The postnotes package
Code documentation

gusbrs

https://github.com/gusbrs/postnotes
https://www.ctan.org/pkg/postnotes

Version v0.4.1 – 2024-11-14

Contents

1 Initial setup

Start the DocStrip guards.

```
⟨*package⟩
```

Identify the internal prefix (LaTeX3 DocStrip convention).

```
⟨@@=postnotes⟩
```

The new syntax for file/package hooks, which the package assumes, requires kernel 2021-11-15 (`ltnews34`, `ltfilehook`). Furthermore, the kernel of 2022-06-01 introduced a couple of very nice features which simplifies the relation with hyperref (`ltnews35`, `hyperref-linktarget`): the provision of `\MakeLinkTarget` and the definition by the kernel of the starred version of `\ref`, which we can use regardless of hyperref being loaded. Also, since we followed the move to e-type expansion, to play safe we require the 2023-11-01 kernel or newer. Finally, the tagging sockets and block code required for tagging support need the 2024-06-01 kernel.

```
\def\postnotes@required@kernel{2024-06-01}
\NeedsTeXFormat{LaTeX2e}[\postnotes@required@kernel]
\providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
\IfFormatAtLeastTF{\postnotes@required@kernel}
  {}
  {%
    \PackageError{postnotes}{LaTeX kernel too old}
      {%
        'postnotes' requires a LaTeX kernel \postnotes@required@kernel\space or newer.%
      }%
  }%
\ProvidesExplPackage {postnotes} {2024-11-14} {0.4.1}
  {Endnotes for LaTeX}
```

`\l__postnotes_tmpa_tl` `\l__postnotes_tmpb_tl` `\l__postnotes_tmpa_seq` `\l__postnotes_tmpb_seq` `\l__postnotes_tmpa_box`

Temporary scratch variables.

```
\tl_new:N \l__postnotes_tmpa_tl
\tl_new:N \l__postnotes_tmpb_tl
\seq_new:N \l__postnotes_tmpa_seq
\seq_new:N \l__postnotes_tmpb_seq
\box_new:N \l__postnotes_tmpa_box
```

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

2 Data

`\__postnotes_data_name:n`

Returns the name of the property list variable which stores the data of the `\postnote` with ⟨*note id*⟩ number.

`\__postnotes_data_name:n {⟨note id⟩}`

```
\cs_new:Npn \__postnotes_data_name:n #1
  { g__postnotes_ #1 _data_prop }
\cs_generate_variant:Nn \__postnotes_data_name:n { e }
```

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

postnotes provides a number of hooks from the new hook system to grant some points of access in key places of the package. Note that hooks created with `\NewHook` are meant to be public interfaces (see https://chat.stackexchange.com/transcript/message/62955941#62955941, and following discussion).

`\__postnotes_store:nn` Stores the metadata and ⟨*note content*⟩ of `\postnote` with ID ⟨*note id*⟩, from where it is called. The `postnotes/note/store` hook is intended to add further data to the note, when required to support packages with specific needs.

`\__postnotes_store:nn {`⟨*note id*⟩`} {`⟨*note content*⟩`}`

```
\NewHook { postnotes/note/store }
\cs_new_protected:Npn \__postnotes_store:nn #1#2
  {
    \prop_new:c { \__postnotes_data_name:e {#1} }
    \prop_gput:cnn { \__postnotes_data_name:e {#1} } { type } { note }
    \prop_gput:cne { \__postnotes_data_name:e {#1} } { mark }
      { \l__postnotes_mark_tl }
    \prop_gput:cne { \__postnotes_data_name:e {#1} } { counter }
      { \int_use:N \c@postnote }
    \prop_gput:cne { \__postnotes_data_name:e {#1} } { sortnum }
      {
        \bool_if:NTF \l__postnotes_manual_sortnum_bool
          { \fp_use:N \l__postnotes_sort_num_fp }
          { \int_use:N \c@postnote }
      }
    \cs_if_exist:cT { chapter }
      {
        \prop_gput:cne { \__postnotes_data_name:e {#1} }
          { thechapter } { \thechapter }
      }
    \prop_gput:cne { \__postnotes_data_name:e {#1} } { thesection }
      { \thesection }
    \prop_gput:cne { \__postnotes_data_name:e {#1} } { pnsectname }
      { \g__postnotes_section_name_tl }
    \prop_gput:cne { \__postnotes_data_name:e {#1} } { pnsectid }
      { \int_use:N \g__postnotes_sectid_int }
    \prop_gput:cne { \__postnotes_data_name:e {#1} } { multibool }
      { \bool_to_str:N \l__postnotes_maybe_multi_bool }
    \prop_gput:cnn { \__postnotes_data_name:e {#1} } { content } {#2}
    \UseHook { postnotes/note/store }
  }
```

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

`\__postnotes_store_section:nn` Stores the metadata and ⟨*note content*⟩ of `\postnotesection` with ID ⟨*note id*⟩, from where it is called.

`\__postnotes_store_section:nn {`⟨*note id*⟩`} {`⟨*note content*⟩`}`

```
\cs_new_protected:Npn \__postnotes_store_section:nn #1#2
  {
    \prop_new:c { \__postnotes_data_name:e {#1} }
```

```
    \prop_gput:cnn { \__postnotes_data_name:e {#1} } { type } { section }
    \cs_if_exist:cT { chapter }
      {
        \prop_gput:cne { \__postnotes_data_name:e {#1} }
          { thechapter } { \thechapter }
      }
    \prop_gput:cne { \__postnotes_data_name:e {#1} } { thesection }
      { \thesection }
    \prop_gput:cnn { \__postnotes_data_name:e {#1} } { content } {#2}
  }
\cs_generate_variant:Nn \__postnotes_store_section:nn { ne }
```

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

`\__postnotes_prop_get:nnN`
`\__postnotes_prop_item:nn`
`\__postnotes_prop_gclear:n`

Convenience functions to retrieve and clear data from a note based on the ID number.

```
\__postnotes_prop_get:nnN {⟨note id⟩} {⟨property⟩} {⟨tl var to set⟩}
\__postnotes_prop_item:nn {⟨note id⟩} {⟨property⟩}
\__postnotes_prop_gclear:n {⟨note id⟩}
```

```
\cs_new_protected:Npn \__postnotes_prop_get:nnN #1#2#3
  {
    \prop_get:cnNF { \__postnotes_data_name:e {#1} } {#2} #3
      { \tl_clear:N #3 }
  }
\cs_new:Npn \__postnotes_prop_item:nn #1#2
  { \prop_item:cn { \__postnotes_data_name:e {#1} } {#2} }
\cs_new_protected:Npn \__postnotes_prop_gclear:n #1
  { \prop_gclear:c { \__postnotes_data_name:e {#1} } }
```

(*End of definition for* `\__postnotes_prop_get:nnN`, `\__postnotes_prop_item:nn`, *and* `\__postnotes_prop_gclear:n`.)

`\post@note`
`\__postnotes_store_labelseq:nn`
`\__postnotes_step_counteraux:nnn`

`\post@note` is the `\newlabel` equivalent for postnotes. Based on the kernel's `\@newl@bel` so that we get LaTeX checks for multiple and changed references for free (procedure learnt from zref). `\post@note`, when the `.aux` file is read, defines macros named `\postnote@r@`⟨*label type*⟩`@`⟨*id*⟩, according to the prefix set by `\c__postnotes_ref_prefix_tl`. `\__postnotes_store_labelseq:nn` is an auxiliary function to store the sequence of the labels in the `.aux` file, used to check for possible sorting problems due to floats. `\__postnotes_step_counteraux:nnn` handles counter stepping and storing for the counteraux option.

```
\post@note {⟨label type⟩} {⟨id⟩} {⟨label content (page)⟩}
  {⟨counteraux step⟩}
\__postnotes_store_labelseq:nn {⟨label type⟩} {⟨id⟩}
\__postnotes_step_counteraux:nnn {⟨label type⟩} {⟨id⟩}
  {⟨counteraux step⟩}
```

```
\tl_const:Nn \c__postnotes_ref_prefix_tl { postnote@r }
\seq_new:N \g__postnotes_labelseq_seq
\int_new:N \g__postnotes_postnote_counteraux_int
\prop_new:N \g__postnotes_counteraux_prop
\bool_new:N \g__postnotes_firstrun_bool
\bool_gset_true:N \g__postnotes_firstrun_bool
```

```
\cs_new_protected:Npn \__postnotes_store_labelseq:nn #1#2
  {
    \bool_lazy_any:nT
      {
        { \str_if_eq_p:nn {#1} { mark } }
        { \str_if_eq_p:nn {#1} { section } }
        { \str_if_eq_p:nn {#1} { preprint } }
      }
      { \seq_gput_right:Nn \g__postnotes_labelseq_seq { {#1} {#2} } }
  }
\cs_new_protected:Npn \__postnotes_step_counteraux:nnn #1#2#3
  {
    \bool_lazy_and:nnT
      { \g__postnotes_counteraux_bool }
      { \str_if_eq_p:nn {#1} { mark } }
      {
        \int_gadd:Nn \g__postnotes_postnote_counteraux_int { #3 }
        \prop_gput:Nne \g__postnotes_counteraux_prop { #2 }
          { \int_use:N \g__postnotes_postnote_counteraux_int }
      }
  }
\cs_new_protected:Npe \post@note #1#2#3#4
  {
    \exp_not:N \bool_gset_false:N \exp_not:N \g__postnotes_firstrun_bool
    \exp_not:N \__postnotes_store_labelseq:nn { #1 } { #2 }
    \exp_not:N \__postnotes_step_counteraux:nnn { #1 } { #2 } { #4 }
    \exp_not:N \@newl@bel { \c__postnotes_ref_prefix_tl } { #1 @ #2 } { #3 }
  }
```

(*End of definition for* `\post@note`, `\__postnotes_store_labelseq:nn`, *and* `\__postnotes_step_counteraux:nnn`.)

And ensure `\post@note`, `\postnote@setcounteraux`, and `\postnote@addtocounteraux` are defined in the `.aux` file. The hooks are the same used by hyperref for similar purpose.

```
\AddToHook { begindocument }
  {
    \legacy_if:nT { @filesw }
      {
        \iow_now:Ne \@mainaux
          {
            \token_to_str:N \providecommand
            \token_to_str:N \post@note [4] { }
          }
        \iow_now:Ne \@mainaux
          {
            \token_to_str:N \providecommand
            \token_to_str:N \postnote@setcounteraux [1] { }
          }
        \iow_now:Ne \@mainaux
          {
            \token_to_str:N \providecommand
            \token_to_str:N \postnote@addtocounteraux [1] { }
          }
      }
  }
```

```
\AddToHook { include/before }
  {
    \legacy_if:nT { @filesw }
      {
        \iow_now:Ne \@mainaux
          {
            \token_to_str:N \providecommand
            \token_to_str:N \post@note [4] { }
          }
        \iow_now:Ne \@mainaux
          {
            \token_to_str:N \providecommand
            \token_to_str:N \postnote@setcounteraux [1] { }
          }
        \iow_now:Ne \@mainaux
          {
            \token_to_str:N \providecommand
            \token_to_str:N \postnote@addtocounteraux [1] { }
          }
      }
  }
```

__postnotes_set_label:nnnn
__postnotes_set_mark_page_label:nn
__postnotes_set_section_page_label:n
__postnotes_set_text_page_label:n
__postnotes_set_print_page_label:n
__postnotes_set_pre_print_label:n

Label setting functions for each pertinent context. They must use `\iow_shipout_e:Nn`, since the main information we are interested in is the page.

```
\__postnotes_set_label:nnnn {⟨label type⟩} {⟨note id⟩} {⟨value⟩}
  {⟨counteraux step⟩}
\__postnotes_set_mark_page_label:nn {⟨note id⟩} {⟨counteraux step⟩}
\__postnotes_set_section_page_label:n {⟨note id⟩}
\__postnotes_set_text_page_label:n {⟨note id⟩}
\__postnotes_set_print_page_label:n {⟨note id⟩}
\__postnotes_set_pre_print_label:n {⟨note id⟩}
```

```
\cs_new_protected:Npn \__postnotes_set_label:nnnn #1#2#3#4
  {
    \legacy_if:nT { @filesw }
      {
        \iow_shipout_e:Nn \@auxout
          { \token_to_str:N \post@note { #1 } { #2 } { #3 } { #4 } }
      }
  }
\cs_new_protected:Npn \__postnotes_set_mark_page_label:nn #1#2
  { \__postnotes_set_label:nnnn { mark } { #1 } { \thepage } { #2 } }
\cs_generate_variant:Nn \__postnotes_set_mark_page_label:nn { ee }
\cs_new_protected:Npn \__postnotes_set_section_page_label:n #1
  { \__postnotes_set_label:nnnn { section } { #1 } { \thepage } { } }
\cs_generate_variant:Nn \__postnotes_set_section_page_label:n { e }
\cs_new_protected:Npn \__postnotes_set_text_page_label:n #1
  { \__postnotes_set_label:nnnn { text } { #1 } { \int_use:N \c@page } { } }
\cs_generate_variant:Nn \__postnotes_set_text_page_label:n { e }
\cs_new_protected:Npn \__postnotes_set_print_page_label:n #1
  { \__postnotes_set_label:nnnn { print } { #1 } { \int_use:N \c@page } { } }
\cs_generate_variant:Nn \__postnotes_set_print_page_label:n { e }
\cs_new_protected:Npn \__postnotes_set_pre_print_label:n #1
  { \__postnotes_set_label:nnnn { preprint } { #1 } { } { } }
```

```
\cs_generate_variant:Nn \__postnotes_set_pre_print_label:n { e }
```

(*End of definition for* `\__postnotes_set_label:nnnn` *and others.*)

`\__postnotes_get_pageref:Nn`
`\__postnotes_extract_pageref:n`

Reference data extraction functions.

`\__postnotes_get_pageref:Nn {⟨tl var to set⟩} {⟨label name⟩}`
`\__postnotes_extract_pageref:n {⟨label name⟩}`

```
\cs_new_protected:Npn \__postnotes_get_pageref:Nn #1#2
  {
    \cs_if_exist:cTF { \c__postnotes_ref_prefix_tl @ #2 }
      { \tl_set:Nv #1 { \c__postnotes_ref_prefix_tl @ #2 } }
      { \tl_clear:N #1 }
  }
\cs_generate_variant:Nn \__postnotes_get_pageref:Nn { Ne }
\cs_new:Npn \__postnotes_extract_pageref:n #1
  {
    \cs_if_exist:cTF { \c__postnotes_ref_prefix_tl @ #1 }
      { \exp_not:v { \c__postnotes_ref_prefix_tl @ #1 } }
      { \c_empty_tl }
  }
\cs_generate_variant:Nn \__postnotes_extract_pageref:n { e }
```

(*End of definition for* `\__postnotes_get_pageref:Nn` *and* `\__postnotes_extract_pageref:n`.)

3 Options

heading option

```
\keys_define:nn { postnotes/setup }
  {
    heading .cs_set_protected:Np = \pnheading ,
    heading .value_required:n = true ,
  }
```

`\pnheading` Provide default value for `\pnheading`.

```
\cs_if_exist:cTF { chapter }
  {
    \cs_new_protected:Npn \pnheading
      {
        \chapter*{\pntitle}
        \@mkboth{\pnheaderdefault}{\pnheaderdefault}
      }
  }
  {
    \cs_new_protected:Npn \pnheading
      {
        \section*{\pntitle}
        \@mkboth{\pnheaderdefault}{\pnheaderdefault}
      }
  }
```

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

`format` option

```
\tl_new:N \l__postnotes_print_format_tl
\keys_define:nn { postnotes/setup }
  {
    format .tl_set:N = \l__postnotes_print_format_tl ,
    format .initial:n = { \small } ,
    format .value_required:n = true ,
  }
```

`listenv` option

```
\tl_new:N \l__postnotes_print_env_tl
\bool_new:N \l__postnotes_print_as_list_bool
\keys_define:nn { postnotes/setup }
  {
    listenv .code:n =
      {
        \tl_if_eq:nnTF {#1} { none }
          {
            \bool_set_false:N \l__postnotes_print_as_list_bool
            \tl_set:Nn \l__postnotes_post_printnote_tl { \par }
```

A sensible default just in case. It should not get to be used though.

```
            \tl_set:Nn \l__postnotes_print_env_tl { itemize }
          }
          {
            \bool_set_true:N \l__postnotes_print_as_list_bool
            \tl_set:Nn \l__postnotes_print_env_tl {#1}
          }
      } ,
    listenv .initial:n = { postnoteslist } ,
    listenv .value_required:n = true ,
  }
```

A couple of built-in list environments provided for convenience, and `postnoteslist` as default. The horizontal setup of the label in these lists is based on the `description` environment of the standard classes (see the *The LaTeX Companion*).

```
\NewDocumentEnvironment { postnoteslist } { }
  {
    \list { }
      {
        \setlength { \leftmargin }    { 0pt }
        \setlength { \labelwidth }    { 0pt }
        \setlength { \itemindent }    { .5\parindent }
        \cs_set_eq:NN \makelabel \__postnotes_list_makelabel:n
        \setlength { \rightmargin }   { 0pt }
        \setlength { \listparindent } { \parindent }
        \setlength { \parsep }    { \parskip }
        \setlength { \itemsep }   { 0pt }
        \setlength { \topsep }    { .5\topsep }
        \setlength { \partopsep } { .5\partopsep }
      }
  }
  { \endlist }
\NewDocumentEnvironment { postnoteslisthang } { }
```

```
  {
    \list { }
      {
        \setlength { \leftmargin }    { 1em }
        \setlength { \labelwidth }    { -\leftmargin }
        \setlength { \itemindent }    { -2\leftmargin }
        \cs_set_eq:NN \makelabel \__postnotes_list_makelabel:n
        \setlength { \rightmargin }   { 0pt }
        \setlength { \listparindent } { \parindent }
        \setlength { \parsep }    { \parskip }
        \setlength { \itemsep }   { 0pt }
        \setlength { \topsep }    { .5\topsep }
        \setlength { \partopsep } { .5\partopsep }
      }
  }
  { \endlist }
\cs_new:Npn \__postnotes_list_makelabel:n #1
  { \hspace { \labelsep } \normalfont ~ #1 }
```

`makemark` and `maketextmark` options

The arguments are: `#1` is the mark, `#2` and `#3` are, respectively, the start and the end of the backlink.

```
\keys_define:nn { postnotes/setup }
  {
    makemark .cs_set:Np = \__postnotes_make_mark:nnn #1#2#3 ,
    makemark .value_required:n = true ,
```

From the default kernel definition of `\@makefnmark`.

```
    makemark .initial:n =
      { #2 \hbox { \@textsuperscript { \normalfont #1 } } #3 } ,
    maketextmark .cs_set:Np = \__postnotes_make_text_mark:nnn #1#2#3  ,
    maketextmark .value_required:n = true ,
    maketextmark .initial:n = { #2 #1 . #3 } ,
  }
\cs_generate_variant:Nn \__postnotes_make_mark:nnn { Vnn }
```

`pretextmark`, `posttextmark`, `postprintnote` options

```
\tl_new:N \l__postnotes_pre_textmark_tl
\tl_new:N \l__postnotes_post_textmark_tl
\tl_new:N \l__postnotes_post_printnote_tl
\keys_define:nn { postnotes/setup }
  {
    pretextmark .tl_set:N = \l__postnotes_pre_textmark_tl ,
    pretextmark .value_required:n = true ,
    posttextmark .tl_set:N = \l__postnotes_post_textmark_tl ,
    posttextmark .value_required:n = true ,
    postprintnote .tl_set:N = \l__postnotes_post_printnote_tl ,
    postprintnote .value_required:n = true ,
  }
```

