The postnotes package
Code documentation

gusbrs

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

Version v0.2.8 – 2023-12-12

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. Finally, since we followed the move to `e`-type expansion, to play safe we require the 2023-11-01 kernel or newer.

```
\def\postnotes@required@kernel{2023-11-01}
\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} {2023-12-12} {0.2.8}
  {Endnotes for LaTeX}
```

`\l__postnotes_tmpa_tl`
`\l__postnotes_tmpb_tl`
`\l__postnotes_tmpa_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
\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`

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 name*⟩, according to the prefix set by `\c__postnotes_ref_prefix_tl`.

```
\post@note {⟨label name⟩} {⟨label content⟩}
```

```
\tl_const:Nn \c__postnotes_ref_prefix_tl { postnote@r }
\cs_new_protected:Npe \post@note #1#2
  { \exp_not:N \@newl@bel { \c__postnotes_ref_prefix_tl } {#1} {#2} }
```

(*End of definition for* `\post@note`*.*)

And ensure `\post@note` is 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 [2]{} }
      }
  }
\AddToHook { include/before }
  {
    \legacy_if:nT { @filesw }
      {
        \iow_now:Ne \@partaux
          { \token_to_str:N \providecommand \token_to_str:N \post@note [2]{} }
      }
  }
```

__postnotes_set_label:nn
__postnotes_set_mark_page_label:n
__postnotes_set_text_page_label:n
__postnotes_set_print_page_label:n

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

```
\__postnotes_set_label:nn {⟨label name⟩} {⟨value⟩}
\__postnotes_set_mark_page_label:n {⟨note id⟩}
\__postnotes_set_text_page_label:n {⟨note id⟩}
\__postnotes_set_print_page_label:n {⟨note id⟩}
```

```
\cs_new_protected:Npn \__postnotes_set_label:nn #1#2
  {
    \legacy_if:nT { @filesw }
      {
        \iow_shipout_x:Nn \@auxout
          { \token_to_str:N \post@note { #1 } { #2 } }
      }
  }
\cs_new_protected:Npn \__postnotes_set_mark_page_label:n #1
  { \__postnotes_set_label:nn { mark@ #1 } { \thepage } }
\cs_generate_variant:Nn \__postnotes_set_mark_page_label:n { e }
\cs_new_protected:Npn \__postnotes_set_text_page_label:n #1
  { \__postnotes_set_label:nn { 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:nn { print@ #1 } { \int_use:N \c@page } }
\cs_generate_variant:Nn \__postnotes_set_print_page_label:n { e }
```

(*End of definition for* `\__postnotes_set_label:nn` *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 } ,
  }
```

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 ,
  }
```

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'. }
```

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 ,
  }
```

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 }
          } ,
        pretextmark = { \par } ,
```

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 = { \par\noindent } ,
        maketextmark = { { \normalfont ##2 ##1 . ##3 } } ,
        posttextmark = { ~ } ,
      } ,
  }
```

\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.

```
326 \newcounter { postnote }
```

`\g__postnotes_note_id_int`
`\l_postnotes_note_id_tl`
`\g__postnotes_queue_seq`

`\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
```

(*End of definition for* `\g__postnotes_note_id_int`*,* `\l_postnotes_note_id_tl`*, and* `\g__postnotes_-queue_seq`*. This function is documented on page* **??***.*)

`\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}
    \__postnotes_inhibit_note:F
      {
        \int_gincr:N \g__postnotes_note_id_int
        \tl_if_empty:NT \l__postnotes_mark_tl
          {
            \stepcounter { postnote }
            \tl_set:Ne \l__postnotes_mark_tl { \thepostnote }
          }
        \seq_gput_right:Ne \g__postnotes_queue_seq
          { \l_postnotes_note_id_tl }
        \UseHook { postnotes/note/begin }
        \cs_set:Npn \@currentcounter { postnote }
        \cs_set:Npe \@currentlabel { \p@postnote \l__postnotes_mark_tl }
        \MakeLinkTarget* { postnote. \l_postnotes_note_id_tl .mark }
        \__postnotes_set_mark_page_label:e { \l_postnotes_note_id_tl }
        \__postnotes_set_user_labels:
        \bool_if:NF \l__postnotes_nomark_bool
          {
            \__postnotes_typeset_mark:eV
              { \l_postnotes_note_id_tl } \l__postnotes_mark_tl
          }
        \__postnotes_store:nn { \l_postnotes_note_id_tl } {#2}
      }
    \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
\tl_new:N \l__postnotes_note_label_tl
\bool_new:N \l__postnotes_manual_sortnum_bool
\bool_new:N \l__postnotes_maybe_multi_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 .tl_set:N = \l__postnotes_note_label_tl ,
    label .value_required: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_if:NT \l__postnotes_print_plain_mark_bool
      {
        \tl_if_empty:NT \l__postnotes_mark_tl
          {
            \bool_if:NTF \l__postnotes_print_plain_mark_stepcounter_bool
              {
                \stepcounter { postnote }
                \tl_set:Ne \l__postnotes_mark_tl { \thepostnote }
              }
              {
                \group_begin:
                \int_incr:N \c@postnote
                \exp_args:NNNe
                  \group_end:
                  \tl_set:Nn \l__postnotes_mark_tl { \thepostnote }
              }
          }
        \__postnotes_typeset_mark_wrapper:n
          { \__postnotes_make_mark:nnn { \l__postnotes_mark_tl } { } { } }
      }
    \bool_if:NTF \l__postnotes_inhibit_note_bool
      { \prg_return_true:  }
      { \prg_return_false: }
  }
```

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

`\__postnotes_typeset_mark:nn`
`\__postnotes_typeset_mark_wrapper:n`

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

