 Standard design settings

```
24 \setcounter { tocdepth } { 3 }
25 \setlength \arraycolsep { 5pt }
26 \setlength \tabcolsep { 6pt }
27 \setlength \arrayrulewidth { 0.4pt }
28 \setlength \doublerulesep { 2pt }
29 \setlength \tabbingsep { \labelsep }
30 \skip \@mpfootins = \skip \footins
```

```
\setlength \fboxsep { 3pt }
\setlength \fboxrule { 0.4pt }
```

1.2 List support

```
\setlength  \labelsep  { 0.5em }
\cs_new:Npn \labelenumi { \theenumi . }
\cs_new:Npn \labelenumii { ( \theenumii ) }
\cs_new:Npn \labelenumiii { \theenumiii . }
\cs_new:Npn \labelenumiv { \theenumiv . }
\cs_new:Npn \labelitemi  { \labelitemfont \textbullet }
\cs_new:Npn \labelitemii { \labelitemfont \bfseries \textendash }
\cs_new:Npn \labelitemiii { \labelitemfont \textasteriskcentered }
\cs_new:Npn \labelitemiv  { \labelitemfont \textperiodcentered }
\cs_new:Npn \labelitemfont { \normalfont }
```

```
\setlength \leftmargini   { 2em }
\setlength \leftmarginii  { 2em }
\setlength \leftmarginiii { 2em }
\setlength  \labelsep  { 0.5em }
\setlength  \labelwidth { \leftmargini }
\addtolength \labelwidth { -\labelsep }
\cs_gset_nopar:Npn \@listi
  {
    \leftmargin \leftmargini
    \topsep 3pt plus 2pt minus 2.5pt
    \parsep 0pt
    \itemsep 3pt plus 2pt minus 3pt
  }
\cs_gset_eq:NN \@listI \@listi
\cs_gset_nopar:Npn \@listii
  {
    \leftmargin \leftmarginii
    \topsep    2pt plus 1pt minus 2pt
    \parsep    0pt plus 1pt
    \itemsep   \parsep
  }
\cs_gset_nopar:Npn \@listiii
  {
    \leftmargin \leftmarginiii
    \topsep    2pt plus 1pt minus 2pt
    \parsep    0pt plus 1pt
    \itemsep   \parsep
  }
\setlength \partopsep { 0pt }
```

⟨/class⟩

Part IX
ltx-talk-structure – Structural commands

1 ltx-talk-structure implementation

Start the DocStrip guards.

```
1 ⟨*class⟩
```

Identify the internal prefix.

```
2 ⟨@@=talk⟩
```

1.1 Frame title

\g__talk_frame_title_tl
\g__talk_frame_subtitle_tl

```
3 \tl_new:N \g__talk_frame_title_tl
4 \tl_new:N \g__talk_frame_subtitle_tl
```

(*End of definition for* \g__talk_frame_title_tl *and* \g__talk_frame_subtitle_tl.)

\frametitle

Just data storage: at the present no use of the optional argument.

```
5 \NewDocumentCommand \frametitle { D <> { all } O {#3} m }
6   {
7     \__talk_if_overlay:nT {#1}
8       { \tl_gset:Nn \g__talk_frame_title_tl {#3} }
9   }
10 \NewDocumentCommand \framesubtitle { D <> { all } O {#3} m }
11   {
12     \__talk_if_overlay:nT {#1}
13       { \tl_gset:Nn \g__talk_frame_subtitle_tl {#3} }
14   }
```

(*End of definition for* \frametitle. *This function is documented on page* ??.)