`style` option

```
\keys_define:nn { postnotes/setup }
```

```
  {
    style .choice: ,
    style / endnotes .meta:n =
      {
        listenv = none ,
        format =
          {
            \footnotesize
            \setlength { \rightskip } { 0pt   }
            \setlength { \leftskip  } { 0pt   }
            \setlength { \parindent } { 1.8em }
          } ,
```

endnotes uses a zero width box to get the desired alignment to the right, but that does not play well with the backlinks, so we have a little more work to do to get this right.

```
        maketextmark =
          {
            \hbox_set:Nn \l__postnotes_tmpa_box
              { \@textsuperscript { \normalfont ##1 } }
            \skip_horizontal:n { - \box_wd:N \l__postnotes_tmpa_box }
            ##2 \box_use:N \l__postnotes_tmpa_box ##3
          } ,
      } ,
    style / pagenote .meta:n =
      {
        listenv = none ,
        format = { } ,
        pretextmark = { \noindent } ,
        maketextmark = { { \normalfont ##2 ##1 . ##3 } } ,
        posttextmark = { ~ } ,
      } ,
  }
```

hyperref and backlink options

```
\bool_new:N \l__postnotes_hyperlink_bool
\bool_new:N \l__postnotes_hyperref_warn_bool
\bool_new:N \l__postnotes_backlink_bool
\keys_define:nn { postnotes/setup }
  {
    hyperref .choice: ,
    hyperref / auto .code:n =
      {
        \bool_set_true:N \l__postnotes_hyperlink_bool
        \bool_set_false:N \l__postnotes_hyperref_warn_bool
      } ,
    hyperref / true .code:n =
      {
        \bool_set_true:N \l__postnotes_hyperlink_bool
        \bool_set_true:N \l__postnotes_hyperref_warn_bool
      } ,
    hyperref / false .code:n =
      {
        \bool_set_false:N \l__postnotes_hyperlink_bool
        \bool_set_false:N \l__postnotes_hyperref_warn_bool
```

```
      } ,
    hyperref .initial:n = auto ,
    hyperref .default:n = true ,
    backlink .bool_set:N = \l__postnotes_backlink_bool ,
    backlink .initial:n = true ,
    backlink .default:n = true ,
  }
\AddToHook { begindocument }
  {
    \IfPackageLoadedTF { hyperref }
      { }
      {
        \bool_if:NT \l__postnotes_hyperref_warn_bool
          { \msg_warning:nn { postnotes } { missing-hyperref } }
        \bool_set_false:N \l__postnotes_hyperlink_bool
      }
    \keys_define:nn { postnotes/setup }
      {
        hyperref .code:n =
          {
            \msg_warning:nnn { postnotes }
              { option-preamble-only } { hyperref }
          } ,
        backlink .code:n =
          {
            \msg_warning:nnn { postnotes }
              { option-preamble-only } { backlink }
          } ,
      }
  }
\msg_new:nnn { postnotes } { option-preamble-only }
  { Option~'#1'~only~available~in~the~preamble~\msg_line_context:. }
\msg_new:nnn { postnotes } { missing-hyperref }
  { Missing~'hyperref'~package.~Setting~'hyperref=false'. }
```

`multiple`, `multisep` options

As far as I can tell, the `multiple` option has its origins in the footmisc package, which offers this feature for footnotes. About this option, `footmisc.dtx` observes:

> This (revised) code derives from a suggestion by Alexander Rozhenko (the author of the manyfoot package): the intention is that footmisc and manyfoot should be able to 'interwork', in the sense that each would recognize the other's footnote marks and behave appropriately. The trick is that both `\footnote` and `\footnotemark` insert a marker (a cancelling pair of kerns of `\multiplefootnotemarker` (of opposite signs), which is detected in following `\footnote` or `\footnotemark` commands.

About the same option, `manyfoot.dtx` notes:

> To support `multiple` option from footmisc we add the `\FN@mf@prepare` command from footmisc (suggested by Frank Mittelbach).

Whatever the exact origin of this feature, the fact is that it has spread throughout the ecosystem, using not only the same basic mechanism but, typically, using *the same variables*: `\multiplefootnotemarker` and `\multfootsep`. With the intention, naturally, that different classes and packages can "interwork" with regard to this feature. And this became a sort of "shared feature" in the ecosystem (the list includes footmisc, KOMA-Script, eledmac, reledmac, tufte, memoir, parnotes, sidenotes, and now also postnotes, see discussion at https://chat.stackexchange.com/transcript/message/66421777#66421777). What is crucial for this interplay, however, is not quite the variables themselves, but *the value of the canceling pair of kerns*, stored in `\multiplefootnotemarker`. The fact that the same variables are used by most of the classes and packages which provide the feature speaks for convenience, in that a change in one of them reflects in all participating in the "pool".

Convenient as it is, this shared use of the same variables can only work as long as the community agrees in what these variables contain to some degree. As far as `\multiplefootnotemarker` goes, I see no divergence, and everybody uses the value of `3sp` for the canceling kerns from `\FN@mf@prepare`. `latex-lab-footnotes.dtx` even documents this value for this purpose in its list of "use of kerns to mark h-mode positions" (see https://chat.stackexchange.com/transcript/message/66421893#66421893). Things are not as smooth with regard to `\multfootsep` though. footmisc stores a plain comma in `\multfootsep` and applies formatting around it in `\FN@mf@check` (hard-coded as `\textsuperscript`). KOMA-Script also stores plain content in `\multfootsep` and applies the formatting in the check function. memoir, however, stores the formatting directly `\multfootsep` and applies no formatting later. Also, `latex-lab-footmisc.ltx`, in adding PDF tagging support for footmisc stores the tagging code directly in `\multfootsep`. I don't know if there are others, but these cases are already relevant enough to spoil things. The problem is not that they disagree on the value of `\multfootsep`, but on the function(s) this macro is supposed to perform. That given, any other parties trying to partake in this feature have to handle things differently depending on the value of `\multfootsep`. All in all, postnotes takes the cautious stance of using internal variables, instead of the shared ones, to implement the `multiple` option. The only thing that really needs to be common is the value of the canceling kerns of `3sp` in `\__postnotes_multiple_prepare:`.

```
\tl_new:N \l__postnotes_multisep_tl
\tl_const:Nn \c_postnotes_multi_notemarker_tl { 3sp }
\bool_new:N \l__postnotes_multiple_bool
\tl_new:N \l__postnotes_saved_spacefactor_multi_tl
\keys_define:nn { postnotes/setup }
  {
    multiple .bool_set:N = \l__postnotes_multiple_bool ,
    multiple .default:n = true ,
    multiple .initial:n = false ,
    multisep .tl_set:N = \l__postnotes_multisep_tl ,
    multisep .value_required:n = true ,
    multisep .initial:n = {,} ,
  }
```

I'm using the definitions in latex-lab-footmisc.ltx as base, see `texdoc latex-lab-footnotes`. For the formatting of the separator, though, I take inspiration from KOMA-Script and use `\__postnotes_make_mark:nnn`, instead of hard-coding `\textsuperscript`.

```
\cs_new_protected:Npn \__postnotes_multiple_prepare:
  {
```

```
    \bool_if:NT \l__postnotes_multiple_bool
      {
        \kern -\c_postnotes_multi_notemarker_tl
        \kern  \c_postnotes_multi_notemarker_tl
        \scan_stop:
      }
  }
```

Note that `\__postnotes_multiple_check:` has to be called before any whatsits (labels, anchors, etc.) are placed, since they destroy `\lastkern` (see https://chat.stackexchange.com/transcript/message/66554870#66554870, thanks Ulrike Fischer).

```
\cs_new_protected:Npn \__postnotes_multiple_check:
  {
    \bool_if:NT \l__postnotes_multiple_bool
      {
        \dim_compare:nNnT
          { \c_postnotes_multi_notemarker_tl } = { \lastkern }
          {
            \tl_set:Ne \l__postnotes_saved_spacefactor_multi_tl
              { \int_use:N \spacefactor }
            \unkern
            \unkern
            \tag_socket_use:n { postnotes/multisep/begin }
            \__postnotes_make_mark:Vnn \l__postnotes_multisep_tl { } { }
            \tag_socket_use:n { postnotes/multisep/end }
            \spacefactor \l__postnotes_saved_spacefactor_multi_tl
            \scan_stop:
          }
      }
  }
```

`sort` option

```
\bool_new:N \l__postnotes_sort_bool
\keys_define:nn { postnotes/setup }
  {
    sort .bool_set:N = \l__postnotes_sort_bool ,
    sort .initial:n = true ,
    sort .default:n = true ,
  }
```

`checkduplicates` and `checkfloats` options

```
\bool_new:N \l__postnotes_check_dupli_bool
\bool_new:N \l__postnotes_check_floats_bool
\keys_define:nn { postnotes/setup }
  {
    checkduplicates .bool_set:N = \l__postnotes_check_dupli_bool ,
    checkduplicates .default:n = true ,
    checkduplicates .initial:n = true ,
    checkfloats .bool_set:N = \l__postnotes_check_floats_bool ,
    checkfloats .default:n = true ,
    checkfloats .initial:n = false ,
  }
```

maybemulti option

```
\bool_new:N \l__postnotes_maybe_multi_bool
\keys_define:nn { postnotes/setup }
  {
    maybemulti .bool_set:N = \l__postnotes_maybe_multi_bool ,
    maybemulti .default:n = true ,
    maybemulti .initial:n = false ,
  }
```

counteraux option

```
\bool_new:N \g__postnotes_counteraux_bool
\keys_define:nn { postnotes/setup }
  {
    counteraux .bool_gset:N = \g__postnotes_counteraux_bool ,
    counteraux .default:n = true ,
    counteraux .initial:n = false ,
  }
\AddToHook { begindocument/before }
  {
    \bool_if:NT \g__postnotes_counteraux_bool
      { \postnotesetup { sort=false } }
    \keys_define:nn { postnotes/setup }
      {
        counteraux .code:n =
          {
            \msg_warning:nnn { postnotes }
              { option-preamble-only } { counteraux }
          } ,
      }
  }
```

\pnsetcounteraux
\pnaddtocounteraux
\postnote@setcounteraux
\postnote@addtocounteraux

\pnsetcounteraux{⟨*int*⟩}
\pnaddtocounteraux{⟨*int*⟩}
\postnote@setcounteraux{⟨*int*⟩}
\postnote@addtocounteraux{⟨*int*⟩}

```
\cs_new_protected:Npn \postnote@setcounteraux #1
  { \int_gset:Nn \g__postnotes_postnote_counteraux_int { #1 } }
\cs_new_protected:Npn \postnote@addtocounteraux #1
  { \int_gadd:Nn \g__postnotes_postnote_counteraux_int { #1 } }
\NewDocumentCommand \pnsetcounteraux { m }
  {
    \@bsphack
    \legacy_if:nT { @filesw }
      {
        \iow_shipout_e:Nn \@auxout
          { \token_to_str:N \postnote@setcounteraux { #1 } }
      }
    \@esphack
  }
\NewDocumentCommand \pnaddtocounteraux { m }
  {
    \@bsphack
    \legacy_if:nT { @filesw }
```

```
      {
        \iow_shipout_e:Nn \@auxout
          { \token_to_str:N \postnote@addtocounteraux { #1 } }
      }
    \@esphack
  }
```

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

\postnotesetup

`\postnotesetup` Provide `\postnotesetup`.

`\postnotesetup{`⟨*options*⟩`}`

```
\NewDocumentCommand \postnotesetup { m }
  { \keys_set:nn { postnotes/setup } {#1} }
```

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

4 \postnote

Different from the traditional `\footnotemark` / `\footnotetext` system, in the context of end notes, the functionality which corresponds to `\footnotetext` is simply to store the data to be typeset later. Hence, some of the problems that afflict footnotes do not apply to end notes. Namely, and as far as I can tell, they can be used in "inner horizontal mode" (`\mbox` etc.), and in math mode, and if the "text" will be typeset in the same page as the "mark" is of little concern.

However, the separation between "mark" and "text" is still useful in other contexts: floats and contexts where multiple typesetting passes are performed. David Carlisle and Ulrike Fischer shared some thoughts on the matter at the TeX.SX chat: https://chat.stackexchange.com/transcript/message/60754383#60754383.

The interesting questions here are: if they are replaceable in their roles in these contexts and how much would we lose by providing them. In analyzing this, we have to distinguish two situations: when `\footnotemark` is called with no argument (and thus steps the counter), and when it is called with the optional argument (and thus refrains from stepping the counter).

For floats, the problem they pose is that they may disturb the *ordering* of the notes. This particular issue can be solved by using `\footnotemark` without argument, and manually adjusting the counter on subsequent calls to `\footnotetext`. A good example of the technique is https://tex.stackexchange.com/a/43694. True, a user may wish to specify the mark explicitly, but doesn't necessarily need to do it to solve the ordering issue.

Multiple typesetting passes of content are much harder. And they abound: the standard classes' `\caption` typesets the caption once, if it is short, but twice if it is longer than a line; amsmath's math environments perform a measuring pass before actually typesetting the equations; amsmath's `\text` macro runs the contents through `\mathchoice` (which typesets the contents four times) when in math mode; tabularx and tabularray also perform measuring passes of their tables; so does csquotes' blockquotes; and certainly

more that I'm unaware. A number of these places offer some one or another way to mitigate the issue: amsmath, tabularx, csquotes and (optionally) tabularray restore counter values after measuring steps; amsmath offers a boolean to indicate when it is a measuring pass; csquotes offers further handles. But the standard `\caption` offers none, and neither does amsmath's `\text` macro. Well, the pkgcaption package can disable the multiple passes for `\caption` with the option `singlelinecheck`, but it is not reasonable to require it for our purposes, so we must assume the worst case.

Enrico Gregorio is categorical in stating that `\endnotemark` and `\endnotetext` are required for enotez to handle `\caption`, which apparently it didn't offer originally: "The package should implement `\endnotemark` and `\endnotetext` for this case. According to the documentation, the author deems them to not be needed: he's wrong." (https://tex.stackexchange.com/a/314937). See also https://tex.stackexchange.com/a/43794 and https://tex.stackexchange.com/a/358207.

In this scenario, when there's no way around the multiple passes, `\footnotemark` can only handle the general case if used with an argument, precisely because it inhibits the stepping of the counter. Otherwise the counter is stepped multiple times, and we'd get the wrong number (and mark). In some circumstances, if we know the number of passes is deterministic, we might get away by adjusting the counter manually (`\caption` may be dealt with this way: if we know it to be two lines, we can decrement the counter before it and get correct results, even hyperlinked). But in cases which adjusting the counter is sufficient, end notes can be dealt with in the same way, and doesn't need the separation between "mark" and "text". So, what is distinctive of the kernel's footnote apparatus, which allows it as much flexibility as one would like, is receiving an arbitrary number as argument and not stepping the counter. And as far as `\footnotemark` and `\footnotetext` are concerned, the main point of the optional argument is really to "manually establish the relation" between the two of them. So, if *not stepping the counter* is what is needed to handle the general case, is it viable to do so? What would we loose in so doing?

When receiving an arbitrary number as argument, as the kernel functionality for footnotes and other endnotes packages do, this value is expected to the printed as such, hence it must correspond to the `postnote` counter (in our case). But this counter is in the hands of the user, and can be reset along the document, thus its uniqueness cannot be ensured. But not stepping `postnote` is perfectly viable, as it just aims at storing how the mark is to be typeset. However, not stepping the ID counter would complicate things considerably. Not doing so implies we'd lose the connection we have between the "mark" and the corresponding "text". We might add the "text" to the queue, but all the metadata would be lost, including the hyperref anchor, but really the set of data without which the kind of functionality offered would be nonviable, or severely hampered. Not stepping `postnote` but stepping the ID counter also is not sufficient, because we'd get a note in duplicity. We could naively think that a gap in the ID is not a problem, and just not add the duplicate to the queue. But how could we tell the difference between a legitimate and an illegitimate step of the ID counter?

I have not been able to devise a way to "reconnect" "text" and "mark" in the absence of the unique ID counter. The most promising idea was to have mandatory arguments to `\postnotemark` and `\postnotetext` receiving a ⟨*label*⟩ which we could use to identify their counterparts, but I was not able to go through with this, and the attempts all increased complexity considerably. It is not just a label/ref system, there's got to be a one-to-one correspondence between the sets, uniqueness has to be ensured on both sides, and there cannot be "lone" marks or texts (a bijection). Besides, this label based system of identification would have to live side-by-side with the one based on the

counter. So, even if we'd have unique IDs, we wouldn't know beforehand in what form it comes. Considering the ID is used to build the variable name in which we store the note's information, this would also complicate things.

Besides, there are ways to get things working with multiple passes without the "mark"/"text" partition. As mentioned, there are a number of cases which offer some kind of "handle" or way to identify the multiple passes. csquotes has a dedicated hook that can be used. amsmath sets the `measuring@` boolean (which hyperref also defines). So, not all cases are as tricky as `\caption` or `\text`, and even that can be decently dealt with without a separation between "mark" and "text". Besides, in difficult cases, the package offers a `nomark` option to `\postnote` to place a note, but typeset no mark. Then we can typeset a mark with `\postnoteref` referring to a `\label` in the note of interest. This would result in a correct mark without duplicity, and in a correct link from there to the note's text at `\printpostnotes`. The drawback is that the placement of `\postnote` would be important, and results sensitive to it. All the metadata is collected at the point of `\postnote`, anchor included, not at the point of `\postnoteref`. So the consequences are a slightly off backlink, possibly imprecise metadata, etc. Considering hyperref itself shies away completely from linking `\footnotemark` with an argument, I'd say there's some gain.

The truth is there are some trade-offs, there's no "ideal" solution. Still, all in all, my judgment is that the unique ID counter is worth more than the inconveniences of an ocasional `\postnote[nomark]` referenced with `\postnoteref`, and even that should not be needed much. So, for the time being, until something else shakes this balance, I won't be offering `\postnotemark` and `\postnotetext`.

For the hyperref support for cross-references in `\postnote`, I've moved back and forth quite a lot. One of the ideas I fancied was using `\refstepcounter` and let hyperref do its job. But, since I want to have control of the anchor/destination name on both "sides", I'd have to set `\theHpostnote` locally before calling `\refstepcounter`, otherwise results might sensitive to user calls to `\counterwithin` (see https://github.com/latex3/hyperref/issues/230, thanks Ulrike Fischer). However, even if that worked well for the default case, we still had to setup things manually, in case of a manually supplied mark. All in all, I'm just calling `\stepcounter`, setting the relevant cross-reference variables once and setting the anchor manually.

`postnote` is the public, user facing, counter for `\postnote`. It determines how the note's mark gets to be typeset. It can be reset, set, and have its printed representation changed. Of course, whether those are meaningful is up to the user.

```
\newcounter { postnote }
```

\g__postnotes_note_id_int
\l_postnotes_note_id_tl
\g__postnotes_queue_seq
\l__postnotes_counteraux_step_int
\l__postnotes_mark_typeset_tl
\l__postnotes_note_set_labels_tl

`\g__postnotes_note_id_int` is the internal, unique counter which provides the ID number of each note. It ties "mark" and "text" together, is also the connection between each note and its data, including the content, which is stored in a property list named according to `\__postnotes_data_name:n` and the ID number. `\l_postnotes_note_id_tl` is a convenience variable storing the counter's value. `\g__postnotes_queue_seq` stores the sequence of notes' IDs to be processed by the next call of `\printpostnotes`.

```
\int_new:N \g__postnotes_note_id_int
\tl_new:N \l_postnotes_note_id_tl
\tl_set:Nn \l_postnotes_note_id_tl { \int_use:N \g__postnotes_note_id_int }
\seq_new:N \g__postnotes_queue_seq
\int_new:N \l__postnotes_counteraux_step_int
```

```
\tl_new:N \l__postnotes_mark_typeset_tl
\tl_new:N \l__postnotes_note_set_labels_tl
```

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

`\postnote` Provide `\postnote`.