```
\__postnotes_typeset_mark:nn {⟨note id⟩} {⟨mark⟩}
\__postnotes_typeset_mark_wrapper:n {⟨mark⟩}
```

```
\cs_new_protected:Npn \__postnotes_typeset_mark:nn #1#2
  {
    \__postnotes_typeset_mark_wrapper:n
      {
        \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} { } { } }
      }
  }
\cs_generate_variant:Nn \__postnotes_typeset_mark:nn { eV }
\tl_new:N \l__postnotes_saved_spacefactor_tl
\cs_new_protected:Npn \__postnotes_typeset_mark_wrapper:n #1
```

```
  {
    \mode_leave_vertical:
    \mode_if_horizontal:T
      {
        \tl_set:Ne \l__postnotes_saved_spacefactor_tl { \the\spacefactor }
        \nobreak
      }
    #1
    \mode_if_horizontal:T
      { \spacefactor \l__postnotes_saved_spacefactor_tl }
    \scan_stop:
  }
```

(*End of definition for* `\__postnotes_typeset_mark:nn` *and* `\__postnotes_typeset_mark_wrapper:n`.)

`\__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:
  {
    \tl_if_empty:NF \l__postnotes_note_label_tl
      { \exp_args:NV \label \l__postnotes_note_label_tl }
   \tl_if_empty:NF \l__postnotes_note_zlabel_tl
      { \exp_args:NV \zlabel \l__postnotes_note_zlabel_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⟩}`

```
\cs_new_protected:Npn \__postnotes_note_ref:nn #1#2
  {
    \group_begin:
    \__postnotes_typeset_mark_wrapper:n
      {
        \bool_lazy_and:nnTF
          { ! #1 }
          { \l__postnotes_hyperlink_bool }
          {
            \hyperref [#2]
              { \__postnotes_make_mark:nnn { \ref*{#2} } { } { } }
```

```
          }
          { \__postnotes_make_mark:nnn { \ref*{#2} } { } { } }
      }
    \group_end:
  }
```

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

6 \postnotesection

`\postnotesection`
`\postnotesectionx`

Provide `\postnotesection` and `\postnotesectionx`.

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