__talk_frame_title:n
__talk_frame_title_tagged:n

Inserting the frame title requires we deal with tagging as well as appearance: if there is a title, we need to tag just this part of the header.

```
15 \NewTemplateType { frametitle } { 1 }
16 \DeclareTemplateInterface { frametitle } { talk } { 1 }
17   {
18     after-vspace  : skip = \bigskipamount ,
19     before-vspace : skip = 0em ,
20     color         : tokenlist = ,
21     font          : tokenlist = \Large \bfseries
22   }
23 \DeclareTemplateCode { frametitle } { talk } { 1 }
24   {
25     after-vspace  = \l__talk_frametitle_after_skip ,
26     before-vspace = \l__talk_frametitle_before_skip ,
27     color         = \l__talk_frametitle_color_tl ,
28     font          = \l__talk_frametitle_font_tl
29   }
```

```
  {
    \noindent
    \vspace { \l__talk_frametitle_before_skip }
    \group_begin:
      \tl_if_empty:NF \l__talk_frametitle_color_tl
        { \color_select:V \l__talk_frametitle_color_tl }
      \l__talk_frametitle_font_tl
      \tl_if_blank:nF {#1}
        { \__talk_frame_title:n {#1} }
      \par
    \group_end:
    \vspace { \l__talk_frametitle_after_skip }
  }
\DeclareInstance { frametitle } { header } { talk } { }
\cs_new_protected:Npn \__talk_frame_title:n #1
  {
    \bool_if:NTF \g__talk_frame_tag_bool
      { \__talk_frame_title_tagged:n }
      { \use:n }
        {#1}
  }
\cs_new_protected:Npn \__talk_frame_title_tagged:n #1
  {
   \__talk_header_tag_begin:e
      {
        firstkid = true ,
        parent   = \int_use:N \g__talk_frame_struct_int ,
        tag      = frametitle ,
        title    = { \text_purify:n { \g__talk_frame_title_tl } } ,
      }
    \group_begin:
      \tagpdfparaOff
      #1
    \group_end:
    \__talk_header_tag_end:
  }
```

(*End of definition for* `\__talk_frame_title:n` *and* `\__talk_frame_title_tagged:n`.)

1.2 Sectioning

`\l__talk_section_tl`
`\g__talk_section_tl`
`\l__talk_subsection_tl`
`\g__talk_subsection_tl`
`\l__talk_subsubsection_tl`
`\g__talk_subsubsection_tl`

Two versions of the data store: one set locally (but at the top level) for general use, one set (and more importantly cleared) globally to allow insertion in the header area just once per name.

```
\tl_new:N \l__talk_section_tl
\tl_new:N \g__talk_section_tl
\tl_new:N \l__talk_subsection_tl
\tl_new:N \g__talk_subsection_tl
\tl_new:N \l__talk_subsubsection_tl
\tl_new:N \g__talk_subsubsection_tl
```

(*End of definition for* `\l__talk_section_tl` *and others.*)

`\section`
`\subsection`
`\subsubsection`
`\thesection`
`\thesubsection`
`\thesubsubsection`

Here, we need full LaTeX counters, so create them using the appropriate mechanism: that also means we can sort out counter dependency and the appearance (using the same setup

as in article). As (subsub)section numbers never increment inside frames, we remove these counters from the general tracker.

```
\newcounter { section }
\newcounter { subsection } [ section ]
\newcounter { subsubsection } [ subsection ]
\seq_gremove_all:Nn \l__talk_cnt_reset_seq { section }
\seq_gremove_all:Nn \l__talk_cnt_reset_seq { subsection }
\seq_gremove_all:Nn \l__talk_cnt_reset_seq { subsubsection }
\cs_gset:Npn \thesection { \@arabic \c@section }
\cs_gset:Npn \thesubsection { \thesection . \@arabic \c@subsection }
\cs_gset:Npn \thesubsubsection { \thesubsection . \@arabic \c@subsubsection }
```

(*End of definition for* `\section` *and others. These functions are documented on page* **??**.)

`\section`
`\subsection`
`\subsubsection`
`\insertsection`
`\insertsubsection`
`\insertsubsubsection`