`\postnote[`⟨*options*⟩`]{`⟨*note text*⟩`}`

```
\NewDocumentCommand \postnote { O { } +m }
  { \__postnotes_note:nn {#1} {#2} }
```

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

`\__postnotes_note:nn` The internal version of `\postnote`. The `postnotes/note/begin` hook is meant to provide a place from where some additional setup for the note can be performed. This is being used for adding support for some features/packages, but can also be used, for example, to add an extra local property for zref.

`\__postnotes_note:nn [`⟨*options*⟩`] {`⟨*note content*⟩`}`

```
\NewHook { postnotes/note/begin }
\cs_new_protected:Npn \__postnotes_note:nn #1#2
  {
    \group_begin:
      \keys_set:nn { postnotes/note } {#1}
      \bool_if:NT \l__postnotes_nomark_bool { \@bsphack }
      \__postnotes_inhibit_note:F
        {
          \int_gincr:N \g__postnotes_note_id_int
          \tl_if_empty:NTF \l__postnotes_mark_tl
            {
              \stepcounter { postnote }
              \int_set:Nn \l__postnotes_counteraux_step_int { 1 }
              \bool_if:NT \g__postnotes_counteraux_bool
                {
                  \exp_args:NNe \prop_gpop:NnNT \g__postnotes_counteraux_prop
                    { \l_postnotes_note_id_tl } \l__postnotes_tmpa_tl
                    { \int_set:Nn \c@postnote { \l__postnotes_tmpa_tl } }
                  \tl_clear:N \l__postnotes_tmpa_tl
                }
              \tl_set:Ne \l__postnotes_mark_tl { \thepostnote }
            }
            { \int_set:Nn \l__postnotes_counteraux_step_int { 0 } }
          \UseHook { postnotes/note/begin }
          \seq_gput_right:Ne \g__postnotes_queue_seq
            { \l_postnotes_note_id_tl }
          \cs_set:Npn \@currentcounter { postnote }
          \cs_set:Npe \@currentlabel { \p@postnote \l__postnotes_mark_tl }
          \cs_gset:Npe \@currentHref
            { postnote. \l_postnotes_note_id_tl .mark }
          \__postnotes_store:nn { \l_postnotes_note_id_tl } {#2}
          \tl_set_eq:NN \l__postnotes_mark_typeset_tl \l__postnotes_mark_tl
```

Prefer `label` for typesetting measuring passes, if available, see comments at `\__postnotes_inhibit_note:F`.

```
          \bool_lazy_or:nnT
            { \g__postnotes_counteraux_bool }
            { \l__postnotes_maybe_multi_bool }
            {
              \bool_lazy_and:nnT
                { ! \g__postnotes_firstrun_bool }
                {
                  ! \cs_if_exist_p:c
                    { \c__postnotes_ref_prefix_tl @ mark @ \l_postnotes_note_id_tl }
                }
                { \__postnotes_get_label_if_exist:N \l__postnotes_mark_typeset_tl }
            }
          \tl_set:Nn \l__postnotes_note_set_labels_tl
            {
              \MakeLinkTarget* { postnote. \l_postnotes_note_id_tl .mark }
              \__postnotes_set_mark_page_label:ee { \l_postnotes_note_id_tl }
                { \int_use:N \l__postnotes_counteraux_step_int }
              \__postnotes_set_user_labels:
            }
          \bool_if:NTF \l__postnotes_nomark_bool
            {
              \tag_socket_use:n { postnotes/nomark/begin }
              \l__postnotes_note_set_labels_tl
              \tag_socket_use:n { postnotes/nomark/end }
            }
            {
              \__postnotes_typeset_mark:eVN
                { \l_postnotes_note_id_tl } \l__postnotes_mark_typeset_tl
                \l__postnotes_note_set_labels_tl
            }
        }
      \bool_if:NT \l__postnotes_nomark_bool { \@esphack }
    \group_end:
  }
```

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

Options for `\postnote`.

```
\tl_new:N \l__postnotes_mark_tl
\bool_new:N \l__postnotes_nomark_bool
\fp_new:N \l__postnotes_sort_num_fp
\str_new:N \l__postnotes_note_label_str
\bool_new:N \l__postnotes_manual_sortnum_bool
\keys_define:nn { postnotes/note }
  {
    markstr .tl_set:N = \l__postnotes_mark_tl ,
    markstr .value_required:n = true ,
    sortnum .code:n =
      {
        \fp_set:Nn \l__postnotes_sort_num_fp {#1}
        \bool_set_true:N \l__postnotes_manual_sortnum_bool
      } ,
```

```
    sortnum .value_required:n = true ,
    mark .meta:n =
      {
        markstr = {#1} ,
        sortnum = {#1} ,
      } ,
    mark .value_required:n = true ,
    nomark .bool_set:N = \l__postnotes_nomark_bool ,
    nomark .default:n = true ,
    label .str_set:N = \l__postnotes_note_label_str ,
    label .value_required:n = true ,
    maybemulti .bool_set:N = \l__postnotes_maybe_multi_bool ,
    maybemulti .default:n = true ,
  }
```

__postnotes_inhibit_note:TF

In contexts of multiple passes of content, it may be needed, or preferred, to inhibit the note altogether to avoid side effects and duplicity. This conditional, obviously, will always return the true branch unless something is done in the `postnotes/note/inhibit` hook. This hook is meant to handle support for packages or features which may justify note inhibition, and the code there should set `\l__postnotes_inhibit_note_bool`, `\l__postnotes_print_plain_mark_bool` and `\l__postnotes_print_plain_mark_stepcounter_bool` as appropriate to the case.

```
\bool_new:N \l__postnotes_inhibit_note_bool
\bool_new:N \l__postnotes_print_plain_mark_bool
\bool_new:N \l__postnotes_print_plain_mark_stepcounter_bool
\NewHook { postnotes/note/inhibit }
\prg_new_protected_conditional:Npnn \__postnotes_inhibit_note: { F }
  {
    \bool_set_false:N \l__postnotes_inhibit_note_bool
    \bool_set_false:N \l__postnotes_print_plain_mark_bool
    \bool_set_false:N \l__postnotes_print_plain_mark_stepcounter_bool
    \UseHook { postnotes/note/inhibit }
```

Printing a plain mark here may be needed because, if we are inhibiting the note in a "measuring context" and omit it completely, the measuring being performed will be off by the size of the mark. So, to ensure the measuring can be done correctly, we place the mark. What to do with the counter itself, depends on the situation. In places that are known to restore the counter values after the measuring pass, we can let the counter be stepped. And, actually we should do so, for example, in a `tabularx` with multiple postnotes, if we don't step the counter, all the measuring will be done with the number of the first note. Otherwise, we don't actually step the counter but, to typeset correctly the mark that would be printed if the counter had been stepped, we increment `\c@postnote` locally and grouped, and smuggle `\thepostnote` out of the group.

```
    \bool_lazy_all:nT
      {
        { \l__postnotes_inhibit_note_bool }
        { \l__postnotes_print_plain_mark_bool }
        { ! \l__postnotes_nomark_bool }
      }
      {
        \tl_if_empty:NTF \l__postnotes_mark_tl
          {
            \bool_if:NTF \l__postnotes_print_plain_mark_stepcounter_bool
```

```
            {
              \stepcounter { postnote }
              \tl_set:Ne \l__postnotes_mark_typeset_tl { \thepostnote }
            }
            {
              \group_begin:
                \int_incr:N \c@postnote
                \exp_args:NNNe
                  \group_end:
                  \tl_set:Nn \l__postnotes_mark_typeset_tl { \thepostnote }
            }
```

If the note has a `label`, use a cross-reference to that as the mark instead. In principle, the procedure above is expected to work reasonably well for cases where we know whether we are in a measuring pass or not, and how it handles the counters (if it restores counters or not). This is true though, only if we are going in the expansion order of the document. If we are using the `counteraux` option, the mere sequence of the notes is no longer an indicator of who is a measuring pass of who, indeed the measuring pass does not even get to the `.aux` file. If the label exists, though, it is *known to be right* regardless of the case, since it belongs to the pass which actually gets typeset. Hence, if we have a label, it is more general and more reliable: use it.

```
            \__postnotes_get_label_if_exist:N \l__postnotes_mark_typeset_tl
          }
          { \tl_set_eq:NN \l__postnotes_mark_typeset_tl \l__postnotes_mark_tl }
        \group_begin:
          \socket_assign_plug:nn { tagsupport/postnotes/multisep/begin } { noop }
          \socket_assign_plug:nn { tagsupport/postnotes/multisep/end } { noop }
          \__postnotes_typeset_mark_wrapper:nnn
            { \__postnotes_make_mark:Vnn \l__postnotes_mark_typeset_tl { } { } }
            { } { }
        \group_end:
      }
    \bool_if:NTF \l__postnotes_inhibit_note_bool
      { \prg_return_true:  }
      { \prg_return_false: }
  }
```

(*End of definition for* `\__postnotes_inhibit_note:TF`.)

`\__postnotes_get_label_if_exist:N`

`\__postnotes_get_label_if_exist:N` ⟨`\l__postnotes_mark_tl`⟩

```
\cs_new_protected:Npn \__postnotes_get_label_if_exist:N #1
  {
    \str_if_empty:NTF \l__postnotes_note_label_str
      {
        \str_if_empty:NF \l__postnotes_note_zlabel_str
          {
            \exp_args:NV \zref@ifrefundefined \l__postnotes_note_zlabel_str
              { }
              {
                \exp_args:NV \zref@ifrefcontainsprop
                  \l__postnotes_note_zlabel_str
                  { postnote@mark }
                  {
```

```
                    \exp_args:NNNo \exp_args:NNo \tl_set:Nn #1
                      {
                        \zref@extract
                          { \l__postnotes_note_zlabel_str } { postnote@mark }
                      }
                  }
                  { }
              }
          }
      }
      {
        \exp_args:Ne \property_if_recorded:nnT
          { postnotes@ \l__postnotes_note_label_str }
          { postnotes/mark }
          {
            \tl_set:Ne #1
              {
                \property_ref:ee
                  { __postnotes_ \l__postnotes_note_label_str } { postnotes/mark }
              }
          }
      }
  }
```

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

`\__postnotes_typeset_mark:nnN` `\__postnotes_typeset_mark_wrapper:nnn`

Auxiliary functions for mark typesetting in `\__postnotes_note:nn`. `\__postnotes_typeset_mark_wrapper:nnn` is based on the definition of `\@footnotemark` in the kernel.

`\__postnotes_typeset_mark:nnN {`⟨*note id*⟩`} {`⟨*mark*⟩`} {`⟨*labels*⟩`}`
`\__postnotes_typeset_mark_wrapper:nnn {`⟨*mark*⟩`}`
`{`⟨*begin tagging*⟩`} {`⟨*end tagging*⟩`}`

```
\cs_new_protected:Npn \__postnotes_typeset_mark:nnN #1#2#3
  {
    \__postnotes_typeset_mark_wrapper:nnn
      {
        #3
        \bool_if:NTF \l__postnotes_hyperlink_bool
          {
            \__postnotes_make_mark:nnn {#2}
              { \hyper@linkstart { link } { postnote. #1 .text } }
              { \hyper@linkend }
          }
          { \__postnotes_make_mark:nnn {#2} { } { } }
      }
      { \tag_socket_use:n { postnotes/mark/begin } }
      { \tag_socket_use:n { postnotes/mark/end }   }
  }
\cs_generate_variant:Nn \__postnotes_typeset_mark:nnN { eVN }
\tl_new:N \l__postnotes_saved_spacefactor_tl
\cs_new_protected:Npn \__postnotes_typeset_mark_wrapper:nnn #1#2#3
  {
    \mode_leave_vertical:
    \mode_if_horizontal:T
```

```
      {
        \tl_set:Ne \l__postnotes_saved_spacefactor_tl
          { \int_use:N \spacefactor }
        \__postnotes_multiple_check:
        \nobreak
      }
    #2 % begin tagging socket
    #1 % mark
    #3 % end tagging socket
    \__postnotes_multiple_prepare:
    \mode_if_horizontal:T
      { \spacefactor \l__postnotes_saved_spacefactor_tl }
    \scan_stop:
  }
```

(*End of definition for* `\__postnotes_typeset_mark:nnN` *and* `\__postnotes_typeset_mark_wrapper:nnn`.)

`\__postnotes_set_user_labels:` Auxiliary function for user label setting in `\__postnotes_note:nn`. Supports the `label` and `zlabel` options of `\postnote`.

```
\cs_new_protected:Npn \__postnotes_set_user_labels:
  {
    \str_if_empty:NF \l__postnotes_note_label_str
      {
        \exp_args:NV \label \l__postnotes_note_label_str
        \property_record:ee { __postnotes_ \l__postnotes_note_label_str }
          { postnotes/mark }
      }
    \str_if_empty:NF \l__postnotes_note_zlabel_str
      { \exp_args:NV \zlabel \l__postnotes_note_zlabel_str }
  }
\property_new:nnnn { postnotes/mark } { now } { } { \l__postnotes_mark_tl }
```

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

5 \postnoteref

`\postnoteref` Provide `\postnoteref`.

`\postnoteref`⟨*⟩{⟨*label*⟩}

```
\NewDocumentCommand \postnoteref { s m }
  { \__postnotes_note_ref:nn {#1} {#2} }
```

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

`\__postnotes_note_ref:nn` The internal version of `\postnoteref`.

`\__postnotes_note_ref:nn` {⟨*star bool*⟩} {⟨*label*⟩}

```
\tl_new:N \l__postnotes_note_ref_label_tl
\cs_new_protected:Npn \__postnotes_note_ref:nn #1#2
  {
    \group_begin:
      \tl_set:Nn \l__postnotes_note_ref_label_tl {#2}
      \__postnotes_typeset_mark_wrapper:nnn
        {
          \bool_lazy_and:nnTF
            { ! #1 }
            { \l__postnotes_hyperlink_bool }
            {
              \hyperref [#2]
                { \__postnotes_make_mark:nnn { \ref*{#2} } { } { } }
            }
            { \__postnotes_make_mark:nnn { \ref*{#2} } { } { } }
        }
        { \tag_socket_use:n { postnotes/postnoteref/begin } }
        { \tag_socket_use:n { postnotes/postnoteref/end }   }
    \group_end:
  }
```

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

6 \postnotesection

`\postnotesection` Provide `\postnotesection`.

`\postnotesection[`⟨*options*⟩`]{`⟨*section content*⟩`}`

```
\NewDocumentCommand \postnotesection { O { } +m }
  {
    \@bsphack
    \__postnotes_section:nn {#1} {#2}
    \@esphack
  }
```

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

`\__postnotes_section:nn` The internal version of `\postnotesection`.

`\__postnotes_section:nn {`⟨*options*⟩`} {`⟨*content*⟩`}`

```
\int_new:N \g__postnotes_sectid_int
\cs_new_protected:Npn \__postnotes_section:nn #1#2
  {
    \group_begin:
      \int_gincr:N \g__postnotes_sectid_int
      \int_gincr:N \g__postnotes_note_id_int
      \seq_gput_right:Ne \g__postnotes_queue_seq { \l_postnotes_note_id_tl }
      \tl_gclear:N \g__postnotes_section_name_tl
      \keys_set:nn { postnotes/section } {#1}
      \__postnotes_set_section_page_label:e { \l_postnotes_note_id_tl }
      \bool_if:NTF \l__postnotes_section_exp_bool
```

```
        { \__postnotes_store_section:ne { \l_postnotes_note_id_tl } {#2} }
        { \__postnotes_store_section:nn { \l_postnotes_note_id_tl } {#2} }
    \group_end:
  }
```

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

Options for `\postnotesection`. Actually, I would have preferred to use "label" for the `name` option, but I feared I might need it further down the road for the traditional meaning.

```
\tl_new:N \g__postnotes_section_name_tl
\bool_new:N \l__postnotes_section_exp_bool
\keys_define:nn { postnotes/section }
  {
    name .tl_gset:N = \g__postnotes_section_name_tl ,
    name .value_required:n = true ,
    exp .bool_set:N = \l__postnotes_section_exp_bool ,
    exp .initial:n = false ,
    exp .default:n = true ,
  }
```

7 `\printpostnotes`

`\printpostnotes` Provide `\printpostnotes`.

`\printpostnotes`

```
\NewDocumentCommand \printpostnotes { }
  { \__postnotes_print_notes: }
```

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

`\pnthechapter` `\pnthesection` `\pnthechapternextnote` `\pnthesectionnextnote` `\pnthepage` `\pnidnextnote`

User facing variables, aimed at making available some of the notes' and sections' metadata for the user at specific contexts.

```
\tl_new:N \pnthechapter
\tl_new:N \pnthesection
\tl_new:N \pnidnextnote
\tl_new:N \pnthechapternextnote
\tl_new:N \pnthesectionnextnote
\tl_new:N \pnthepage
```

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

`\g__postnotes_print_postnotes_int` `\g__postnotes_print_queue_seq` `\l_postnotes_print_note_id_tl` `\l__postnotes_print_note_id_next_tl` `\l__postnotes_print_counter_tl` `\l__postnotes_print_mark_tl` `\l__postnotes_print_typeset_mark_tl` `\l__postnotes_print_type_curr_tl` `\l__postnotes_print_type_next_tl` `\l__postnotes_print_type_prev_tl` `\l__postnotes_print_content_tl` `\l__postnotes_clear_queue_seq`

Auxiliary variables for `\__postnotes_print_notes:`.

```
\int_new:N \g__postnotes_print_postnotes_int
\seq_new:N \g__postnotes_print_queue_seq
\tl_new:N \l_postnotes_print_note_id_tl
\tl_new:N \l__postnotes_print_note_id_next_tl
\tl_new:N \l__postnotes_print_counter_tl
\tl_new:N \l__postnotes_print_mark_tl
\tl_new:N \l__postnotes_print_typeset_mark_tl
\tl_new:N \l__postnotes_print_type_curr_tl
\tl_new:N \l__postnotes_print_type_next_tl
```

```
\tl_new:N \l__postnotes_print_type_prev_tl
\tl_new:N \l__postnotes_print_content_tl
\seq_new:N \l__postnotes_clear_queue_seq
```

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

`\__postnotes_print_notes:`' hooks. Both meant at providing points of entry for additional setup, specially to add support to packages and features which require it. The `postnotes/print/begin` hook is run early in `\__postnotes_print_notes:` and only once per call, after the user options have been processed. The `postnotes/print/note/begin` hook is run once for each note, at the point where environment variables are being set or restored, before the typesetting of either the mark or the text, but within a group of its own of each note.

```
\NewHook { postnotes/print/begin }
\NewHook { postnotes/print/note/begin }
```

The `postnotetext` is a counter used to restore the original value of `postnote` at the time of printing, for the purposes of cross-referencing, it should be different from `postnote` if a note may occur inside `\printpostnotes`. The `postnotesection` is a counter which is stepped for every postnote section which gets to be actually typeset. It's aim is to provide a valid "enclosing counter" to `postnote` in the context of `\printpostnotes`. Since we don't know where `postnote` may have been reset along the document, in the general case, we can't rely on any other preexisting counter. This means that the particular value of `postnotesection` is of little practical meaning, it really is just meant to provide recognizable "bounds" for `postnote` along the printing of the notes. Indeed, it is initialized to a very high value (larger than the conceivable number of postnote sections in a document), so that "marks" and "texts" don't mix in the same reference list, which would occur if the enclosing counters of both belonged to the same range, and with somewhat arbitrary results, since we cannot ensure the step of the enclosing counter along the document matches `postnotesection`. This is actually a tricky problem from the cross-referencing standpoint: two different things, which should be of the same type, are reset along the document, but shouldn't really be mixed together. They are both LaTeX 2_ε counters, since they may be required externally. Their main intended use case is to support zref-clever, but in principle they can be of general use.

```
\newcounter { postnotetext }
\newcounter { postnotesection }
\setcounter { postnotesection } { 10000 }
```

`\__postnotes_print_notes:` The internal version of `\printpostnotes`.