```
\NewDocumentCommand \postnotesection { O { } +m }
  { \__postnotes_section:nn {#1} {#2} }
\NewDocumentCommand \postnotesectionx { O { } +m }
  {
    % NOTE Command deprecated in 2022-12-27 for v0.2.0.
    \msg_warning:nn { postnotes } { postnotesectionx-deprecated }
    \postnotesection [ #1 , exp ] {#2}
  }
\msg_new:nnn { postnotes } { postnotesectionx-deprecated }
  {
    '\iow_char:N\\postnotesectionx'~is~deprecated~\msg_line_context:.~
    Use~the~'exp'~option~of~'\iow_char:N\\postnotesection'~instead.
  }
```

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

`\__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}
    \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
\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_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
\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_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. This function is documented on page* **??**.)

`\__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 $\LaTeX\,2_\varepsilon$ 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
    \seq_if_empty:NTF \g__postnotes_queue_seq
      { \msg_warning:nn { postnotes } { empty-printpostnotes } }
      {
        \pnheading
        \UseHook { postnotes/print/begin }
        \tl_set:Nn \l__postnotes_print_type_prev_tl { open }
        \seq_set_eq:NN \l__postnotes_clear_queue_seq \g__postnotes_queue_seq
        \__postnotes_verify_multipass:N \g__postnotes_queue_seq
        \bool_if:NT \l__postnotes_sort_bool
          { \__postnotes_sort_queue:N \g__postnotes_queue_seq }
        \bool_gset_true:N \g__postnotes_header_vars_next_bool
        \__postnotes_get_headers_data:N \g__postnotes_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_queue_seq }
        {
          \seq_gpop_left:NN \g__postnotes_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_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_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’
          \tl_if_eq:NnF \l__postnotes_print_type_prev_tl { note }
            {
              \bool_if:NTF \l__postnotes_print_as_list_bool
                { \exp_args:Ne \begin { \l__postnotes_print_env_tl } }
                { \group_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 }
          \__postnotes_text_mark_wrapper:n
            {
              \MakeLinkTarget*
                { postnote. \l_postnotes_print_note_id_tl .text }
              \__postnotes_set_text_page_label:e
                { \l_postnotes_print_note_id_tl }
              \__postnotes_typeset_text_mark:eV
                { \l_postnotes_print_note_id_tl }
                \l__postnotes_print_mark_tl
            }
          \l__postnotes_print_content_tl
          \l__postnotes_post_printnote_tl
          \group_end:
```

For notes, query for next note’s type `after` the current note was typeset, to handle possible nesting. Even if nesting is not a feature, this should avoid hard crashes related to “lonely `\item`” or “extra `\endgroup`” errors, in case it occurs.

```
          \seq_if_empty:NTF \g__postnotes_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_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_next_tl { note }
                          {
                            \bool_if:NTF \l__postnotes_print_as_list_bool
                              { \exp_args:Ne \end { \l__postnotes_print_env_tl } }
                              { \group_end: }
```

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

```
                            \par
                          }
```

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. Given how we populated `\l__postnotes_clear_queue_seq`, this won't catch nested notes. But it's not worth to conditionally add new items along the way (testing it every iteration) for this. Again, not a feature.

```
        \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`
`\__postnotes_text_mark_wrapper:n`

Auxiliary functions for mark typesetting in `\__postnotes_print_notes:`.

`\__postnotes_typeset_text_mark:nn {⟨note id⟩} {⟨mark⟩}`
`\__postnotes_text_mark_wrapper:n {⟨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 }
\cs_new_protected:Npn \__postnotes_text_mark_wrapper:n #1
  {
    \bool_if:NTF \l__postnotes_print_as_list_bool
      {
        \item [ \l__postnotes_pre_textmark_tl #1 \l__postnotes_post_textmark_tl ]
```

Leave vertical mode to avoid "perhaps a missing `\item`" error for empty notes.

```
        \mode_leave_vertical:
      }
      { \l__postnotes_pre_textmark_tl #1 \l__postnotes_post_textmark_tl }
  }
```

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

Print auxiliary

`\__postnotes_verify_multipass: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 `\postnotes` of measuring/trial passes don't end up being output and hence don't generate labels in the `.aux` file. This is the equivalent for postnotes to the effect of write restrictions for the packages based on external files, which is how they actually handle these cases. However, despite this being a general test, and a reasonable one, I'd like to restrain it's use to the minimum possible. First, using this criterion across the board would result in large swings on the content of `\printpostnotes` and spurious warnings in an initial compilation since the labels are not available on the first run. Second, I'd prefer not to interfere with the queue, unless we really need to. 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_verify_multipass:N`

`\__postnotes_verify_multipass:N` ⟨`\g__postnotes_queue_seq`⟩

```
\cs_new_protected:Npn \__postnotes_verify_multipass:N #1
  {
    \group_begin:
    \seq_clear:N \l__postnotes_tmpa_seq
    \seq_map_inline:Nn #1
      {
        \__postnotes_prop_get:nnN {##1} { multibool } \l__postnotes_tmpa_tl
        \str_if_eq:VnTF \l__postnotes_tmpa_tl { true }
          {
            \cs_if_exist:cT
              { \c__postnotes_ref_prefix_tl @ mark@ ##1 }
              { \seq_put_right:Nn \l__postnotes_tmpa_seq {##1} }
          }
          { \seq_put_right:Nn \l__postnotes_tmpa_seq {##1} }
      }
    \seq_gset_eq:NN #1 \l__postnotes_tmpa_seq
```

```
    \group_end:
  }
```

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

`\__postnotes_sort_queue:N` Sorting function for `\__postnotes_print_notes:`.