The sectioning commands all have essentially the same form: we therefore create using a generator with the necessary conditionals in place. As we do not typeset sections at this stage, the code is quite different from article. This also means that the bookmark links need to point forward to the next slide: if that doesn't appear, the bookmarks will be out. Using the general scratch sequence here should be OK: t really is a one-off setting. We need a sequence to allow indexed mapping to avoid any extra setup for the depth value.

```
\seq_set_from_clist:Nn \l_tmpa_seq
  { section , subsection , subsubsection }
\seq_map_indexed_inline:Nn \l_tmpa_seq
  {
    \use:e
      {
        \NewDocumentCommand \exp_not:c { insert #2 } { }
          {
            \exp_not:N \tl_use:N
            \exp_not:c { l__talk_ #2 _tl }
          }
        \NewDocumentCommand \exp_not:c {#2}
          { s D <> { all } O {##4} m }
          {
            \exp_not:N \refstepcounter {#2}
            \tag_tool:n { sec-start = #2 , restore-para }
            \tl_set:Nn \exp_not:c { l__talk_ #2 _tl } {##4}
            \tl_gset_eq:NN \exp_not:c { g__talk_ #2 _tl }
              \exp_not:c { l__talk_ #2 _tl }
            \str_if_eq:nnT {#2} { section }
              { \tl_clear:N \exp_not:N \l__talk_subsection_tl }
            \str_if_eq:nnF {#2} { subsubsection }
              { \tl_clear:N \exp_not:N \l__talk_subsubsection_tl }
            \exp_not:N \addcontentsline { toc } {#2}
              {
                \exp_not:N \int_compare:nNnF {#1} >
                  { \exp_not:N \value { secnumdepth } }
                  {
                    \exp_not:N \protect \exp_not:N \numberline
                      { \exp_not:c { the #2 } }
                  }
                ##4
```

```
            }
          \hook_use:n { #2 / begin }
        }
      \hook_new:n { #2 / begin }
    }
}
```

(*End of definition for* `\section` *and others. These functions are documented on page* **??**.)

`\__talk_section_tagged:`

```
\cs_new_protected:Npn \__talk_section_tagged:
  {
    \clist_map_inline:nn { section , subsection , subsubsection }
      {
        \tl_if_empty:cF { g__talk_ ##1 _ tl }
          {
            \__talk_header_tag_begin:e
              {
                tag   = ##1 ,
                title = { \text_purify:v { g__talk_ ##1 _ tl } } ,
              }
            \__talk_header_tag_end:
            \tl_gclear:c { g__talk_ ##1 _ tl }
          }
      }
  }
```

(*End of definition for* `\__talk_section_tagged:`.)

1.3 Table of contents

`\@starttoc` The standard kernel implementation here deliberately overwrites the file as soon as it's read. That's no good for us as the table of contents can be read multiple times. So we modify the code: we start from the tagging-aware version (this may need to be revisited). We retain the LaTeX 2_ε code as much as possible.

```
\cs_gset_protected:Npn \@starttoc #1
  {
    \begingroup
      \makeatletter
      \UseTaggingSocket { toc / starttoc / before } {#1}
      \@input { \jobname .#1 }
      \UseTaggingSocket { toc / starttoc / after } {#1}
      \legacy_if:nT { @filesw }
        {
          \AddToHook { enddocument / afterlastpage }
            {
              \expandafter \newwrite \csname tf@ #1 \endcsname
              \immediate \openout \csname tf@ #1 \endcsname \jobname .#1 \relax
            }
        }
      \@nobreakfalse
    \endgroup
  }
```

(*End of definition for* `\@starttoc`. *This function is documented on page* **??**.)

`\tableofcontents` For the present simply print the output.

```
\NewDocumentCommand \tableofcontents { O { } }
  {
    \group_begin:
      \@starttoc { toc }
    \group_end:
  }
```

(*End of definition for* `\tableofcontents`. *This function is documented on page* **??**.)

`\l@section` `\l@subsection` `\l@subsubsection` `\__talk_toc_aux:nnnn` `\__talk_toc_dest:n` `\__talk_toc_dest:w` `\__talk_toc_level:nnnn`

Initial hard-coded versions to be templated once we have some other effects also working. We may need to look at this "higher up" as we will need to know the section numbers.

```
\cs_new_protected:Npn \l@section #1#2
  { \__talk_toc_aux:nnnn { 1 } { \bfseries \color { structure } } {#1} {#2} }
\cs_new_protected:Npn \l@subsection #1#2
  {
    \__talk_toc_aux:nnnn
      { 2 }
      {
        \skip_set:Nn \leftskip { 2em }
        \color { . }
      }
      {#1} {#2}
  }
\cs_new_protected:Npn \l@subsubsection #1#2
  {
    \__talk_toc_aux:nnnn
      { 3 }
      {
        \skip_set:Nn \leftskip { 4em }
        \color { . }
        \footnotesize
      }
      {#1} {#2}
  }
\cs_new_protected:Npn \__talk_toc_aux:nnnn #1#2#3#4
  {
    \int_compare:nNnTF { \value { section } } < 1
      { \use:n }
      { \__talk_toc_dest:n }
        { \__talk_toc_level:nnnn {#1} {#2} {#3} {#4} }
  }
```