`\__postnotes_print_notes:`

```
\cs_new_protected:Npn \__postnotes_print_notes:
  {
    \group_begin:
      \int_gincr:N \g__postnotes_print_postnotes_int
      \__postnotes_split_labelseq:
```

We make the cut at this point. `\g__postnotes_print_queue_seq` is stored won't receive any more notes for the duration of this call of `\printpostnotes`, any notes added to the queue from this point on belong to the next call of `\printpostnotes`.

```
      \bool_lazy_and:nnTF
```

```
      { \g__postnotes_counteraux_bool }
      { ! \g__postnotes_firstrun_bool }
      {
        \seq_gset_eq:NN \g__postnotes_print_queue_seq
          \g__postnotes_print_labelseq_queue_seq
      }
      {
        \seq_gset_eq:NN \g__postnotes_print_queue_seq
          \g__postnotes_queue_seq
      }
    \seq_set_eq:NN \l__postnotes_clear_queue_seq \g__postnotes_print_queue_seq
    \seq_gclear:N \g__postnotes_queue_seq
```

The purpose of this label is to provide a point for splitting the labelseq with the `counteraux` option. It only needs to exist, it doesn't even store the page value. The `\__postnotes_set_print_page_label:e` done at `\__postnotes_set_headers_vars_first:` right below does not suffice for this purpose for two reasons. It won't be set if the queue is empty (or not yet populated), and also it comes after the heading (as it must), which means `\postnotesection`s added through hooks to it will come before it.

```
    \__postnotes_set_pre_print_label:e
      { \int_use:N \g__postnotes_print_postnotes_int }
    \seq_if_empty:NTF \g__postnotes_print_queue_seq
      { \msg_warning:nn { postnotes } { empty-printpostnotes } }
      {
        \pnheading
        \UseHook { postnotes/print/begin }
        \tl_set:Nn \l__postnotes_print_type_prev_tl { open }
        \__postnotes_check_duplicates:N \g__postnotes_print_queue_seq
        \__postnotes_sort_queue:N \g__postnotes_print_queue_seq
        \__postnotes_check_floats:N \g__postnotes_print_queue_seq
        \bool_gset_true:N \g__postnotes_header_vars_next_bool
        \__postnotes_get_headers_data:N \g__postnotes_print_queue_seq
        \__postnotes_set_headers_vars_first:
```

Ensure the first note after a heading has paragraph indentation when `listenv` is `none`. `endnotes` uses a workaroundish solution in `\enoteheading`, setting a box and then skipping back a line. Enrico Gregorio is correct, though, in criticizing it at https://tex.stackexchange.com/q/575905#comment1450213_575915, and suggests the use of `\@afterindenttrue`, which is what `indentfirst` does (we do the same, just locally).

```
        \bool_if:NF \l__postnotes_print_as_list_bool
          {
            \cs_set_eq:NN \@afterindentfalse \@afterindenttrue
            \@afterindenttrue
          }
        \bool_until_do:nn { \seq_if_empty_p:N \g__postnotes_print_queue_seq }
          {
            \seq_gpop_left:NN \g__postnotes_print_queue_seq
              \l_postnotes_print_note_id_tl
            \__postnotes_prop_get:nnN { \l_postnotes_print_note_id_tl }
              { type } \l__postnotes_print_type_curr_tl
            \tl_if_eq:NnTF \l__postnotes_print_type_curr_tl { section }
              { % type_curr = 'section'
                \seq_if_empty:NTF \g__postnotes_print_queue_seq
                  {
```

```
      \tl_set:Nn \l__postnotes_print_note_id_next_tl { noid }
      \tl_set:Nn \l__postnotes_print_type_next_tl { close }
    }
    {
      \seq_get_left:NN \g__postnotes_print_queue_seq
        \l__postnotes_print_note_id_next_tl
      \__postnotes_prop_get:nnN
        { \l__postnotes_print_note_id_next_tl }
        { type } \l__postnotes_print_type_next_tl
    }
```

We only process the entry if `type_next` is `note`: here are skipped empty sections.

```
    \tl_if_eq:NnT \l__postnotes_print_type_next_tl { note }
      {
        \stepcounter { postnotesection }
        \group_begin:
          \__postnotes_prop_get:nnN
            { \l_postnotes_print_note_id_tl }
            { thechapter } \pnthechapter
          \__postnotes_prop_get:nnN
            { \l_postnotes_print_note_id_tl }
            { thesection } \pnthesection
          \tl_set:NV \pnidnextnote
            \l__postnotes_print_note_id_next_tl
          \__postnotes_prop_get:nnN
            { \l__postnotes_print_note_id_next_tl }
            { thechapter } \pnthechapternextnote
          \__postnotes_prop_get:nnN
            { \l__postnotes_print_note_id_next_tl }
            { thesection } \pnthesectionnextnote
          \__postnotes_prop_get:nnN
            { \l_postnotes_print_note_id_tl }
            { content } \l__postnotes_print_content_tl
          \l__postnotes_print_content_tl
        \group_end:
```

Set `type_prev` for the next iteration.

```
        \tl_set:NV \l__postnotes_print_type_prev_tl
          \l__postnotes_print_type_curr_tl
      }
  }
  { % type_curr = 'note'
    \seq_if_empty:NTF \g__postnotes_print_queue_seq
      {
        \tl_set:Nn \l__postnotes_print_note_id_next_tl { noid }
        \tl_set:Nn \l__postnotes_print_type_next_tl { close }
      }
      {
        \seq_get_left:NN \g__postnotes_print_queue_seq
          \l__postnotes_print_note_id_next_tl
        \__postnotes_prop_get:nnN
          { \l__postnotes_print_note_id_next_tl }
          { type } \l__postnotes_print_type_next_tl
      }
    \tl_if_eq:NnF \l__postnotes_print_type_prev_tl { note }
```

```
      {
        \bool_if:NTF \l__postnotes_print_as_list_bool
          { \exp_args:NV \begin \l__postnotes_print_env_tl }
          { \group_begin: }
        \tag_socket_use:n { postnotes/printlist/begin }
        \l__postnotes_print_format_tl
      }
    \group_begin:
      \UseHook { postnotes/print/note/begin }
      \__postnotes_get_pageref:Ne \pnthepage
        { mark@ \l_postnotes_print_note_id_tl }
      \__postnotes_prop_get:nnN
        { \l_postnotes_print_note_id_tl }
        { mark } \l__postnotes_print_mark_tl
      \__postnotes_prop_get:nnN
        { \l_postnotes_print_note_id_tl }
        { counter } \l__postnotes_print_counter_tl
      \__postnotes_prop_get:nnN
        { \l_postnotes_print_note_id_tl }
        { content } \l__postnotes_print_content_tl
      \cs_set:Npn \@currentcounter { postnotetext }
      \int_set:Nn \c@postnotetext
        { \l__postnotes_print_counter_tl }
      \cs_set:Npe \@currentlabel
        { \p@postnote \l__postnotes_print_mark_tl }
      \tl_set:Nn \l__postnotes_print_typeset_mark_tl
        {
          \tag_socket_use:n { postnotes/printmark/begin }
          \MakeLinkTarget*
            { postnote. \l_postnotes_print_note_id_tl .text }
          \__postnotes_set_text_page_label:e
            { \l_postnotes_print_note_id_tl }
          \l__postnotes_pre_textmark_tl
          \__postnotes_typeset_text_mark:eV
            { \l_postnotes_print_note_id_tl }
            \l__postnotes_print_mark_tl
          \l__postnotes_post_textmark_tl
          \tag_socket_use:n { postnotes/printmark/end }
        }
```

Note that the placement of the tagging socket for the list case may depend on the tagging structure, in other words, on the content of the socket. It currently does nothing for the list case, so I've placed it in the "potentially most useful place". Review this if the content changes. Leave vertical mode after `\item` for the list case to avoid "perhaps a missing `\item`" error for empty notes (see `pn-bug-empty-postnote01.lvt`). And leave vertical mode before the note (and the tagging socket) for `listenv=none` (see https://github.com/gusbrs/postnotes/issues/8#issuecomment-2429501962, thanks Ulrike Fischer).

```
      \bool_if:NTF \l__postnotes_print_as_list_bool
        {
          \item
            [
              \tag_socket_use:n { postnotes/printnote/begin }
              \l__postnotes_print_typeset_mark_tl
            ]
```

```
                \mode_leave_vertical:
              }
              {
                \mode_leave_vertical:
                \tag_socket_use:n { postnotes/printnote/begin }
                \l__postnotes_print_typeset_mark_tl
              }
            \tag_socket_use:n { postnotes/printtext/begin }
            \l__postnotes_print_content_tl
            \tag_socket_use:n { postnotes/printtext/end }
            \tag_socket_use:n { postnotes/printnote/end }
            \l__postnotes_post_printnote_tl
          \group_end:
          \tl_if_eq:NnF \l__postnotes_print_type_next_tl { note }
            {
              \tag_socket_use:n { postnotes/printlist/end }
```

Ensure `\par` at the end of `\printpostnotes` (see https://github.com/u-fischer/tagpdf/issues/68#issuecomment-1587343876, thanks Ulrike Fischer).

```
              \bool_if:NTF \l__postnotes_print_as_list_bool
                {
                  \exp_args:NV \end \l__postnotes_print_env_tl
                  \par
                }
                {
                  \par
                  \group_end:
                }
            }
```

Set `type_prev` for the next iteration.

```
          \tl_set:NV \l__postnotes_print_type_prev_tl
            \l__postnotes_print_type_curr_tl
        }
      }
    \AddToHookNext { shipout/after }
      { \bool_gset_false:N \g__postnotes_header_vars_next_bool }
```

We won't use the variables anymore, clear them to reduce memory usage.

```
    \seq_map_inline:Nn \l__postnotes_clear_queue_seq
      { \__postnotes_prop_gclear:n { ##1 } }
  }
  \group_end:
}
```

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

```
\msg_new:nnn { postnotes } { empty-printpostnotes }
  { Empty~'\iow_char:N\\printpostnotes'~\msg_line_context:. }
```

`\__postnotes_typeset_text_mark:nn` Auxiliary function for mark typesetting in `\__postnotes_print_notes:`.

`\__postnotes_typeset_text_mark:nn {`⟨*note id*⟩`} {`⟨*mark*⟩`}`

```
\cs_new_protected:Npn \__postnotes_typeset_text_mark:nn #1#2
  {
    \bool_lazy_and:nnTF
      { \l__postnotes_hyperlink_bool }
      { \l__postnotes_backlink_bool  }
      {
        \__postnotes_make_text_mark:nnn {#2}
          { \hyper@linkstart { link } { postnote. #1 .mark } }
          { \hyper@linkend }
      }
      { \__postnotes_make_text_mark:nnn {#2} { } { } { } }
  }
\cs_generate_variant:Nn \__postnotes_typeset_text_mark:nn { eV }
```

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

Print auxiliary

The conditions used to split `\g__postnotes_labelseq_seq` are subtly different depending on whether we are using `\g__postnotes_counteraux_bool` or not. In the standard case, the numbering of the floats are set at "expansion time", so they belong where they are set. With `counteraux`, the numbering of floats are set at "shipout time", so they belong where they are typeset. In other words, with `counteraux` notes on floats can cross the boundaries of `\printpostnotes`, while without it, they must not. Furthermore, the purpose of `\g__postnotes_labelseq_seq` is different in each case. With `counteraux` it is used to build the actual print queue, while in the standard case it is only used in `\__postnotes_check_floats:N`. Therefore, with `counteraux` the cut is made at the place the preprint label for the current `\printpostnotes` is found, while in the standard case, `\g__postnotes_note_id_int` is used directly to distribute the elements between the current `\printpostnotes` and future ones.

`\__postnotes_split_labelseq:`

```
\seq_new:N \g__postnotes_print_labelseq_queue_seq
\cs_new_protected:Npn \__postnotes_split_labelseq:
  {
    \group_begin:
      \seq_clear:N \l__postnotes_tmpa_seq
      \seq_clear:N \l__postnotes_tmpb_seq
      \bool_if:NTF \g__postnotes_counteraux_bool
        {
          \tl_set:Ne \l__postnotes_tmpa_tl
            { { preprint } { \int_use:N \g__postnotes_print_postnotes_int } }
```

The preprint label of a `\printpostnotes` at the end of the document may not have been written: if it's empty, it may not be shipped out at all. But, since it's a counter, stepped sequentially and not floating, even if it is transitorily missing, it will be at the end. In other words, if this one is not found, there will be no subsequent preprints in the sequence.

```
          \seq_if_in:NVF \g__postnotes_labelseq_seq \l__postnotes_tmpa_tl
            { \seq_gput_right:NV \g__postnotes_labelseq_seq \l__postnotes_tmpa_tl }
          \bool_do_until:nn
            { \tl_if_eq_p:NN \l__postnotes_tmpb_tl \l__postnotes_tmpa_tl }
            {
```

```
            \seq_gpop_left:NN \g__postnotes_labelseq_seq \l__postnotes_tmpb_tl
            \str_case:enT
              { \tl_item:Nn \l__postnotes_tmpb_tl { 1 } }
              {
                { mark }    { }
                { section } { }
              }
              {
```

If the id of the labelseq item is larger than the current note id, we don't have data for the note at this point, and cannot print it. Period. Now, usually this will occur due to transitory effects. But it is theoretically possible that a float is sent to the top of the page and gets typeset before a "future `\printpostnotes`". So what we cannot print, we give back to the label sequence of the next `\printpostnotes`. If they are transitory, they will eventually go away. If they are not, better to keep them with the wrong numbering than dropping it altogether. Alas, there's nothing else we could do, short of writing the whole data to the `.aux` file, which is clearly not worth this corner case.

```
                \int_compare:nNnTF
                  { \tl_item:Nn \l__postnotes_tmpb_tl { 2 } }
                  >
                  { \g__postnotes_note_id_int }
                  {
                    \seq_put_right:Ne \l__postnotes_tmpb_seq
                      \l__postnotes_tmpb_tl
                  }
                  {
                    \seq_put_right:Ne \l__postnotes_tmpa_seq
                      { \tl_item:Nn \l__postnotes_tmpb_tl { 2 } }
                  }
              }
```

No need for the `F` branch of `\str_case:enT`, at this point the preprint of past `\printpostnotes` can no longer be here, and we don't go further than the current one. In theory, we could even go without `\str_case:enT`, but let's play safe and keep the function robust against future changes of the code.

```
            }
          \seq_gset_eq:NN \g__postnotes_print_labelseq_queue_seq
            \l__postnotes_tmpa_seq
          \seq_concat:NNN \l__postnotes_tmpa_seq \l__postnotes_tmpb_seq
            \g__postnotes_labelseq_seq
          \seq_gset_eq:NN \g__postnotes_labelseq_seq \l__postnotes_tmpa_seq
        }
        {
          \seq_map_inline:Nn \g__postnotes_labelseq_seq
            {
              \str_case:enT
                { \tl_item:nn { ##1 } { 1 } }
                {
                  { mark }    { }
                  { section } { }
                }
                {
                  \int_compare:nNnTF
                    { \tl_item:nn { ##1 } { 2 } }
```

```
                    >
                    { \g__postnotes_note_id_int }
                    { \seq_put_right:Nn \l__postnotes_tmpb_seq { ##1 } }
                    {
                      \seq_put_right:Ne \l__postnotes_tmpa_seq
                        { \tl_item:nn { ##1 } { 2 } }
                    }
                }
```

Also no need for the `F` branch here, but for a different reason. The preprint label plays no role whatsoever if `couteraux=false`, so we just drop them altogether if found.

```
            }
          \seq_gset_eq:NN \g__postnotes_print_labelseq_queue_seq
            \l__postnotes_tmpa_seq
          \seq_gset_eq:NN \g__postnotes_labelseq_seq \l__postnotes_tmpb_seq
        }
    \group_end:
  }
```

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

`\__postnotes_check_duplicates:N` provides a general procedure for handling cases of multiple passes of content. Ideally, the job should be done at `\__postnotes_inhibit_note:F`, if at all possible. But, failing that, we can rely on the fact that the labels of `\postnote`s of measuring/trial passes, being delayed `\write`s (whatsits), don't end up being written to the `.aux` file. However, despite this being a general test, and a reasonable one, I'd like to restrain it's use to the minimum possible. Using this criterion across the board could result in large swings on the content of `\printpostnotes` and spurious warnings. Hence, we only apply this check for "eligible" items. For signaling this eligibility, the note must have been stored with the `\l__postnotes_maybe_multi_bool` set to `true`, which is then saved in the `multibool` property. One implication of this procedure is that, if there are any new notes marked as `multibool`, three rounds of compilation will be needed, since the labels of the printed notes will be written only on the second run and the document will thus require a third one to stabilize.