`\__postnotes_sort_queue:N` ⟨`\g__postnotes_queue_seq`⟩

```
\cs_new_protected:Npn \__postnotes_sort_queue:N #1
  {
    \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`.)

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:n`, `\__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_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 } [ postnotes/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.

From the user's perspective, one-line captions will just work. For two-line captions, there are two alternatives: i) decrement the counter by 1 \addtocounter{postnote}{-1} before the caption, then call \postnote inside the caption; or ii) right before the caption, call \postnote[nomark]{\label{mynote}...}, then use \postnoteref{mynote} inside the caption.

```
\AddToHook { postnotes/note/begin } [ postnotes ]
  {
    \cs_if_exist:NT \@captype
      { \bool_set_true:N \l__postnotes_maybe_multi_bool }
  }
```

biblatex

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

```
\AddToHook { package/biblatex/after }
  {
```

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

```
    \AddToHook { postnotes/print/begin } [ postnotes ]
      { \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/mkbibendnote ]
      {
        \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 refsegments, refcontexts and refsections. However, this attempt was rash. Even if I could get many example files to work for refsegments and refcontexts, I could not do so for refsections. 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 }
  {
```

Store biblatex variables for each note.

```
    \AddToHook { postnotes/note/store } [ postnotes ]
      {
        \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 ]
      {
        \__postnotes_biblatex_endrefcontext_local:
        \__postnotes_biblatex_citereset_local:
      }
```

Restore biblatex variables for each note.

```
    \AddToHook { postnotes/print/note/begin } [ postnotes ]
      {
        \__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 `refsegments` and `refcontexts` 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 }
  {
    \tl_new:N \l__postnotes_biblatex_orig_refsection_tl
    \tl_new:N \g__postnotes_biblatex_prev_refsection_tl
    \AddToHook { postnotes/print/begin } [ postnotes ]
      {
        \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 ]
      {
        \__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_tl

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

```
\tl_new:N \l__postnotes_note_zlabel_tl
```

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

```
\AddToHook { package/zref-user/after }
  {
```

Provide `zlabel` option.

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

\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⟩}`

```
    \cs_new_protected:Npn \__postnotes_note_zref:nn #1#2
      {
        \group_begin:
        \__postnotes_typeset_mark_wrapper:n
          {
            \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} } { } { } }
          }
        \group_end:
      }
```

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

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

zref-clever

```
\AddToHook { package/zref-clever/after }
  {
    \zcsetup
      {
        countertype = { postnote = endnote } ,
        countertype = { postnotetext = endnote } ,
      }
    \AddToHook { postnotes/print/begin } [ postnotes ]
      { \zcsetup { counterresetby = { postnotetext = postnotesection } } }
  }
```

zref-check

```
\AddToHook { package/zref-check/after }
  {
    \IfPackageAtLeastTF { zref-check } { 2022-07-05 }
      {
        \AddToHook { postnotes/note/store } [ postnotes ]
          {
            \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 ]
          {
            \__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 }
  {
```

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

```
    \AddToHook { postnotes/note/inhibit } [ postnotes ]
      {
        \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 \l__postnotes_maybe_multi_bool 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 ]
      {
        \legacy_if:nF { firstchoice@ }
          { \bool_set_true:N \l__postnotes_maybe_multi_bool }
      }
  }
```

csquotes

```
\AddToHook { package/csquotes/after }
  {
    \bool_new:N \l__postnotes_csquotes_measuring_bool
    \BlockquoteDisable
      { \bool_set_true:N \l__postnotes_csquotes_measuring_bool }
    \AddToHook { postnotes/note/inhibit } [ postnotes ]
      {
        \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 }
  {
    \bool_new:N \l__postnotes_tabularx_inside_env_bool
    \AddToHook { env/tabularx/begin } [ postnotes ]
      {
        \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 ]
      {
        \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
          }
      }
  }
```

tabularray

```
\AddToHook { package/tabularray/after }
  {
```

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:NTF \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, pkgpostnotes expects and requires the `counter` module.

```
        \AddToHook { postnotes/note/inhibit } [ postnotes ]
          {
            \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
              }
          }
      }
      {
```

If the new boolean is not yet available, we use `\__postnotes_verify_multipass:N` to distinguish a trial/measure pass from the final one.

```
        \clist_map_inline:nn
          {
            tblr , longtblr , talltblr , booktabs ,
            longtabs , talltabs , +array
          }
          {
            \AddToHook { env/#1/begin } [ postnotes ]
              { \bool_set_true:N \l__postnotes_maybe_multi_bool }
          }
      }
```

```
  }
```

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 .ldfs 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 .ldfs 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.

U

W

Z