We can extract the details for the TOC levels from `\@contentsline@destination`. At present, that is quite simple-minded: if we are in the current section, show fully, else make semi-opaque. Needs a rounded-out interface but the basic idea will be the same.

```
\cs_new_protected:Npn \__talk_toc_dest:n
  {
    \exp_after:wN \__talk_toc_dest:w \@contentsline@destination
      . 0 . 0 . 0 . \q_stop
  }
\cs_new_protected:Npn \__talk_toc_dest:w #1 . #2 . #3 . #4 . #5 \q_stop #6
```

```
  {
    \int_compare:nNnTF { \value { section } } = {#2}
      {#6}
      {
        \group_begin:
          \opacity_select:n { 0.2 }
          #6
        \group_end:
      }
  }
\cs_new_protected:Npn \__talk_toc_level:nnnn #1#2#3#4
  {
    \int_compare:nNnF {#1} > { \value { tocdepth } }
      {
        \group_begin:
          \noindent
          #2
          \UseHookWithArguments { contentsline / text / before } { 4 }
            {#1} {#3} {#4} { \@contentsline@destination }
          #3
          \UseHookWithArguments { contentsline / text / after } { 4 }
            {#1} {#3} {#4} { \@contentsline@destination }
          \UseHookWithArguments { contentsline / page / before } { 4 }
            {#1} {#3} {#4}
            { \@contentsline@destination }
         \UseHookWithArguments { contentsline / page / after } { 4 }
            {#1} {#3} {#4}
            { \@contentsline@destination }
          \par
        \group_end:
        \vfil
      }
  }
```

(*End of definition for* `\l@section` *and others. These functions are documented on page* **??**.)

```
\setcounter { tocdepth } { 2 }
```

1.4 Block environments

`description` (*env.*)
`quote` (*env.*)
`quotation` (*env.*)
`verse` (*env.*)
`stdquote` (*env.*)
`stdquotation` (*env.*)
`stdverse` (*env.*)

Stub logical environments: needed as the tagging setup expects these to exist.

```
\NewDocumentEnvironment { description } { } { } { }
\NewDocumentEnvironment { quote } { } { } { }
\NewDocumentEnvironment { quotation } { } { } { }
\NewDocumentEnvironment { verse } { } { } { }
\AddToHook { begindocument / before }
  {
    \clist_map_inline:nn { quote , quotation , verse }
      {
        \NewEnvironmentCopy { std #1 } {#1}
        \RenewDocumentEnvironment {#1} { D <> { all } !O { } }
          {
            \__talk_action_begin:n {##1}
            \begin { std #1 } [ {##2} ]
              \ignorespaces
```

```
          }
          {
            \end { std #1 }
            \__talk_action_end:
          }
      }
  }
```

`block` (*env.*)

```
\NewDocumentEnvironment { block } { D <> { all } m }
  {
    \__talk_action_begin:n {#1}
    \par
    \vbox_set:Nw \l__talk_tmp_box
      \group_begin:
        \medskip
        \leavevmode
        \normalfont \large \bfseries
        \color { structure }
        #2
        \par
        \medskip
      \group_end:
  }
  {
    \vbox_set_end:
    \box_use:N \l__talk_tmp_box
    \par
    \__talk_action_end:
  }
```