`\__postnotes_check_duplicates:N` `\__postnotes_check_duplicates:N` ⟨`\g__postnotes_print_queue_seq`⟩

```
\cs_new_protected:Npn \__postnotes_check_duplicates:N #1
  {
```

On a first run, don't even try to check for duplicates. Better `transitorily` let some duplicates pass than to drop every legitimate note.

```
    \bool_lazy_and:nnT
      { ! \g__postnotes_firstrun_bool }
      { ! \g__postnotes_counteraux_bool }
      {
        \group_begin:
          \seq_clear:N \l__postnotes_tmpa_seq
          \seq_map_inline:Nn #1
            {
              \bool_lazy_or:nnTF
                { \cs_if_exist_p:c { \c__postnotes_ref_prefix_tl @ mark @ ##1 } }
```

Always keep sections. Empty sections are discarded anyway, and they are unlikely to occur in places performing multiple passes.

```
                {
                  \str_if_eq_p:ee
                    { \__postnotes_prop_item:nn {##1} { type } } { section }
                }
                { \seq_put_right:Nn \l__postnotes_tmpa_seq {##1} }
                {
```

If `multibool` is true for the note, we drop it silently, otherwise we include it, but warn of possible duplicate.

```
                  \str_if_eq:eeF
                    { \__postnotes_prop_item:nn {##1} { multibool } }
                    { true }
                    {
                      \seq_put_right:Nn \l__postnotes_tmpa_seq {##1}
                      \bool_if:NT \l__postnotes_check_dupli_bool
                        {
                          \msg_warning:nne { postnotes } { possible-duplicate }
                            { \__postnotes_prop_item:nn {##1} { mark } }
                        }
                    }
              }
            \seq_gset_eq:NN #1 \l__postnotes_tmpa_seq
          \group_end:
        }
    }
```

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

```
\msg_new:nnn { postnotes } { possible-duplicate }
  { Possible~duplicate~*around*~note~'#1'~\msg_line_context:. }
```

`\__postnotes_check_floats:N`

```
  \__postnotes_check_floats:N ⟨\g__postnotes_print_queue_seq⟩
```

```
\cs_new_protected:Npn \__postnotes_check_floats:N #1
  {
    \bool_lazy_and:nnT
      { \l__postnotes_check_floats_bool }
```

Ditto. In this case, the queue is not touched, but it would still be a spurious warning.

```
      { ! \g__postnotes_firstrun_bool }
      {
        \group_begin:
```

Only compare sequence net of non-existing labels.

```
          \seq_set_filter:NNn \l__postnotes_tmpa_seq #1
            {
              \bool_lazy_or_p:nn
                { \cs_if_exist_p:c { \c__postnotes_ref_prefix_tl @ mark @ ##1 } }
                { \cs_if_exist_p:c { \c__postnotes_ref_prefix_tl @ section @ ##1 } }
            }
```

Not very expl3-y, I know. But I don't see a \seq_if_eq:NNTF available and, technically, sequences are just structured token lists.

```
          \tl_if_eq:NNF
            \g__postnotes_print_labelseq_queue_seq \l__postnotes_tmpa_seq
            { \msg_warning:nn { postnotes } { possible-shuffle } }
        \group_end:
      }
  }
```

(*End of definition for* __postnotes_check_floats:N.)

```
\msg_new:nnn { postnotes } { possible-shuffle }
  { Possible~out~of~sequence~notes~due~to~floats~\msg_line_context:. }
```

__postnotes_sort_queue:N Sorting function for __postnotes_print_notes:.

__postnotes_sort_queue:N ⟨\g__postnotes_print_queue_seq⟩

```
\cs_new_protected:Npn \__postnotes_sort_queue:N #1
  {
    \bool_if:NT \l__postnotes_sort_bool
      {
        \group_begin:
          \seq_gsort:Nn #1
            {
              \__postnotes_prop_get:nnN {##1} { pnsectid } \l__postnotes_tmpa_tl
              \__postnotes_prop_get:nnN {##2} { pnsectid } \l__postnotes_tmpb_tl
              \tl_if_eq:NNTF \l__postnotes_tmpa_tl \l__postnotes_tmpb_tl
                {
                  \__postnotes_prop_get:nnN {##1} { type } \l__postnotes_tmpa_tl
                  \__postnotes_prop_get:nnN {##2} { type } \l__postnotes_tmpb_tl
                  \bool_lazy_and:nnTF
                    { \str_if_eq_p:Vn \l__postnotes_tmpa_tl { note } }
                    { \str_if_eq_p:Vn \l__postnotes_tmpb_tl { note } }
                    {
                      \__postnotes_prop_get:nnN {##1}
                        { sortnum } \l__postnotes_tmpa_tl
                      \__postnotes_prop_get:nnN {##2}
                        { sortnum } \l__postnotes_tmpb_tl
                      \fp_compare:nNnTF
                        { \l__postnotes_tmpa_tl } > { \l__postnotes_tmpb_tl }
                        { \sort_return_swapped: }
                        { \sort_return_same:    }
                    }
                    { \sort_return_same: }
                }
                { \sort_return_same: }
            }
        \group_end:
      }
  }
```

(*End of definition for* __postnotes_sort_queue:N.)

```
\AddToHook { enddocument/afterlastpage }
```

```
{
  \group_begin:
    \bool_if:NTF \g__postnotes_counteraux_bool
      {
        \seq_set_filter:NNn \l__postnotes_tmpa_seq \g__postnotes_labelseq_seq
          { \str_if_eq_p:ee { \tl_item:nn {#1} { 1 } } { mark } }
      }
      {
        \seq_set_filter:NNn \l__postnotes_tmpa_seq \g__postnotes_queue_seq
          { \str_if_eq_p:ee { \__postnotes_prop_item:nn {#1} { type } } { note } }
      }
    \seq_if_empty:NF \l__postnotes_tmpa_seq
      {
        \msg_warning:nne { postnotes } { stray-notes }
          { \seq_count:N \l__postnotes_tmpa_seq }
      }
  \group_end:
}
\msg_new:nnn { postnotes } { stray-notes }
  {
    There~are~'#1'~stray~notes~after~the~last~'\iow_char:N\\printpostnotes'~
    \msg_line_context:.~At~this~point,~they~are~lost.
  }
```

8 Headers

The headers infrastructure of postnotes is comprised of three basic parts:

1. For each `\postnote`, labels are set storing the `page` where the note occurs. Each note actually generates a pair of such labels, once when `\postnote` is called (with the mark), and another where the note is printed (in `\printpostnotes`). The former ones store `\thepage`, since we want the printed representation of it for typesetting purposes, the latter ones store the value of the `page` counter, since we don't need to typeset it, but do need to perform algebraic operations with it. These labels are set by `\__postnotes_set_mark_page_label:nn`, `\__postnotes_set_text_page_label:n`, and `\__postnotes_set_print_page_label:n` at the appropriate places. The set of these labels provides a mapping from each note's "mark" and "text" to the page where it occurs.

2. This information set is processed by `\__postnotes_get_headers_data:N` at the beginning of `\__postnotes_print_notes:` to identify the first and last note of each page in `\printpostnotes`, and to generate a mapping from these first and last notes on each page to the pages where their corresponding marks occur. We also take the opportunity to enrich this mapping with other metadata of each note. So we get also mappings from the first and last note on each page to `\thechapter`, `\thesection`, and the `name` of the section in which they occur. These mappings are stored in property lists `\g__postnotes_header_`⟨*info*⟩`_first_prop` and `\g__postnotes_header_`⟨*info*⟩`_last_prop` where the `key` is the page in `\printpostnotes` where their note's content is typeset (or rather where it starts to be typeset, it is the page where the text's mark is printed).

3. Based on these mappings, along the span of notes section we run `\__postnotes_set_headers_vars_next:` at each `shipout/before` hook to set user facing variables for the *next* page, which will be available when their heading gets typeset. Given that at `shipout` we can rely on a correct value of the `page` counter, we use it as `key` to query the property lists generated in the previous step. These user facing variables are called `\pnhd`⟨*info*⟩`first` and `\pnhd`⟨*info*⟩`last`. Since we cannot rely on the shipout hook for the first page of `\printpostnotes`, `\__postnotes_set_headers_vars_first:` is run at its beginning to ensure correct values are in place on the first page of the notes section.

These `\pnhd`⟨*info*⟩`first` and `\pnhd`⟨*info*⟩`last` variables can then be used to build simple functions which can be passed to mark commands to achieve rich contextual running headers.

`\pnhdpagefirst` `\pnhdpagelast` `\pnhdchapfirst` `\pnhdchaplast` `\pnhdsectfirst` `\pnhdsectlast` `\pnhdnamefirst` `\pnhdnamelast`

User facing variables, aimed at making available header data for the user. Setting these variables with correct values at the moment the header gets typeset is *the* objective of the whole headers infrastructure of the package.

```
\tl_new:N \pnhdpagefirst
\tl_new:N \pnhdpagelast
\tl_new:N \pnhdchapfirst
\tl_new:N \pnhdchaplast
\tl_new:N \pnhdsectfirst
\tl_new:N \pnhdsectlast
\tl_new:N \pnhdnamefirst
\tl_new:N \pnhdnamelast
```

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

`\g__postnotes_header_page_first_prop` `\g__postnotes_header_page_last_prop` `\g__postnotes_header_chap_first_prop` `\g__postnotes_header_chap_last_prop` `\g__postnotes_header_sect_first_prop` `\g__postnotes_header_sect_last_prop` `\g__postnotes_header_name_first_prop` `\g__postnotes_header_name_last_prop` `\g__postnotes_header_prev_last_page_tl` `\g__postnotes_header_prev_last_chap_tl` `\g__postnotes_header_prev_last_sect_tl` `\g__postnotes_header_prev_last_name_tl` `\l__postnotes_prev_text_page_tl` `\l__postnotes_curr_text_page_tl` `\l__postnotes_prev_mark_page_tl` `\l__postnotes_prev_mark_chap_tl` `\l__postnotes_prev_mark_sect_tl` `\l__postnotes_prev_mark_name_tl`

Auxiliary variables for the headers infrastructure.

```
\prop_new:N \g__postnotes_header_page_first_prop
\prop_new:N \g__postnotes_header_page_last_prop
\prop_new:N \g__postnotes_header_chap_first_prop
\prop_new:N \g__postnotes_header_chap_last_prop
\prop_new:N \g__postnotes_header_sect_first_prop
\prop_new:N \g__postnotes_header_sect_last_prop
\prop_new:N \g__postnotes_header_name_first_prop
\prop_new:N \g__postnotes_header_name_last_prop
\tl_new:N \g__postnotes_header_prev_last_page_tl
\tl_new:N \g__postnotes_header_prev_last_chap_tl
\tl_new:N \g__postnotes_header_prev_last_sect_tl
\tl_new:N \g__postnotes_header_prev_last_name_tl
\tl_new:N \l__postnotes_prev_text_page_tl
\tl_new:N \l__postnotes_curr_text_page_tl
\tl_new:N \l__postnotes_prev_mark_page_tl
\tl_new:N \l__postnotes_prev_mark_chap_tl
\tl_new:N \l__postnotes_prev_mark_sect_tl
\tl_new:N \l__postnotes_prev_mark_name_tl
```

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

`\__postnotes_get_headers_data:N` Process header data for `\__postnotes_set_headers_vars:n`.

`\__postnotes_get_headers_data:N` ⟨`\g__postnotes_print_queue_seq`⟩

```
\cs_new_protected:Npn \__postnotes_get_headers_data:N #1
  {
    \group_begin:
      \tl_gclear:N \pnhdpagefirst
      \tl_gclear:N \pnhdpagelast
      \tl_gclear:N \pnhdchapfirst
      \tl_gclear:N \pnhdchaplast
      \tl_gclear:N \pnhdsectfirst
      \tl_gclear:N \pnhdsectlast
      \tl_gclear:N \pnhdnamefirst
      \tl_gclear:N \pnhdnamelast
      \prop_gclear:N \g__postnotes_header_page_first_prop
      \prop_gclear:N \g__postnotes_header_page_last_prop
      \prop_gclear:N \g__postnotes_header_chap_first_prop
      \prop_gclear:N \g__postnotes_header_chap_last_prop
      \prop_gclear:N \g__postnotes_header_sect_first_prop
      \prop_gclear:N \g__postnotes_header_sect_last_prop
      \prop_gclear:N \g__postnotes_header_name_first_prop
      \prop_gclear:N \g__postnotes_header_name_last_prop
      \tl_gclear:N \g__postnotes_header_prev_last_page_tl
      \tl_gclear:N \g__postnotes_header_prev_last_chap_tl
      \tl_gclear:N \g__postnotes_header_prev_last_sect_tl
      \tl_gclear:N \g__postnotes_header_prev_last_name_tl
      \tl_clear:N \l__postnotes_prev_text_page_tl
      \tl_clear:N \l__postnotes_curr_text_page_tl
      \tl_clear:N \l__postnotes_prev_mark_page_tl
      \tl_clear:N \l__postnotes_prev_mark_chap_tl
      \tl_clear:N \l__postnotes_prev_mark_sect_tl
      \tl_clear:N \l__postnotes_prev_mark_name_tl
      \seq_map_inline:Nn #1
        {
          \exp_args:Ne \tl_if_eq:nnT
            { \__postnotes_prop_item:nn {##1} { type } }
            { note }
            {
              \__postnotes_get_pageref:Nn
                \l__postnotes_curr_text_page_tl { text@ ##1 }
              \tl_if_empty:NF \l__postnotes_curr_text_page_tl
                {
                  \tl_if_eq:NNTF
                    \l__postnotes_prev_text_page_tl
                    \l__postnotes_curr_text_page_tl
                    {
```

We are on the same page as the previous note, just update the `prev_mark` data.

```
                      \__postnotes_get_pageref:Nn
                        \l__postnotes_prev_mark_page_tl { mark@ ##1 }
                      \__postnotes_prop_get:nnN {##1} { thechapter }
                        \l__postnotes_prev_mark_chap_tl
                      \__postnotes_prop_get:nnN {##1} { thesection }
                        \l__postnotes_prev_mark_sect_tl
                      \__postnotes_prop_get:nnN {##1} { pnsectname }
                        \l__postnotes_prev_mark_name_tl
                    }
```

```
                {
```

We are on the transition between two pages, current ID is the first note of the new page (or on the very first note of `\printpostnotes`, given `\l__postnotes_prev_text_page_tl` is initialized to empty).

Set 'last' values for previous page, based on the last valid `prev_mark` stored ones. There is no previous page to the first one of `\printpostnotes`, so we don't set 'last' values for it (conditioning on `\l__postnotes_prev_text_page_tl` being empty, which only occurs on the first note).

```
              \tl_if_empty:NF \l__postnotes_prev_text_page_tl
                {
                  \prop_gput:Nee \g__postnotes_header_page_last_prop
                    { \l__postnotes_prev_text_page_tl }
                    { \l__postnotes_prev_mark_page_tl }
                  \prop_gput:Nee \g__postnotes_header_chap_last_prop
                    { \l__postnotes_prev_text_page_tl }
                    { \l__postnotes_prev_mark_chap_tl }
                  \prop_gput:Nee \g__postnotes_header_sect_last_prop
                    { \l__postnotes_prev_text_page_tl }
                    { \l__postnotes_prev_mark_sect_tl }
                  \prop_gput:Nee \g__postnotes_header_name_last_prop
                    { \l__postnotes_prev_text_page_tl }
                    { \l__postnotes_prev_mark_name_tl }
                }
```

Set 'first' values for current page, based on the current note ID.

```
              \prop_gput:Nee \g__postnotes_header_page_first_prop
                { \l__postnotes_curr_text_page_tl }
                { \__postnotes_extract_pageref:n { mark@ ##1 } }
              \prop_gput:Nee \g__postnotes_header_chap_first_prop
                { \l__postnotes_curr_text_page_tl }
                { \__postnotes_prop_item:nn {##1} { thechapter } }
              \prop_gput:Nee \g__postnotes_header_sect_first_prop
                { \l__postnotes_curr_text_page_tl }
                { \__postnotes_prop_item:nn {##1} { thesection } }
              \prop_gput:Nee \g__postnotes_header_name_first_prop
                { \l__postnotes_curr_text_page_tl }
                { \__postnotes_prop_item:nn {##1} { pnsectname } }
```

Store `prev_mark` data for the first note on the page.

```
              \__postnotes_get_pageref:Nn
                \l__postnotes_prev_mark_page_tl { mark@ ##1 }
              \__postnotes_prop_get:nnN {##1} { thechapter }
                \l__postnotes_prev_mark_chap_tl
              \__postnotes_prop_get:nnN {##1} { thesection }
                \l__postnotes_prev_mark_sect_tl
              \__postnotes_prop_get:nnN {##1} { pnsectname }
                \l__postnotes_prev_mark_name_tl
```

Set `\l__postnotes_prev_text_page_tl` for the next page (`\l__postnotes_curr_text_page_tl` is never empty at this point, since we conditioned to it).

```
              \tl_set:NV \l__postnotes_prev_text_page_tl
                \l__postnotes_curr_text_page_tl
            }
        }
```

```
            }
        }
```

We can't catch the transition from the last page of `\printpostnotes` to the following one through the mapping above, but the `prev_mark` values of the last note in the loop are the ones we want, so we set 'last' values for the last page based on them.

```
      \tl_if_empty:NF \l__postnotes_prev_text_page_tl
        {
          \prop_gput:Nee \g__postnotes_header_page_last_prop
            { \l__postnotes_prev_text_page_tl }
            { \l__postnotes_prev_mark_page_tl }
          \prop_gput:Nee \g__postnotes_header_chap_last_prop
            { \l__postnotes_prev_text_page_tl }
            { \l__postnotes_prev_mark_chap_tl }
          \prop_gput:Nee \g__postnotes_header_sect_last_prop
            { \l__postnotes_prev_text_page_tl }
            { \l__postnotes_prev_mark_sect_tl }
          \prop_gput:Nee \g__postnotes_header_name_last_prop
            { \l__postnotes_prev_text_page_tl }
            { \l__postnotes_prev_mark_name_tl }
        }
    \group_end:
  }
```

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

The sequence of pages processed in `\__postnotes_get_headers_data:N` is not ensured to be continuous, since not every page of `\printpostnotes` starts a note. There may be notes that fill whole pages, or the last page of the notes may end with a note that started on the penultimate page. We must handle this case at `\__postnotes_set_headers_vars:n`. For every page for which there is information provided by `\__postnotes_get_headers_data:N` we store a `header_prev_last` (the last value of the previous header) for each of the variables of interest. If the next page is skipped in the sequence (no notes starting on it), we can use these stored values to set both 'first' and 'last' variables based on them for that page.

`\__postnotes_set_headers_vars:n` Set user facing variables based on data generated by `\__postnotes_get_headers_data:N`.

`\__postnotes_set_headers_vars:n {`⟨*page number*⟩`}`

```
\cs_new_protected:Npn \__postnotes_set_headers_vars:n #1
  {
    \group_begin:
      \prop_get:NnNTF \g__postnotes_header_page_first_prop
        {#1} \l__postnotes_tmpa_tl
        { \tl_gset:NV \pnhdpagefirst \l__postnotes_tmpa_tl }
        { \tl_gset:NV \pnhdpagefirst \g__postnotes_header_prev_last_page_tl }
      \prop_get:NnNTF \g__postnotes_header_page_last_prop
        {#1} \l__postnotes_tmpa_tl
        {
          \tl_gset:NV \pnhdpagelast \l__postnotes_tmpa_tl
          \tl_gset:NV \g__postnotes_header_prev_last_page_tl
            \l__postnotes_tmpa_tl
        }
```

```
      { \tl_gset:NV \pnhdpagelast \g__postnotes_header_prev_last_page_tl }
    \prop_get:NnNTF \g__postnotes_header_chap_first_prop
      {#1} \l__postnotes_tmpa_tl
      { \tl_gset:NV \pnhdchapfirst \l__postnotes_tmpa_tl }
      { \tl_gset:NV \pnhdchapfirst \g__postnotes_header_prev_last_chap_tl }
    \prop_get:NnNTF \g__postnotes_header_chap_last_prop
      {#1} \l__postnotes_tmpa_tl
      {
        \tl_gset:NV \pnhdchaplast \l__postnotes_tmpa_tl
        \tl_gset:NV \g__postnotes_header_prev_last_chap_tl
          \l__postnotes_tmpa_tl
      }
      { \tl_gset:NV \pnhdchaplast \g__postnotes_header_prev_last_chap_tl }
    \prop_get:NnNTF \g__postnotes_header_sect_first_prop
      {#1} \l__postnotes_tmpa_tl
      { \tl_gset:NV \pnhdsectfirst \l__postnotes_tmpa_tl }
      { \tl_gset:NV \pnhdsectfirst \g__postnotes_header_prev_last_sect_tl }
    \prop_get:NnNTF \g__postnotes_header_sect_last_prop
      {#1} \l__postnotes_tmpa_tl
      {
        \tl_gset:NV \pnhdsectlast \l__postnotes_tmpa_tl
        \tl_gset:NV \g__postnotes_header_prev_last_sect_tl
          \l__postnotes_tmpa_tl
      }
      { \tl_gset:NV \pnhdsectlast \g__postnotes_header_prev_last_sect_tl }
    \prop_get:NnNTF \g__postnotes_header_name_first_prop
      {#1} \l__postnotes_tmpa_tl
      { \tl_gset:NV \pnhdnamefirst \l__postnotes_tmpa_tl }
      { \tl_gset:NV \pnhdnamefirst \g__postnotes_header_prev_last_name_tl }
    \prop_get:NnNTF \g__postnotes_header_name_last_prop
      {#1} \l__postnotes_tmpa_tl
      {
        \tl_gset:NV \pnhdnamelast \l__postnotes_tmpa_tl
        \tl_gset:NV \g__postnotes_header_prev_last_name_tl
          \l__postnotes_tmpa_tl
      }
      { \tl_gset:NV \pnhdnamelast \g__postnotes_header_prev_last_name_tl }
  \group_end:
}
\cs_generate_variant:Nn \__postnotes_set_headers_vars:n { e }
```

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

`\__postnotes_set_headers_vars_next:`
`\__postnotes_set_headers_vars_first:`

The functions that actually call `\__postnotes_set_headers_vars:n` at the appropriate contexts with appropriate page values. Though we set `\__postnotes_set_headers_vars_next:` to run at every `shipout/before` hook of the document, it is made no-op by `\g__postnotes_header_vars_next_bool` which only has a `true` value inside `\printpostnotes`. `\__postnotes_set_headers_vars_first:` must set a label and retrieve its value to be able to have a reliable value of its own page.

```
\AddToHook { shipout/before } [ ./header ]
  { \__postnotes_set_headers_vars_next: }
\bool_new:N \g__postnotes_header_vars_next_bool
\cs_new_protected:Npn \__postnotes_set_headers_vars_next:
```

```
  {
    \bool_if:NT \g__postnotes_header_vars_next_bool
      { \__postnotes_set_headers_vars:e { \int_eval:n { \c@page + 1 } } }
  }
\cs_new_protected:Npn \__postnotes_set_headers_vars_first:
  {
    \__postnotes_set_print_page_label:e
      { \int_use:N \g__postnotes_print_postnotes_int }
    \__postnotes_set_headers_vars:e
      {
        \__postnotes_extract_pageref:e
          { print@ \int_use:N \g__postnotes_print_postnotes_int }
      }
  }
```

(*End of definition for* `\__postnotes_set_headers_vars_next:` *and* `\__postnotes_set_headers_vars_first:`.)