1.5 Lists

`\item`
`\__talk_item_parse_spec:w`
`\__talk_item_parse_spec:n`

Again, add the additional argument: here, we have to do a little gymnastics. The test for an overlay has to come after the standard item definition: in a list, items have to *close* the structure before them first, so if we test too early, we'd end up covering then uncovering straight away!

```
\AddToHook { begindocument / before }
  {
    \NewCommandCopy \stditem \item
    \RenewDocumentCommand \item { d <> o }
      {
        \IfNoValueTF {#2}
          { \stditem }
          { \stditem [ {#2} ] }
        \IfNoValueTF {#1}
          {
            \exp_after:wN \__talk_item_parse_spec:w
              \l__talk_action_spec_str < all > \q_stop
          }
          { \__talk_item_parse_spec:n {#1} }
      }
  }
```

Parsing the spec is a separate function here as there are a couple of routes to get here. At present we only have a `false` branch, but for spacing we likely will need to add something to the `true` branch too. The odd stuff with `\currentgrouplevel` here is needed so we only close the item at the correct nesting, allowing for the group that gets added.

```
\cs_new_protected:Npn \__talk_item_parse_spec:w #1 < #2 > #3 \q_stop
  { \__talk_item_parse_spec:n {#2} }
\cs_new_protected:Npn \__talk_item_parse_spec:n #1
  {
    \tl_if_blank:nF {#1}
      {
        \tl_set:Ne \l__talk_list_end_tl
          {
            \exp_not:N \int_compare:nNnT \tex_currentgrouplevel:D =
              { \int_use:N \tex_currentgrouplevel:D + 1 }
              {
                \__talk_action_end:
                \tl_clear:N \exp_not:N \l__talk_list_end_tl
              }
          }
        \__talk_action_begin:n {#1}
      }
  }
```

(*End of definition for* `\item`*,* `\__talk_item_parse_spec:w`*, and* `\__talk_item_parse_spec:n`*. This function is documented on page* **??**.)

`\l__talk_list_end_tl`

```
\tl_new:N \l__talk_list_end_tl
```

(*End of definition for* `\l__talk_list_end_tl`.)

`\__block_inter_item:`
`\endblockenv`

There are no currently no hooks for insertion at the end of list items, so we have to do it manually. We cannot target `\__block_list_item_end:`/`\__block_list_end:` as these change definition if tagging is suspended.

```
\cs_gset_protected:Npn \__block_inter_item:
  {
    \legacy_if:nT { @inlabel }
      { \indent \par }
   \mode_if_horizontal:T
     {
       \__block_skip_remove_last:
       \__block_skip_remove_last:
       \par
     }
    \l__talk_list_end_tl
    \__kernel_list_item_end:
    \__kernel_list_item_begin:
    \addpenalty \@itempenalty
    \addvspace \itemsep
  }
\cs_gset:Npn \endblockenv
  {
    \__block_debug_typeout:n { blockenv~common~ending \on@line }
    \bool_if:NT \l__block_level_incr_bool
```

```
      { \int_gdecr:N \g_block_nesting_depth_int }
    \legacy_if:nT { @inlabel }
      {
        \mode_leave_vertical:
        \legacy_if_gset_false:n { @inlabel }
      }
    \__block_if_list:T
      { \legacy_if:nT { @newlist } { \@noitemerr } }
    \mode_if_horizontal:TF
       {
         \__block_skip_remove_last:
         \__block_skip_remove_last:
         \par
       }
       { \@inmatherr { \end { \@currenvir } } }
    \l__talk_list_end_tl
    \__kernel_displayblock_end:
    \__block_if_list:T { \legacy_if_gset_false:n { @newlist } }
    \legacy_if:nF { @noparlist }
      {
        \__block_skip_set_to_last:N \l_tmpa_skip
        \dim_compare:nNnT \l_tmpa_skip > \c_zero_dim
          {
            \skip_vertical:n { - \l_tmpa_skip }
            \skip_vertical:n { \l_tmpa_skip + \parskip - \@outerparskip }
          }
        \addpenalty \@endparpenalty
        \addvspace \l__block_topsepadd_skip
      }
    \socket_use:n { block / endpe }
  }
```

(*End of definition for* `\__block_inter_item:` *and* `\endblockenv`*. This function is documented on page* ??.)

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

Allow for the classical beamer syntax.

```
\AddToHook { begindocument / before }
  {
    \clist_map_inline:nn { itemize , enumerate , description }
      {
        \RenewDocumentEnvironment {#1} { = { action-spec } !o }
          {
            \IfNoValueTF {##1}
              { \UseInstance { blockenv } {#1} { } }
              { \UseInstance { blockenv } {#1} {##1} }
          }
          { \endblockenv }
      }
  }
```

And add the structural color to item labels.

```
\AddToHook { begindocument / before }
  {
    \EditInstance { item } { basic }
      { label-format = \color { structure } #1 }
```

```
    \EditInstance { item } { description }
      { label-format = \normalfont \bfseries \color { structure } #1 }
  }
```

\l__talk_action_spec_str

Add an overlay key to the `block` template. Placed here, it applies *before* the `\item` starts, so we do not have to redefine the latter to do actions up-front. This also means it can apply to whatever we want it to within a block.

```
\keys_define:nn { template / block / display  }
  { action-spec .str_set:N = \l__talk_action_spec_str }
```

(*End of definition for* `\l__talk_action_spec_str`.)

1.6 Theorems, *etc.*

\newtheorem
\stdnewtheorem

We need to extend the creation of theorems in two ways: add the overlay argument, and add the counter to the list of those reset during overlay creation.

```
\NewCommandCopy \stdnewtheorem \newtheorem
\RenewDocumentCommand \newtheorem { m O {#1} m o }
  {
    \IfNoValueTF {#4}
      { \stdnewtheorem {#1} [ {#2} ] {#3} }
      { \stdnewtheorem {#1} [ {#2} ] {#3} [ {#4} ] }
    \NewEnvironmentCopy { std #1 } {#1}
    \RenewDocumentEnvironment {#1} { D <> { all } o }
      {
        \__talk_action_begin:n {##1}
        \IfNoValueTF {##2}
          { \begin { std #1 } }
          { \begin { std #1 } [ {##2} ] }
            \ignorespaces
      }
      {
        \end { std #1 }
        \__talk_action_end:
      }
  }
```

(*End of definition for* `\newtheorem` *and* `\stdnewtheorem`. *These functions are documented on page* **??**.)

⟨/class⟩

Part X
ltx-talk-title – Title pages

1 ltx-talk-title implementation

Start the DocStrip guards.

```
1 ⟨*class⟩
```

Identify the internal prefix.

```
2 ⟨@@=talk⟩
```

\institute
\subtitle
\@institute
\@subtitle

Simple storage at present: we use names similar to the kernel ones for author, etc., as this makes data management easier.

```
3 \cs_new_nopar:Npn \@institute { }
4 \cs_new_nopar:Npn \@subtitle { }
5 \NewDocumentCommand \institute { = { short-institute } O {#2} m }
6   { \cs_gset_nopar:Npn \@institute {#2} }
7 \NewDocumentCommand \subtitle { = { short-subtitle } O {#2} m }
8   { \cs_gset_nopar:Npn \@subtitle {#2} }
```

(*End of definition for* `\institute` *and others. These functions are documented on page* **??**.)

\l__talk_titlelem_after_skip
\l__talk_titlelem_before_skip
\l__talk_titlelem_color_tl
\l__talk_titlelem_font_tl
\l__talk_titlelem_tag_begin_tl
\l__talk_titlelem_tag_end_tl

As the various elements of the titlepage share certain characteristics, we use a single template and split them as instances.

```
9  \NewTemplateType { titlepage-element } { 1 }
10 \DeclareTemplateInterface { titlepage-element } { talk } { 1 }
11   {
12     after-skip : length = 0em ,
13     before-skip : length = 0em ,
14     color : tokenlist = . ,
15     font : tokenlist = \normalfont ,
16     tag-begin : tokenlist = ,
17     tag-end : tokenlist =
18   }
19 \DeclareTemplateCode { titlepage-element } { talk } { 1 }
20   {
21     after-skip  = \l__talk_titlelem_after_skip ,
22     before-skip = \l__talk_titlelem_before_skip ,
23     color       = \l__talk_titlelem_color_tl ,
24     font        = \l__talk_titlelem_font_tl ,
25     tag-begin   = \l__talk_titlelem_tag_begin_tl ,
26     tag-end     = \l__talk_titlelem_tag_end_tl
27   }
28   {
29     \tl_if_empty:nF {#1}
30       {
31         \vspace { \l__talk_titlelem_before_skip }
32         \group_begin:
33           \tl_if_empty:NF \l__talk_titlelem_color_tl
34             { \color_select:V \l__talk_titlelem_color_tl }
35           \l__talk_titlelem_font_tl
36           \l__talk_titlelem_tag_begin_tl
```

```
          #1
          \par
          \l__talk_titlelem_tag_end_tl
        \group_end:
        \vspace { \l__talk_titlelem_after_skip }
      }
  }
```