`\pnheaderdefault` A basic header function to be used as default in the `heading` option. It produces a header in the form "Notes to pages N–M", with a text which can be localized (see Section 10).

`\pnheaderdefault`

```
\NewDocumentCommand \pnheaderdefault {}
  {
    \tl_if_eq:NNTF \pnhdpagefirst \pnhdpagelast
      { \pnhdnotes{} ~ \pnhdtopage{} ~ \pnhdpagefirst }
      { \pnhdnotes{} ~ \pnhdtopages{} ~ \pnhdpagefirst -- \pnhdpagelast }
  }
```

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

9 Compatibility

A dedicated temp variable for restoring data.

```
\tl_new:N \l__postnotes_restore_tmp_tl
```

`\caption`

For `\caption`'s possible two passes. This catches more than just captions, of course, but is not overkill. A hook to `\@makecaption` would be better, but ltcmdhooks does not allow it, and using lower level methods for this is a bad idea.

From the user's perspective, one-line captions will just work. For two-line captions, there are two alternatives: i) `\stepcounter{postnote}` before the caption, then call `\postnote` with `mark=\arabic{postnote}`; or ii) right before the caption, call `\postnote[nomark]{\label{mynote}...}`, then use `\postnoteref{mynote}` inside the caption.

```
\AddToHook { postnotes/note/begin } [ ./compat/caption ]
  { \cs_if_exist:NT \@captype { \postnotesetup { maybemulti } } }
```

biblatex

Thanks Moritz Wemheuer: https://tex.stackexchange.com/q/597359#comment1594585_597389.

```
\AddToHook { package/biblatex/after } [ ./compat/biblatex ]
  {
```

Let biblatex know we are in a "notes" context. See https://tex.stackexchange.com/a/304464, including comments.

```
    \AddToHook { postnotes/print/begin } [ postnotes/compat/biblatex ]
      { \toggletrue { blx@footnote } }
```

Make biblatex's `\mkbibendnote` use `\postnote`. This is very likely desired in most cases, but may occasionally not be, so we add it to an individually labeled hook, which can be disabled with `\RemoveFromHook{begindocument/before}[postnotes/mkbibendnote]` in the preamble.

```
    \AddToHook { begindocument/before } [ postnotes/compat/biblatex ]
      {
        \cs_set:Npn \blx@theendnote { \postnote }
        \cs_set:Npn \blx@theendnotetext { \blx@err@endnote \footnotetext }
      }
  }
```

⟨*gobble⟩

I had made an initial experimental attempt to support biblatex's `refsegment`s, `refcontext`s and `refsection`s. However, this attempt was rash. Even if I could get many example files to work for `refsegment`s and `refcontext`s, I could not do so for `refsection`s. More importantly, with this partial implementation, I could also generate documents which confused biblatex more than it helped. Things I couldn't understand well, or fix. All in all, I don't think this partial implementation is tenable, and I could not take it further. Hence, postnotes support for this feature set of biblatex will depend, as it should, on proper upstream support for "saving" and "restoring" citation "context" information.

I have made a feature request at biblatex for this (https://github.com/plk/biblatex/issues/1226), which was (understandably) classified as "long term, no promises".

The attempt was the following (currently "gobbled" from the package):

```
\AddToHook { package/biblatex/after } [ ./compat/biblatex ]
  {
```

Store biblatex variables for each note.

```
    \AddToHook { postnotes/note/store } [ postnotes/compat/biblatex ]
      {
        \prop_gput:cne { \__postnotes_data_name:e { \l_postnotes_note_id_tl } }
          { biblatex@refsection } { \int_use:N \c@refsection }
        \prop_gput:cne { \__postnotes_data_name:e { \l_postnotes_note_id_tl } }
          { biblatex@refsegment } { \int_use:N \c@refsegment }
        \prop_gput:cne { \__postnotes_data_name:e { \l_postnotes_note_id_tl } }
          { biblatex@refcontextbool }
          { \iftoggle { blx@refcontext } { true } { false } }
        \prop_gput:cnV { \__postnotes_data_name:e { \l_postnotes_note_id_tl } }
          { biblatex@refcontext } \blx@refcontext@context
      }
```

biblatex setup, once for \printpostnotes call.

```
\AddToHook { postnotes/print/begin } [ postnotes/compat/biblatex ]
  {
    \__postnotes_biblatex_endrefcontext_local:
    \__postnotes_biblatex_citereset_local:
  }
```

Restore biblatex variables for each note.

```
\AddToHook { postnotes/print/note/begin } [ postnotes/compat/biblatex ]
  {
    \__postnotes_prop_get:nnN { \l_postnotes_print_note_id_tl }
      { biblatex@refsection } \l__postnotes_restore_tmp_tl
    \int_set:Nn \c@refsection { \l__postnotes_restore_tmp_tl }
    \__postnotes_prop_get:nnN { \l_postnotes_print_note_id_tl }
      { biblatex@refsegment } \l__postnotes_restore_tmp_tl
    \int_set:Nn \c@refsegment { \l__postnotes_restore_tmp_tl }
    \__postnotes_prop_get:nnN { \l_postnotes_print_note_id_tl }
      { biblatex@refcontextbool } \l__postnotes_restore_tmp_tl
    \use:c { toggle \l__postnotes_restore_tmp_tl } { blx@refcontext }
    \__postnotes_prop_get:nnN { \l_postnotes_print_note_id_tl }
      { biblatex@refcontext } \l__postnotes_restore_tmp_tl
    \blx@edef@refcontext { \l__postnotes_restore_tmp_tl }
  }
```

Auxiliary functions.

__postnotes_biblatex_endrefcontext_local: Replicate the job of \endrefcontext, but with local effects, restrained to the group of \printpostnotes.

```
\cs_new_protected:Npn \__postnotes_biblatex_endrefcontext_local:
  {
    \togglefalse { blx@refcontext }
    \tl_clear:N \blx@refcontext@labelprefix
    \tl_clear:N \blx@refcontext@labelprefix@real
    \tl_set:Ne \blx@refcontext@sortingtemplatename { \blx@sorting }
    \tl_set:Nn \blx@refcontext@sortingnamekeytemplatename { global }
    \tl_set:Nn \blx@refcontext@uniquenametemplatename { global }
    \tl_set:Nn \blx@refcontext@labelalphanametemplatename { global }
    \blx@edef@refcontext
      {
        \blx@refcontext@sortingtemplatename /
        \blx@refcontext@sortingnamekeytemplatename /
        /
        \blx@refcontext@uniquenametemplatename /
        \blx@refcontext@labelalphanametemplatename
      }
  }
```

(*End of definition for* __postnotes_biblatex_endrefcontext_local:.)

__postnotes_biblatex_citereset_local: Replicate the job of \citereset, but with local effects, restrained to the group of \printpostnotes.

```
\cs_new_protected:Npn \__postnotes_biblatex_citereset_local:
  {
```

```
\global\cslet{blx@bsee@\the\c@refsection}\@empty
\global\cslet{blx@fsee@\the\c@refsection}\@empty
          \tl_clear:c { blx@bsee@ \int_use:N \c@refsection }
          \tl_clear:c { blx@fsee@ \int_use:N \c@refsection }
\blx@ibidreset@force
          \undef \blx@lastkey@text
          \undef \blx@lastkey@foot
\blx@idemreset@force
          \undef \blx@lasthash@text
          \undef \blx@lasthash@foot
\blx@opcitreset@force
          \clist_map_inline:Nn \blx@trackhash@text
            { \csundef { blx@lastkey@text@ ##1 } }
          \tl_clear:N \blx@trackhash@text
          \clist_map_inline:Nn \blx@trackhash@foot
            { \csundef { blx@lastkey@foot@ ##1 } }
          \tl_clear:N \blx@trackhash@foot
\blx@loccitreset@force
          \clist_map_inline:Nn \blx@trackkeys@text
            { \csundef { blx@lastnote@text@ ##1 } }
          \tl_clear:N \blx@trackkeys@text
          \clist_map_inline:Nn \blx@trackkeys@foot
            { \csundef { blx@lastnote@foot@ ##1 } }
          \tl_clear:N \blx@trackkeys@foot
and all of them do:
          \cs_set_eq:NN \blx@lastmpfn \z@
        }
```

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

```
  }
```

biblatex's `refsections`, contrary to `refsegment`s and `refcontext`s which are handled in the LaTeX side of things (as far as I can tell), need to go through `biber`, and must have correct corresponding citation data written to the `.bcf` file. And the way `\refsection` is implemented presumes each section is only ever begun once (fair...), thus making it difficult to "reopen" it, or append new citations to it later on, when the notes are printed. The start of a `refsection` must be registered on the `.bcf` file, and this is done by `\refsection` (and its auxiliary functions). However, a number of its characteristics make things particularly difficult for the purpose at hand: i) it unconditionally sets a label for the section which, of course, cannot be done twice; and, critically, ii) the optional argument of the environment (which receives the ⟨*resources*⟩) is used to set a local assignment to `\blx@bibfiles`, based on which the relevant information is written to the `.bcf` file, and when the group closes the information is gone. My best attempt is below but it is not good. It feels a wrong approach to "go around" the intended use of `\refsection` so much, and it can't handle at all its optional argument, for the reasons above. It's also incomplete, since it does not handle restoring `\l__postnotes_biblatex_orig_refsection_tl`.

```
\AddToHook { package/biblatex/after } [ ./compat/biblatex ]
  {
```

```
    \tl_new:N \l__postnotes_biblatex_orig_refsection_tl
    \tl_new:N \g__postnotes_biblatex_prev_refsection_tl
    \AddToHook { postnotes/print/begin } [ postnotes/compat/biblatex ]
      {
        \tl_set:Ne \l__postnotes_biblatex_orig_refsection_tl
          { \int_use:N \c@refsection }
        \tl_gset:Ne \g__postnotes_biblatex_prev_refsection_tl
          { \l__postnotes_biblatex_orig_refsection_tl }
      }
    \AddToHook { postnotes/print/note/begin } [ postnotes/compat/biblatex ]
      {
        \__postnotes_prop_get:nnN { \l_postnotes_print_note_id_tl }
          { biblatex@refsection } \l__postnotes_restore_tmp_tl
        \tl_if_eq:NNF
          \l__postnotes_restore_tmp_tl
          \g__postnotes_biblatex_prev_refsection_tl
          {
            \int_set:Nn \c@blx@maxsection
              { \l__postnotes_restore_tmp_tl - 1 }
            \tl_gset_eq:NN \g__postnotes_biblatex_prev_refsection_tl
              \l__postnotes_restore_tmp_tl
            \group_begin:
              \cs_set_eq:NN \label \use_none:n
              \cs_set_eq:NN \blx@info \use_none:n
              \blx@endrefsection
              \refsection
            \group_end:
          }
      }
  }
⟨/gobble⟩
```

zref-user

\l__postnotes_note_zlabel_str

Even though the `zlabel` option is provided only when zref-user is loaded, `\l__postnotes_note_zlabel_str` must be unconditionally defined, since it is presumed to exist by `\__postnotes_set_user_labels:` and elsewhere.

```
\str_new:N \l__postnotes_note_zlabel_str
```

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

```
\AddToHook { package/zref-user/after } [ ./compat/zref-user ]
  {
```

Provide `zlabel` option.

```
    \keys_define:nn { postnotes/note }
      {
        zlabel .str_set:N = \l__postnotes_note_zlabel_str ,
        zlabel .value_required:n = true ,
      }
```

Provide property to store the mark for measuring passes.

```
    \zref@newprop { postnote@mark } [] { \l__postnotes_mark_tl }
    \AddToHook { postnotes/note/begin }  [ postnotes/compat/zref-user ]
      { \zref@localaddprop { main } { postnote@mark } }
```

\postnotezref Provide \postnotezref.

\postnotezref⟨*⟩{⟨label⟩}

```
\NewDocumentCommand \postnotezref { s m }
  { \__postnotes_note_zref:nn {#1} {#2} }
```

(*End of definition for* \postnotezref.)

__postnotes_note_zref:nn The internal version of \postnotezref.

__postnotes_note_zref:nn {⟨star bool⟩} {⟨label⟩}

```
\tl_new:N \l__postnotes_note_zref_zlabel_tl
\cs_new_protected:Npn \__postnotes_note_zref:nn #1#2
  {
    \group_begin:
      \tl_set:Nn \l__postnotes_note_zref_zlabel_tl {#2}
      \__postnotes_typeset_mark_wrapper:nnn
        {
          \bool_lazy_all:nTF
            {
              { ! #1 }
              { \l__postnotes_hyperlink_bool }
              { \l__postnotes_zrefhyperref_bool }
            }
            {
              \hyperlink
                { \zref@extractdefault {#2} { anchor } { } }
                { \__postnotes_make_mark:nnn { \zref{#2} } { } { } }
            }
            { \__postnotes_make_mark:nnn { \zref{#2} } { } { } }
        }
        { \tag_socket_use:n { postnotes/postnotezref/begin } }
        { \tag_socket_use:n { postnotes/postnotezref/end }   }
    \group_end:
  }
```

(*End of definition for* __postnotes_note_zref:nn.)

```
}
\bool_new:N \l__postnotes_zrefhyperref_bool
\AddToHook { package/zref-hyperref/after } [ ./compat/zref-hyperref ]
  { \bool_set_true:N \l__postnotes_zrefhyperref_bool }
```

zref-clever

```
\AddToHook { package/zref-clever/after } [ ./compat/zref-clever ]
  {
    \zcsetup
      {
        countertype = { postnote = endnote } ,
        countertype = { postnotetext = endnote } ,
      }
```

```
    \AddToHook { postnotes/print/begin } [ postnotes/compat/zref-clever ]
      { \zcsetup { counterresetby = { postnotetext = postnotesection } } }
  }
```

zref-check

```
\AddToHook { package/zref-check/after } [ ./compat/zref-check ]
  {
    \IfPackageAtLeastTF { zref-check } { 2022-07-05 }
      {
        \AddToHook { postnotes/note/store } [ postnotes/compat/zref-check ]
          {
            \prop_gput:cne { \__postnotes_data_name:e { \l_postnotes_note_id_tl } }
              { zref-check@abschap } { \int_use:N \c@zc@abschap }
            \prop_gput:cne { \__postnotes_data_name:e { \l_postnotes_note_id_tl } }
              { zref-check@abssec } { \int_use:N \c@zc@abssec }
          }
        \AddToHook { postnotes/print/note/begin } [ postnotes/compat/zref-check ]
          {
            \__postnotes_prop_get:nnN { \l_postnotes_print_note_id_tl }
              { zref-check@abschap } \l__postnotes_restore_tmp_tl
            \int_set:Nn \c@zc@abschap { \l__postnotes_restore_tmp_tl }
            \__postnotes_prop_get:nnN { \l_postnotes_print_note_id_tl }
              { zref-check@abssec } \l__postnotes_restore_tmp_tl
            \int_set:Nn \c@zc@abssec { \l__postnotes_restore_tmp_tl }
          }
      }
      { }
  }
```

amsmath

```
\AddToHook { package/amsmath/after } [ ./compat/amsmath ]
  {
```

Testing for `\ifmeasuring@` is sufficient to get things right for the measuring passes in math environments.

```
    \AddToHook { postnotes/note/inhibit } [ postnotes/compat/amsmath ]
      {
        \legacy_if:nT { measuring@ }
          {
            \bool_set_true:N \l__postnotes_inhibit_note_bool
            \bool_set_true:N \l__postnotes_print_plain_mark_bool
            \bool_set_true:N \l__postnotes_print_plain_mark_stepcounter_bool
          }
      }
```

However, the `\text` macro, defined by amstext (required by amsmath), poses problems if its own. Despite my best efforts, I could not salvage things from the use of `\mathchoice` and the redefinitions of `\setcounter` and `\addtocounter` performed by amstext. Setting `maybemulti` when `firstchoice@` is `false` grants us a working situation for display style. But the use of `\postnote` inside `\text` (and, if amsmath is loaded, `\textnormal`, `\textup`, etc.) in inline math environments is not supported. If a note really needs to be there, one can use the `nomark` option and `\postnoteref`. Things should work in text mode and in display style. For some related discussion with regard to footnotes, see

https://tex.stackexchange.com/a/82820 and, in particular, Barbara Beeton's comment: "This is certainly bravura code. I do hope it doesn't result in a request to add \footnote capabilities to amsmath's multi-line display facilities. (The answer will almost certainly be no. We agree with Kopka & Daly.)"

```
    \AddToHook { postnotes/note/begin } [ postnotes/compat/amsmath ]
      { \legacy_if:nF { firstchoice@ } { \postnotesetup { maybemulti } } }
  }
```

csquotes

```
\AddToHook { package/csquotes/after } [ ./compat/csquotes ]
  {
    \bool_new:N \l__postnotes_csquotes_measuring_bool
    \BlockquoteDisable
      { \bool_set_true:N \l__postnotes_csquotes_measuring_bool }
    \AddToHook { postnotes/note/inhibit } [ postnotes/compat/csquotes ]
      {
        \bool_if:NT \l__postnotes_csquotes_measuring_bool
          {
            \bool_set_true:N \l__postnotes_inhibit_note_bool
            \bool_set_true:N \l__postnotes_print_plain_mark_bool
            \bool_set_true:N \l__postnotes_print_plain_mark_stepcounter_bool
          }
      }
  }
```

tabularx

For the identification of the trial passes in tabularx, see https://tex.stackexchange.com/a/640035 (including discussion in the comments, thanks David Carlisle), and also https://tex.stackexchange.com/a/227155 and https://tex.stackexchange.com/a/352134.

```
\AddToHook { package/tabularx/after } [ ./compat/tabularx ]
  {
    \bool_new:N \l__postnotes_tabularx_inside_env_bool
    \AddToHook { env/tabularx/begin } [ postnotes/compat/tabularx ]
      {
        \bool_set_true:N \l__postnotes_tabularx_inside_env_bool
        \cs_set_eq:NN \__postnotes_tabularx_saved_write:Nn \write
      }
    \AddToHook { postnotes/note/inhibit } [ postnotes/compat/tabularx ]
      {
        \bool_lazy_and:nnT
          { \l__postnotes_tabularx_inside_env_bool }
          { ! \cs_if_eq_p:NN \write \__postnotes_tabularx_saved_write:Nn }
          {
            \bool_set_true:N \l__postnotes_inhibit_note_bool
            \bool_set_true:N \l__postnotes_print_plain_mark_bool
            \bool_set_true:N \l__postnotes_print_plain_mark_stepcounter_bool
          }
      }
    \AddToHook { package/xltabular/after } [ postnotes/compat/xltabular ]
      {
        \AddToHook { env/xltabular/begin } [ postnotes/compat/xltabular ]
          {
```

```
            \bool_set_true:N \l__postnotes_tabularx_inside_env_bool
            \cs_set_eq:NN \__postnotes_tabularx_saved_write:Nn \write
          }
      }
  }
```

tabularray

```
\AddToHook { package/tabularray/after } [ ./compat/tabularray ]
  {
```

Since version 2023A, from 2023-03-01, tabularray offers the `\lTblrMeasuringBool` which is true when measuring and false otherwise. See https://tex.stackexchange.com/q/675818 and https://github.com/lvjr/tabularray/issues/179 (thanks Ulrike Fischer).

```
    \bool_if_exist:NT \lTblrMeasuringBool
      {
```

I'd be inclined to restrict the inhibition effect to known tabularray environments to "keep things under control". However this is a dedicated and public boolean, and users can create arbitrary new tabularray environments with `\NewTblrEnviron`, which we either wouldn't catch or have to provide an user interface for. So, for the time being, let's trust this boolean won't be misused by third-parties or users. Note that setting `\l__postnotes_print_plain_mark_stepcounter_bool` to true presumes tabularray's `counter` module is enabled. But, since this is the only way to get the measuring right in this context if there is more than one `\postnote` inside a given table, postnotes expects and requires the `counter` module.

```
        \AddToHook { postnotes/note/inhibit } [ postnotes/compat/tabularray ]
          {
            \bool_if:NT \lTblrMeasuringBool
              {
                \bool_set_true:N \l__postnotes_inhibit_note_bool
                \bool_set_true:N \l__postnotes_print_plain_mark_bool
                \bool_set_true:N \l__postnotes_print_plain_mark_stepcounter_bool
              }
          }
      }
  }
```

PDF Tagging (experimental)

Note: All of this mostly presumes `\DocumentMetadata{testphase=phase-III}` and was tested with it. For `listenv=none`, I'd expect things to work with `phase-II`, but this is only lightly tested.

A first thing to consider in tagging endnotes is how we want to represent them in the PDF structure. My first thought, for lack of another, was: emulate footnotes. There's no relevant semantic difference at the structure level between the two, and the tagging support for footnotes was done by the pros. And one distinctive characteristic the the footnotes tagging is that the footnote itself is placed in the structure as a child to the (parent) `text` element which surrounds the footnote mark. However, for endnotes this introduces a number of problems and complicates things. While footnotes "float around" and have no real structure of their own, that is not true for endnotes. Endnotes comprise a proper document section, and may be printed in a list environment, etc. So when the tagging of footnotes places the footnote structure element as a child element of the

mark's surrounding text, this is arguably for a lack of other options. Where else, after all? Indeed, a typical `html` would render footnotes at the end of the page, and not inline. True the normal `html` page is much smaller than our typical PDF, but the point stands. We can have a hover over call out for footnotes, but the same could be done for end notes, regardless of the parent-child relation (as long as the required cross-references are in place). On the other hand, for endnotes this is not an issue: they have a natural place to be plugged into. Furthermore, making an endnote a child of the text surrounding its mark leaves an empty "skeleton" of the endnotes section: the heading, the list structure, etc. Technically, we could clean that too, but clearly that's not the way to go....

Finally, the parent-child relation is not required by `PDF` standards for the relevant structure types. The `PDF` 2.0 standard (ISO 32000-2:2020), says the following about `FENote`:

> Used to markup footnotes and endnotes. Footnotes and endnotes are content that is not normally read as part of the enclosing content from which it is referenced, but rather consulted at the reading person's discretion. In order for text to be considered a footnote or endnote, there should be a reference from the enclosing content to the footnote or endnote. Such reference may be achieved by means of a Link structure element through a structure destination in its link annotation (see "Table 368 — General inline level structure types"), or use of Ref in structure elements (see "Table 355 — Entries in a structure element dictionary").

The `PDF` 1.7 standard (PDF 32000-1:2008), says the following about `Note`:

> An item of explanatory text, such as a footnote or an endnote, that is referred to from within the body of the document. It may have a label (structure type Lbl; see "List Elements" in 14.8.4.3, "Block-Level Structure Elements") as a child. The note may be included as a child of the structure element in the body text that refers to it, or it may be included elsewhere (such as in an endnotes section) and accessed by means of a reference (structure type Reference).
>
> Tagged PDF does not prescribe the placement of footnotes in the page content order. They may be either inline or at the end of the page, at the discretion of the conforming writer.

So, the note *may* be included as a child of the surrounding text, but that's not required (PDF 2.0 does not even mention that). What is required is the reference between the elements. All in all, let's not follow footnotes in establishing the parent-child relation.

Another smaller but related issue is how to treat the list structure in `\printpostnotes` when `listenv` is used. The natural thing here would be to use an `enumerate` type list (or, in PDF lingo, for the `ListNumbering` attribute to be `Ordered`), where the mark is used as the item's label (even if, technically, we use the list like a `description` in that we feed the label of every `\item` and though there is an implicit underlying counter, the list itself has no bearing upon it). The problem here is that the PDF 2.0 standards determine that the `FENote` structure element cannot be a child of a `LI` (list item). However, at least in principle, we also would like to have the `endnotelabel` element to be a child of the `endnote` element. Thus, we have a conflict, the mc-chunk can only be used once, and can be either the `Lbl` of the `LI`, or the `endnotelabel` for the `endnote`. Currently, for the list case, I'm using an `EndnotesList` class, which we define to have `ListNumbering`

as Ordered, with the mark as Lbl for LI, and letting endnote be a child of LBody. In a way, the possibility of exporting the tagged content to different formats makes me think that this is the most appropriate for the this case. For the case of listenv=none, the endnotes were made child of the Sect in which they occur. The endnotelabel was then included as part (child) of the endnote. In this, this treatment emulates the one given for footnotes in the kernel. But, at the same time, it is less than ideal for machine readability purposes, since whether the endnotelabel is part of endnote or not depends on if there is a list environment involved. Alas, I see no easy way around the PDF standard restriction for the list case.

On the LaTeX side of things, adding support for tagging entails two basic tasks: i) applying the tagging markup at the appropriate places; ii) generating references between the "mark(s)" and the "text" (plus cross-reference commands to their respective targets).

Regarding tagging references, we have three different cases: i) a regular \postnote; ii) a \postnoteref to a note labeled from inside the note; and iii) a \postnoteref to a note labeled with the label option.

For regular \postnotes it is trivial to establish the reference using the label / ref options of tagpdf.

For \postnoterefs, however labeled, the connection *must* be established at \postnoteref, for the simple fact that any \postnote can be referenced by arbitrarily many \postnoterefs. So we need to be able to retrieve the note ID from the "text" to which the reference refers to at \postnoteref. However, at \postnoteref the only information we have is the ⟨*label*⟩ but, since it is unique, we can establish a connection through it.

When the label is set from inside the note, it is actually set at \printpostnotes, at which place we have access to \l_postnotes_print_note_id_tl so we can use the ⟨*label*⟩ to pass that information around to \postnoteref. With a standard \label we must set an additional ltproperties label, using the new label hook, which \postnoteref can retrieve from its own ⟨*label*⟩ argument. For \zlabel this particular task is trivial, since we can simply add a property to store the note ID with the label, which \postnotezref can extract.

When the label is set from the option, things are slightly more complicated, because the label is set at \postnote, at which place we do not have access to the note ID of the "text". What we do here is then store the label with the note and restore it later at \printpostnotes as usual. Then, at that point, we can set an auxiliary label which can be retrieved from \postnoteref from its own ⟨*label*⟩ argument, as we did for the case of labels inside the note. Auxiliary labels are handled with ltproperties labels.

Unconditionally define tagging support sockets.

```
\socket_new:nn { tagsupport/postnotes/mark/begin }{ 0 }
\socket_new:nn { tagsupport/postnotes/mark/end }{ 0 }
\socket_new:nn { tagsupport/postnotes/nomark/begin }{ 0 }
\socket_new:nn { tagsupport/postnotes/nomark/end }{ 0 }
\socket_new:nn { tagsupport/postnotes/multisep/begin }{ 0 }
\socket_new:nn { tagsupport/postnotes/multisep/end }{ 0 }
\socket_new:nn { tagsupport/postnotes/printlist/begin }{ 0 }
\socket_new:nn { tagsupport/postnotes/printlist/end }{ 0 }
\socket_new:nn { tagsupport/postnotes/printnote/begin }{ 0 }
\socket_new:nn { tagsupport/postnotes/printnote/end }{ 0 }
\socket_new:nn { tagsupport/postnotes/printmark/begin }{ 0 }
\socket_new:nn { tagsupport/postnotes/printmark/end }{ 0 }
\socket_new:nn { tagsupport/postnotes/printtext/begin }{ 0 }
```

```
\socket_new:nn { tagsupport/postnotes/printtext/end }{ 0 }
\socket_new:nn { tagsupport/postnotes/postnoteref/begin }{ 0 }
\socket_new:nn { tagsupport/postnotes/postnoteref/end }{ 0 }
\socket_new:nn { tagsupport/postnotes/postnotezref/begin }{ 0 }
\socket_new:nn { tagsupport/postnotes/postnotezref/end }{ 0 }

\bool_lazy_and:nnT
  { \cs_if_exist_p:N \tag_if_active_p: }
  { \tag_if_active_p: }
  {
```

FIXME Review or remove these settings if/when they are included upstream (see https://github.com/latex3/tagging-project/issues/728).

```
    \tagpdfsetup
      {
        role/new-tag = { tag=endnote, role=FENote } ,
        role/new-tag = { tag=endnotemark, role=Lbl } ,
        role/new-tag = { tag=endnotelabel, role=Lbl } ,
        role/new-attribute =
          { EndnoteType } { /O /FENote /NoteType /Endnote } ,
        role/new-attribute =
          { EndnotesList } { /O /List /ListNumbering /Ordered } ,
      }
```

\postnote

```
    \socket_new_plug:nnn { tagsupport/postnotes/mark/begin } { default }
      {
        \tag_mc_end_push:
        \tag_struct_begin:n
          {
            tag = endnotemark ,
            label = { postnotemark. \l_postnotes_note_id_tl } ,
            ref = { postnote. \l_postnotes_note_id_tl } ,
          }
        \__postnotes_tagsup_store_sctructnum:nN
          { postnotemark } \l_postnotes_note_id_tl
        \tag_mc_begin:n { }
      }
    \socket_new_plug:nnn { tagsupport/postnotes/mark/end } { default }
      {
        \tag_mc_end:
        \tag_struct_end: % endnotemark
        \tag_mc_begin_pop:n { }
      }
    \socket_new_plug:nnn { tagsupport/postnotes/nomark/begin } { default }
      {
        \tag_struct_begin:n
          {
            tag = NonStruct ,
            label = { postnotemark. \l_postnotes_note_id_tl } ,
            ref = { postnote. \l_postnotes_note_id_tl } ,
          }
        \__postnotes_tagsup_store_sctructnum:nN
          { postnotemark } \l_postnotes_note_id_tl
      }
```

```
    \socket_new_plug:nnn { tagsupport/postnotes/nomark/end } { default }
      { \tag_struct_end: } % NonStruct
    \socket_assign_plug:nn { tagsupport/postnotes/mark/begin } { default }
    \socket_assign_plug:nn { tagsupport/postnotes/mark/end } { default }
    \socket_assign_plug:nn { tagsupport/postnotes/nomark/begin } { default }
    \socket_assign_plug:nn { tagsupport/postnotes/nomark/end } { default }
```

multiple

```
    \socket_new_plug:nnn { tagsupport/postnotes/multisep/begin } { default }
      {
        \tag_mc_end_push:
        \tag_mc_begin:n { artifact }
      }
    \socket_new_plug:nnn { tagsupport/postnotes/multisep/end } { default }
      {
        \tag_mc_end:
        \tag_mc_begin_pop:n { }
      }
    \socket_assign_plug:nn { tagsupport/postnotes/multisep/begin } { default }
    \socket_assign_plug:nn { tagsupport/postnotes/multisep/end } { default }
```

\printpostnotes

```
    \socket_new_plug:nnn { tagsupport/postnotes/printlist/begin } { default }
      { \tag_tool:n { para/tagging=false } }
    \socket_new_plug:nnn { tagsupport/postnotes/printlist/end } { default }
      { }
    \socket_assign_plug:nn { tagsupport/postnotes/printlist/begin } { default }
    \socket_assign_plug:nn { tagsupport/postnotes/printlist/end } { default }
    \socket_new_plug:nnn { tagsupport/postnotes/printnote/begin } { default }
      {
        \bool_if:NF \l__postnotes_print_as_list_bool
          {
            \tag_struct_begin:n
              {
                tag = endnote ,
                attribute-class = EndnoteType ,
                label = { postnote. \l_postnotes_print_note_id_tl } ,
```

CHECK Should we really add a back reference here? I couldn't find any hint about this in the standards, but latex-lab-footnotes does it. No harm, I guess.

```
                ref = { postnotemark. \l_postnotes_print_note_id_tl } ,
              }
            \__postnotes_tagsup_store_sctructnum:nN
              { postnote } \l_postnotes_print_note_id_tl
          }
      }
    \socket_new_plug:nnn { tagsupport/postnotes/printnote/end } { default }
      {
        \bool_if:NF \l__postnotes_print_as_list_bool
          { \tag_struct_end: }  % endnote
      }
    \socket_assign_plug:nn { tagsupport/postnotes/printnote/begin } { default }
    \socket_assign_plug:nn { tagsupport/postnotes/printnote/end } { default }
    \socket_new_plug:nnn { tagsupport/postnotes/printmark/begin } { default }
      {
```

```
      \bool_if:NF \l__postnotes_print_as_list_bool
        {
          \tag_struct_begin:n { tag=endnotelabel }
          \tag_mc_begin:n { tag=Lbl }
        }
    }
  \socket_new_plug:nnn { tagsupport/postnotes/printmark/end } { default }
    {
      \bool_if:NF \l__postnotes_print_as_list_bool
        {
          \tag_mc_end:
          \tag_struct_end: % endnotelabel
        }
    }
  \socket_assign_plug:nn { tagsupport/postnotes/printmark/begin } { default }
  \socket_assign_plug:nn { tagsupport/postnotes/printmark/end } { default }
  \socket_new_plug:nnn { tagsupport/postnotes/printtext/begin } { default }
    {
      \bool_if:NTF \l__postnotes_print_as_list_bool
        {
          \tag_struct_begin:n
            {
              tag = endnote ,
              attribute-class = EndnoteType ,
              label = { postnote. \l_postnotes_print_note_id_tl } ,
```

CHECK Ditto.

```
              ref = { postnotemark. \l_postnotes_print_note_id_tl } ,
            }
          \__postnotes_tagsup_store_sctructnum:nN
            { postnote } \l_postnotes_print_note_id_tl
          \tag_struct_begin:n { tag=text-unit }
          \tag_struct_begin:n { tag=text }
          \tag_tool:n { para/tagging=true }
          \tag_mc_begin:n { }
        }
        {
          \tag_struct_begin:n { tag=text-unit }
          \tag_struct_begin:n { tag=text }
          \tag_tool:n { para/tagging=true }
          \tag_mc_begin:n { }
        }
    }
  \socket_new_plug:nnn { tagsupport/postnotes/printtext/end } { default }
    {
      \bool_if:NTF \l__postnotes_print_as_list_bool
        {
          \tag_mc_end:
          \tag_tool:n { para/tagging=false }
          \tag_struct_end: % text
          \tag_struct_end: % text-unit
          \tag_struct_end: % endnote
        }
        {
          \tag_mc_end:
```

```
            \tag_tool:n { para/tagging=false }
            \tag_struct_end: % text
            \tag_struct_end: % text-unit
          }
      }
    \socket_assign_plug:nn { tagsupport/postnotes/printtext/begin } { default }
    \socket_assign_plug:nn { tagsupport/postnotes/printtext/end } { default }
```

Provide xtemplate based redefinitions of `postnoteslist` and postnoteslisthang. This is needed because, as far as I can tell, it is the only way to set `tag` and `attribute-class` for the list struct without tampering with latex-lab-testphase-block's internals.

```
    \IfInstanceExistsT { blockenv } { list }
      {
        \DeclareInstance { blockenv } { postnoteslist } { display }
          {
            env-name       = postnoteslist ,
            tag-name       = L ,
            tag-class      = EndnotesList ,
            tagging-recipe = list ,
            inner-level-counter = ,
            level-increase = true ,
            setup-code     = ,
            block-instance = list ,
            inner-instance = postnoteslist ,
          }
        \DeclareInstanceCopy { blockenv }
          { postnoteslisthang } { postnoteslist }
        \EditInstance { blockenv } { postnoteslisthang }
          { env-name = postnoteslisthang }
        \DeclareInstance { list } { postnoteslist } { std }
          { item-instance = postnoteslist }
        \DeclareInstance { item } { postnoteslist } { std }
          {
            label-format = { \hspace { \labelsep } \normalfont ~ #1 } ,
            label-align = left ,
          }
        \RenewDocumentEnvironment { postnoteslist } { }
          {
            \UseInstance { blockenv } { postnoteslist }
              {
                leftmargin     = 0pt ,
                label-width    = 0pt ,
                item-indent    = .5\parindent ,
                rightmargin    = 0pt ,
                parindent      = \parindent ,
                par-skip       = \parskip ,
                item-skip      = 0pt ,
                beginsep       = .5\topsep ,
                begin-par-skip = .5\partopsep ,
              }
          }
          { \endblockenv }
        \RenewDocumentEnvironment { postnoteslisthang } { }
          {
```

```
      \UseInstance { blockenv } { postnoteslisthang }
        {
          leftmargin     = 1em ,
          label-width    = -\leftmargin ,
          item-indent    = -2\leftmargin ,
          rightmargin    = 0pt ,
          parindent      = \parindent ,
          par-skip       = \parskip ,
          item-skip      = 0pt ,
          beginsep       = .5\topsep ,
          begin-par-skip = .5\partopsep ,
        }
    }
    { \endblockenv }
}
```

Setup for `\label` and `\zlabel` inside the note.

```
\bool_new:N \l__postnotes_inside_note_bool
\AddToHookWithArguments { label } [ postnotes/tagsup ]
  {
    \bool_if:NT \l__postnotes_inside_note_bool
      {
        \property_record:nn { postnote@label@innote. #1 }
          { postnotes/tagsup@noteid }
      }
  }
\AddToHook { postnotes/print/note/begin } [ postnotes/tagsup ]
  { \bool_set_true:N \l__postnotes_inside_note_bool }
\property_new:nnnn { postnotes/tagsup@noteid } { now } { 0 }
  { \l_postnotes_print_note_id_tl }
\AddToHook { package/zref-user/after } [ postnotes/tagsup ]
  {
    \zref@newprop { postnotes@tagsup@noteid } [ 0 ]
      { \l_postnotes_print_note_id_tl }
    \AddToHook { postnotes/print/note/begin } [ postnotes/tagsup ]
      { \zref@localaddprop { main } { postnotes@tagsup@noteid } }
  }
```

Setup for `label` and `zlabel` options.

```
\AddToHook { postnotes/note/store } [ postnotes/tagsup ]
  {
    \str_if_empty:NF \l__postnotes_note_label_str
      {
        \prop_gput:cnV
          { \__postnotes_data_name:e { \l_postnotes_note_id_tl } }
          { label } \l__postnotes_note_label_str
      }
  }
\AddToHook { package/zref-user/after } [ postnotes/tagsup ]
  {
    \AddToHook { postnotes/note/store } [ postnotes/tagsup ]
      {
        \str_if_empty:NF \l__postnotes_note_zlabel_str
          {
            \prop_gput:cnV
```

```
                { \__postnotes_data_name:e { \l_postnotes_note_id_tl } }
                { zlabel } \l__postnotes_note_zlabel_str
            }
        }
    }
  \AddToHook { postnotes/print/note/begin } [ postnotes/tagsup ]
    {
      \__postnotes_prop_get:nnN { \l_postnotes_print_note_id_tl }
        { label } \l__postnotes_restore_tmp_tl
      \tl_if_empty:NF \l__postnotes_restore_tmp_tl
        {
          \exp_args:Ne \property_record:nn
            { postnote@label@option. \l__postnotes_restore_tmp_tl }
            { postnotes/tagsup@noteid }
        }
      \__postnotes_prop_get:nnN { \l_postnotes_print_note_id_tl }
        { zlabel } \l__postnotes_restore_tmp_tl
      \tl_if_empty:NF \l__postnotes_restore_tmp_tl
        {
          \exp_args:Ne \property_record:nn
            { postnote@zlabel@option. \l__postnotes_restore_tmp_tl }
            { postnotes/tagsup@noteid }
        }
    }
```

CHECK latex-lab-footnotes creates the footnote structure element (FENote tag) and adds to it a /Ref entry pointing to the structures of *all* marks related to the note, and that includes \footrefs. I don't see anything stating something of the sort in the standards, the backref of the original mark is already a stretch. I also fail to see why this is needed, and how it could be used. But... I trust Ulrike knows better than me.