Standard settings are taken from beamer with minor adjustments.

```
\DeclareInstance { titlepage-element } { author } { talk }
  { before-skip = 1em }
\DeclareInstance { titlepage-element } { date } { talk }
  { after-skip = 0.5em }
\DeclareInstance { titlepage-element } { institute } { talk }
  { font = \scriptsize }
\DeclareInstance { titlepage-element } { subtitle } { talk }
  { before-skip = 0.25em , color = structure }
\DeclareInstance { titlepage-element } { title } { talk }
  {
    color = structure ,
    font = \Large ,
    tag-begin = \tag_struct_begin:n { tag = Title } ,
    tag-end = \tag_struct_end:
  }
```

(*End of definition for* `\l__talk_titlelem_after_skip` *and others.*)

\l__talk_titlepage_order_clist
\l__talk_titlepage_alignment_tl
\l__talk_titlepage_framestyle_tl
\l__talk_frame_alignment_tl

Here, we deal with the overall style: notice that frame vertical alignment actually applies elsewhere, which is why it doesn't show up in the template code part. As a result, we have a slightly repetitive key interface.

```
\NewTemplateType { titlepage } { 0 }
\DeclareTemplateInterface { titlepage } { talk } { 0 }
  {
    element-order : commalist =
      {
        title     ,
        subtitle  ,
        author    ,
        institute ,
        date
      } ,
    framestyle : tokenlist = talk ,
    horizontal-alignment : choice { left , center ,  right } = center ,
    vertical-alignment : choice { bottom , center , stretch , top } = center
  }
\DeclareTemplateCode { titlepage } { talk } { 0 }
  {
    element-order = \l__talk_titlepage_order_clist ,
    framestyle = \l__talk_titlepage_framestyle_tl ,
    horizontal-alignment =
      {
        left = \tl_set:Nn \l__talk_titlepage_alignment_tl { flushleft } ,
        center = \tl_set:Nn \l__talk_titlepage_alignment_tl { center } ,
        right = \tl_set:Nn \l__talk_titlepage_alignment_tl { flushright }
      } ,
```

```
    vertical-alignment =
      {
        bottom = \tl_set:Nn \l__talk_frame_alignment_tl { bottom } ,
        center = \tl_set:Nn \l__talk_frame_alignment_tl { center } ,
        stretch = \tl_set:Nn \l__talk_frame_alignment_tl { stretch } ,
        top = \tl_set:Nn \l__talk_frame_alignment_tl { top }
      }
  }
  {
    \tl_if_empty:NF \l__talk_titlepage_framestyle_tl
      { \exp_args:NV \thispagestyle \l__talk_titlepage_framestyle_tl }
    \begin { \l__talk_titlepage_alignment_tl }
      \cs_set_protected:Npn \and { \quad }
      \clist_map_inline:Nn \l__talk_titlepage_order_clist
        {
          \ExpandArgs { nnv } \UseInstance { titlepage-element }
            {##1} { @ ##1 }
        }
    \end { \l__talk_titlepage_alignment_tl }
  }
```

(*End of definition for* `\l__talk_titlepage_order_clist` *and others.*)

`\maketitle` A very simple setup.

```
\NewDocumentCommand \maketitle { O {} }
  {
    \bool_if:NTF \l__talk_frame_bool
      { \UseTemplate { titlepage } { talk } {#1} }
      {
        \begin { frame }
          \UseTemplate { titlepage } { talk } {#1}
        \end { frame }
      }
  }
```

(*End of definition for* `\maketitle`. *This function is documented on page* **??**.)

```
⟨/class⟩
```

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.

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

U

V

Y