N␣␣␣␣__postnotes_tagsup_store_crossref:nN

__postnotes_tagsup_store_sctructnum:nN {⟨*ref type*⟩} {⟨*ID number of note*⟩}
__postnotes_tagsup_store_crossref:nN {⟨*ref type*⟩} {⟨*ID number of note*⟩}

⟨*ref type*⟩ is either "postnote" or "postnotemark".

```
  \prop_new:N \g__postnotes_tagsup_structnums_prop
  \cs_new_protected:Npn \__postnotes_tagsup_store_sctructnum:nN #1#2
    {
      \prop_gput:Nee \g__postnotes_tagsup_structnums_prop
        { #1 . #2 } { \tag_get:n { struct_num } }
    }
  \prop_new:N \g__postnotes_tagsup_crossrefs_prop
  \cs_new_protected:Npn \__postnotes_tagsup_store_crossref:nN #1#2
    {
      \prop_gput:Nee \g__postnotes_tagsup_crossrefs_prop
        { \tag_get:n { struct_num } } { #1 . #2 }
    }
```

(*End of definition for* __postnotes_tagsup_store_sctructnum:nN *and* __postnotes_tagsup_store_-crossref:nN.)

__postnotes_tagsup_gput_ref:nn

__postnotes_tagsup_gput_ref:nn {⟨*structnum refering from*⟩}
{⟨*structnum being referenced to*⟩}

See __fnote_gput_ref:nn.

```
  \cs_new_protected:Npn \__postnotes_tagsup_gput_ref:nn #1#2
```

```
    {
      \tag_if_active:T
        { \tag_struct_gput:ene {#1} {ref} { \tag_struct_object_ref:e {#2} } }
    }
```

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

The actual inclusion of the reference has to be done at the end, since the `ref` option called by `\tag_struct_begin:n` does not check if the variable to store the refs already exists, resulting in a clash if we add it immediately and a `\posnoteref` is made before `\printpostnotes`, and also so that the "main" reference always comes first at the list.

```
  \AddToHook { tagpdf/finish/before } [ postnotes/tagsup ]
    {
      \prop_map_inline:Nn \g__postnotes_tagsup_crossrefs_prop
        {
          \__postnotes_tagsup_gput_ref:nn
            { \prop_item:Nn \g__postnotes_tagsup_structnums_prop {#2} }
            {#1}
        }
    }
```

`\postnoteref`

```
  \socket_new_plug:nnn { tagsupport/postnotes/postnoteref/begin } { default }
    {
      \tag_mc_end_push:
      \property_if_recorded:eeTF
        { postnote@label@innote. \l__postnotes_note_ref_label_tl }
        { postnotes/tagsup@noteid }
        {
```

Label coming from a `\label` inside the note.

```
          \tl_set:Ne \l__postnotes_tmpa_tl
            {
              \property_ref:ee
                { postnote@label@innote. \l__postnotes_note_ref_label_tl }
                { postnotes/tagsup@noteid }
            }
          \tag_struct_begin:n
            {
              tag = endnotemark ,
              ref = { postnote. \l__postnotes_tmpa_tl } ,
            }
          \__postnotes_tagsup_store_crossref:nN
            { postnote } \l__postnotes_tmpa_tl
        }
        {
          \property_if_recorded:eeTF
            { postnote@label@option. \l__postnotes_note_ref_label_tl }
            { postnotes/tagsup@noteid }
            {
```

Label coming from a `label` option.

```
              \tl_set:Ne \l__postnotes_tmpa_tl
                {
                  \property_ref:ee
```

```
                    { postnote@label@option. \l__postnotes_note_ref_label_tl }
                    { postnotes/tagsup@noteid }
                }
              \tag_struct_begin:n
                {
                  tag = endnotemark ,
                  ref = { postnotemark. \l__postnotes_tmpa_tl } ,
                }
              \__postnotes_tagsup_store_crossref:nN
                { postnotemark } \l__postnotes_tmpa_tl
            }
            { \tag_struct_begin:n { tag = endnotemark } }
        }
      \tag_mc_begin:n { }
    }
  \socket_new_plug:nnn { tagsupport/postnotes/postnoteref/end } { default }
    {
      \tag_mc_end:
      \tag_struct_end: % endnotemark
      \tag_mc_begin_pop:n { }
    }
  \socket_assign_plug:nn { tagsupport/postnotes/postnoteref/begin } { default }
  \socket_assign_plug:nn { tagsupport/postnotes/postnoteref/end } { default }
```

\postnotezref

```
  \AddToHook { package/zref-user/after } [ postnotes/tagsup ]
    {
      \socket_new_plug:nnn { tagsupport/postnotes/postnotezref/begin } { default }
        {
          \tag_mc_end_push:
          \zref@ifrefcontainsprop { \l__postnotes_note_zref_zlabel_tl }
            { postnotes@tagsup@noteid }
            {
```

Label coming from a \zlabel inside the note.

```
              \tl_set:Ne \l__postnotes_tmpa_tl
                {
                  \zref@extract { \l__postnotes_note_zref_zlabel_tl }
                    { postnotes@tagsup@noteid }
                }
              \tag_struct_begin:n
                {
                  tag = endnotemark ,
                  ref = { postnote. \l__postnotes_tmpa_tl } ,
                }
              \__postnotes_tagsup_store_crossref:nN
                { postnote } \l__postnotes_tmpa_tl
            }
            {
              \property_if_recorded:eeTF
                { postnote@zlabel@option. \l__postnotes_note_zref_zlabel_tl }
                { postnotes/tagsup@noteid }
                {
```

Label coming from a zlabel option.

```
                    \tl_set:Ne \l__postnotes_tmpa_tl
                      {
                        \property_ref:ee
                          {
                            postnote@zlabel@option.
                            \l__postnotes_note_zref_zlabel_tl
                          }
                          { postnotes/tagsup@noteid }
                      }
                    \tag_struct_begin:n
                      {
                        tag = endnotemark ,
                        ref = { postnotemark. \l__postnotes_tmpa_tl } ,
                      }
                    \__postnotes_tagsup_store_crossref:nN
                      { postnotemark } \l__postnotes_tmpa_tl
                  }
                  { \tag_struct_begin:n { tag = endnotemark } }
              }
            \tag_mc_begin:n { }
          }
        \socket_new_plug:nnn { tagsupport/postnotes/postnotezref/end } { default }
          {
            \tag_mc_end:
            \tag_struct_end: % endnotemark
            \tag_mc_begin_pop:n { }
          }
        \socket_assign_plug:nn { tagsupport/postnotes/postnotezref/begin } { default }
        \socket_assign_plug:nn { tagsupport/postnotes/postnotezref/end } { default }
      }
  }
```

10 Languages

\pntitle
\pnhdnotes
\pnhdtopage
\pnhdtopages

Set of language specific user variables. They are used in the default value of the `heading` option and in `\pnheaderdefault` which, ultimately, is also used in the same place.

```
\tl_new:N \pntitle
\tl_new:N \pnhdnotes
\tl_new:N \pnhdtopage
\tl_new:N \pnhdtopages
\tl_set:Nn \pntitle { Notes }
\tl_set:Nn \pnhdnotes { Notes }
\tl_set:Nn \pnhdtopage { to~page }
\tl_set:Nn \pnhdtopages { to~pages }
```

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

__postnotes_define_language:nn

Defines language specific values for ⟨*postnote language*⟩ by storing a set of assignments for the language specific variables in ⟨*setup*⟩. ⟨*postnote language*⟩ is an internal name, typically the "main" name of the language, based on which we can set specific babel or polyglossia languages or variants.

`\__postnotes_define_language:nn {`⟨*postnote language*⟩`} {`⟨*setup*⟩`}`

```
\cs_new_protected:Npn \__postnotes_define_language:nn #1#2
  {
    \tl_new:c { g__postnotes_language_ #1 _tl }
    \tl_gset:cn { g__postnotes_language_ #1 _tl } {#2}
  }
```

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

For babel we use the new hook system, it's clean, and avoids the `\addto` pitfalls. The appropriate hook to use is `babel/`⟨*language*⟩`/beforeextras` so that users can override it with a traditional `\addto\extras`⟨*language*⟩.

Note that, for babel, the captions are currently handled in two different ways – the "old way" and the "new way" – and which of them is used depends on the language. Most still use the "old way", but the problem is that it is not universal. And the "new way" uses a different naming scheme – `\`⟨*language*⟩⟨*caption*⟩, which is meant to be set with `\setlocalecaption`, and not suitable for our needs. The `\extras`⟨*language*⟩ macros are meant for "arbitrary" code to be run when the language is selected, which is what we want. The captions used to work in the same way, but no longer for languages which use the "new way".

Note also that there seems to exist some qualms about babel's `\addto`. A number of packages define their own versions of it. Do so at least varioref (probably the original), backref, and cleveref. The latter comments that `\addto` is "flawed". babel itself comments the definition recognizing that there is an "inconsistency": depending on the case, the operation will be either local or global. This is documented in the manual, which explains this inconsistent behavior is preserved for backward compatibility, and recommends etoolbox's facilities if available. polyglossia also recommends etoolbox's `\gappto`. All in all, if there's need to use the traditional way instead of the new hooks, just rely on expl3 and use `\tl_gput_right:Nn`.

`\__postnotes_set_babel_language:nn` Sets ⟨*babel language*⟩ to execute the setup defined by `\__postnotes_define_language:nn` for ⟨*postnote language*⟩ at the `babel/`⟨*language*⟩`/beforeextras` hook.

`\__postnotes_set_babel_language:nn {`⟨*babel language*⟩`} {`⟨*postnote language*⟩`}`

```
\cs_new_protected:Npn \__postnotes_set_babel_language:nn #1#2
  {
    \ActivateGenericHook { babel/#1/beforeextras }
    \exp_args:Nnv \AddToHook { babel/#1/beforeextras }
      { g__postnotes_language_ #2 _tl }
  }
```

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

polyglossia uses a similar set of macros for setting up languages as babel does. However, the `\blockextras@`⟨*language*⟩ macros are unfortunately internal (despite what the manual says, that's what the code does), thus requiring `\makeatletter`/`\makeatother` for user configuration, which would be an inconvenience. On the other hand, polyglossia's `\captions`⟨*language*⟩ works as in babel's "old way", meaning it is just a "hook" to which we can append some code. So we use `\captions`⟨*language*⟩ for polyglossia. Things may complicate here if there's need to set up different values for different language variants, since the hooks available are all necessarily internal, but I doubt we'll ever need variants for these simple strings.

__postnotes_set_polyglossia_language:nn

Sets ⟨*polyglossia language*⟩ to execute the setup defined by `\__postnotes_define_language:nn` for ⟨*postnote language*⟩ at the polyglossia `\captions`⟨*language*⟩ hook.

```
\__postnotes_set_polyglossia_language:nn {⟨polyglossia language⟩}
  {⟨postnote language⟩}
```

```
\cs_new_protected:Npn \__postnotes_set_polyglossia_language:nn #1#2
  {
    \AddToHook { package/polyglossia/after }
      {
        \exp_args:Nnv \csgappto { captions #1 }
          { g__postnotes_language_ #2 _tl }
      }
  }
```

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

English

```
\__postnotes_define_language:nn { english }
  {
    \tl_set:Nn \pntitle     { Notes }
    \tl_set:Nn \pnhdnotes   { Notes }
    \tl_set:Nn \pnhdtopage  { to~page }
    \tl_set:Nn \pnhdtopages { to~pages }
  }
\__postnotes_set_babel_language:nn { english }    { english }
\__postnotes_set_babel_language:nn { british }    { english }
\__postnotes_set_babel_language:nn { american }   { english }
\__postnotes_set_babel_language:nn { canadian }   { english }
\__postnotes_set_babel_language:nn { australian } { english }
\__postnotes_set_babel_language:nn { newzealand } { english }
\__postnotes_set_babel_language:nn { UKenglish }  { english }
\__postnotes_set_babel_language:nn { USenglish }  { english }
\__postnotes_set_polyglossia_language:nn { english } { english }
```

Portuguese

```
\__postnotes_define_language:nn { portuguese }
  {
    \tl_set:Nn \pntitle     { Notas }
    \tl_set:Nn \pnhdnotes   { Notas }
    \tl_set:Nn \pnhdtopage  { da~página }
    \tl_set:Nn \pnhdtopages { das~páginas }
  }
\__postnotes_set_babel_language:nn { portuguese } { portuguese }
\__postnotes_set_babel_language:nn { brazilian }  { portuguese }
\__postnotes_set_babel_language:nn { portuges }   { portuguese }
\__postnotes_set_babel_language:nn { brazil }     { portuguese }
\__postnotes_set_polyglossia_language:nn { portuguese } { portuguese }
```

French

French localization validated by ‘Pika78’ at issue #1.

babel-french also has `.ldf`s for `francais`, `frenchb`, and `canadien`, but they are deprecated as options and, if used, they fall back to either `french` or `acadian`.

```
\__postnotes_define_language:nn { french }
  {
    \tl_set:Nn \pntitle     { Notes }
    \tl_set:Nn \pnhdnotes   { Notes }
    \tl_set:Nn \pnhdtopage  { de~la~page }
    \tl_set:Nn \pnhdtopages { des~pages }
  }
\__postnotes_set_babel_language:nn { french }  { french }
\__postnotes_set_babel_language:nn { acadian } { french }
\__postnotes_set_polyglossia_language:nn { french } { french }
```

German

German localization provided by Herbert Voß at issue #2.

babel-german also has `.ldf`s for `germanb` and `ngermanb`, but they are deprecated as options and, if used, they fall back respectively to `german` and `ngerman`.

```
\__postnotes_define_language:nn { german }
  {
    \tl_set:Nn \pntitle     { Endnoten }
    \tl_set:Nn \pnhdnotes   { Endnoten }
    \tl_set:Nn \pnhdtopage  { zu~Seite }
    \tl_set:Nn \pnhdtopages { zu~Seiten }
  }
\__postnotes_set_babel_language:nn { german }       { german }
\__postnotes_set_babel_language:nn { ngerman }      { german }
\__postnotes_set_babel_language:nn { austrian }     { german }
\__postnotes_set_babel_language:nn { naustrian }    { german }
\__postnotes_set_babel_language:nn { swissgerman }  { german }
\__postnotes_set_babel_language:nn { nswissgerman } { german }
\__postnotes_set_polyglossia_language:nn { german } { german }
```

⟨/package⟩

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.

R

S

T

  \blx@refcontext@labelalphanametemplatename 1457, 1464
  \blx@refcontext@labelprefix 1452
  \blx@refcontext@labelprefix@real 1453
  \blx@refcontext@sortingnamekeytemplatename 1455, 1461
  \blx@refcontext@sortingtemplatename 1454, 1460
  \blx@refcontext@uniquenametemplatename 1456, 1463
  \blx@sorting 1454
  \blx@theendnote 1410
  \blx@theendnotetext 1411
  \blx@trackhash@foot 1478, 1480
  \blx@trackhash@text 1475, 1477
  \blx@trackkeys@foot 1484, 1486
  \blx@trackkeys@text 1481, 1483
  \c@blx@maxsection 1509
  \c@page 169, 172, 1383
  \c@postnote *20*, 32, 37, 520, 623
  \c@postnotetext 924
  \c@refsection 1420, 1438, 1469, 1470, 1497
  \c@refsegment 1422, 1441
  \c@zc@abschap 1581, 1589
  \c@zc@abssec 1583, 1592
  \FN@mf@check *12*
  \FN@mf@prepare *11*, *12*
  \hyper@linkend 688, 996
  \hyper@linkstart 687, 995
  \ifmeasuring@ *48*
  \p@postnote 530, 927
  \post@note *4*, *5*, 78, 119, 140, 159
  \postnote@addtocounteraux *5*, *14*, 129, 150, 467
  \postnote@setcounteraux *5*, *14*, 124, 145, 467
  \postnotes@required@kernel 3, 4, 6, 11
  \z@ 1487
  \zref@extract 658, 2045
  \zref@extractdefault 1551
  \zref@ifrefcontainsprop 652, 2040
  \zref@ifrefundefined 649
  \zref@localaddprop 1533, 1915
  \zref@newprop 1531, 1912
\text *15–17*, *48*
\textnormal *48*
\textsuperscript *12*
\textup *48*
\thechapter *36*, 42, 62
\theHpostnote *17*
\thepage *36*, 163, 166
\thepostnote *20*, 523, 619, 626
\thesection *36*, 45, 65
tl commands:
  \c_empty_tl 188
  \tl_clear:N 72, 181, 521, 1235, 1236, 1237, 1238, 1239, 1240, 1452, 1453, 1469, 1470, 1477, 1480, 1483, 1486
  \tl_const:Nn 78, 382
  \tl_gclear:N 762, 1215, 1216, 1217, 1218, 1219, 1220, 1221, 1222, 1231, 1232, 1233, 1234
  \tl_gput_right:Nn *62*
  \tl_gset:Nn 1328, 1329, 1333, 1334, 1337, 1340, 1341, 1345, 1346, 1349, 1352, 1353, 1357, 1358, 1361, 1364, 1365, 1369, 1370, 1373, 1498, 2103
  \tl_gset_eq:NN 1511
  \tl_if_empty:NTF 512, 614, 1249, 1265, 1306, 1942, 1950
  \tl_if_eq:NNTF 1121, 1138, 1251, 1397, 1505
  \tl_if_eq:NnTF 848, 862, 902, 962
  \tl_if_eq:nnTF 224, 1243
  \tl_if_eq_p:NN 1014
  \tl_item:Nn 1018, 1025, 1034
  \tl_item:nn 1048, 1055, 1061, 1168
  \tl_new:N 16, 17, 211, 218, 285, 286, 287, 381, 384, 495, 499, 500, 569, 696, 729, 770, 782, 783, 784, 785, 786, 787, 790, 791, 792, 793, 794, 795, 796, 797, 798, 1186, 1187, 1188, 1189, 1190, 1191, 1192, 1193, 1202, 1203, 1204, 1205, 1206, 1207, 1208, 1209, 1210, 1211, 1401, 1492, 1493, 1536, 2092, 2093, 2094, 2095, 2102
  \tl_set:Nn 180, 227, 228, 232, 410, 496, 523, 547, 619, 626, 656, 671, 702, 733, 830, 852, 853, 872, 885, 892, 893, 928, 975, 1009, 1300, 1454, 1455, 1456, 1457, 1496, 1540, 1990, 2009, 2043, 2061, 2096, 2097, 2098, 2099, 2121, 2122, 2123, 2124, 2137, 2138, 2139, 2140, 2149, 2150, 2151, 2152, 2159, 2160, 2161, 2162
  \tl_set_eq:NN 534, 630
\togglefalse 1451
\toggletrue 1407
token commands:
  \token_to_str:N 118, 119, 123, 124, 128, 129, 139, 140, 144, 145, 149, 150, 159, 477, 487
\topsep 250, 267, 1875, 1891