

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	2
2	Data	2
3	Options	5
4	<code>\postnote</code>	10
5	<code>\postnoteref</code>	16
6	<code>\postnotesection</code>	17
7	<code>\printpostnotes</code>	18
8	Headers	24
9	Compatibility	30
10	Languages	39
	Index	42

1 Initial setup

Start the DocStrip guards.

```
1 <*package>
   Identify the internal prefix (LATEX3 DocStrip convention).
2 <@@=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.

```
3 \def\postnotes@required@kernel{2023-11-01}
4 \NeedsTeXFormat{LaTeX2e}[\postnotes@required@kernel]
5 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
6 \IfFormatAtLeastTF{\postnotes@required@kernel}
7   {}
8   {%
9     \PackageError{postnotes}{LaTeX kernel too old}
10    {%
11      'postnotes' requires a LaTeX kernel \postnotes@required@kernel\space or newer.%
12    }%
13  }%
14 \ProvidesExplPackage {postnotes} {2023-12-12} {0.2.8}
15 {Endnotes for LaTeX}
```

\l__postnotes_tmpa_tl Temporary scratch variables.

```
\l__postnotes_tmpb_tl 16 \tl_new:N \l__postnotes_tmpa_tl
\l__postnotes_tmpa_seq 17 \tl_new:N \l__postnotes_tmpb_tl
\l__postnotes_tmpa_box 18 \seq_new:N \l__postnotes_tmpa_seq
19 \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>}
20 \cs_new:Npn \__postnotes_data_name:n #1
21   { g__postnotes_ #1 _data_prop }
22 \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>}

23 \NewHook { postnotes/note/store }
24 \cs_new_protected:Npn \__postnotes_store:nn #1#2
25 {
26   \prop_new:c { \__postnotes_data_name:e {#1} }
27   \prop_gput:cnn { \__postnotes_data_name:e {#1} } { type } { note }
28   \prop_gput:cne { \__postnotes_data_name:e {#1} } { mark }
29     { \l__postnotes_mark_tl }
30   \prop_gput:cne { \__postnotes_data_name:e {#1} } { counter }
31     { \int_use:N \c@postnote }
32   \prop_gput:cne { \__postnotes_data_name:e {#1} } { sortnum }
33     {
34       \bool_if:NTF \l__postnotes_manual_sortnum_bool
35         { \fp_use:N \l__postnotes_sort_num_fp }
36         { \int_use:N \c@postnote }
37     }
38   \cs_if_exist:cT { chapter }
39     {
40       \prop_gput:cne { \__postnotes_data_name:e {#1} }
41         { thechapter } { \thechapter }
42     }
43   \prop_gput:cne { \__postnotes_data_name:e {#1} } { thesection }
44     { \thesection }
45   \prop_gput:cne { \__postnotes_data_name:e {#1} } { pnsectname }
46     { \g__postnotes_section_name_tl }
47   \prop_gput:cne { \__postnotes_data_name:e {#1} } { pnsectid }
48     { \int_use:N \g__postnotes_sectid_int }
49   \prop_gput:cne { \__postnotes_data_name:e {#1} } { multibool }
50     { \bool_to_str:N \l__postnotes_maybe_multi_bool }
51   \prop_gput:cnn { \__postnotes_data_name:e {#1} } { content } {#2}
52   \UseHook { postnotes/note/store }
53 }

```

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

`__postnotes_store_section:nn` Stores the metadata and *<note content>* of `\postnotessection` with ID *<note id>*, from where it is called.

```

    \__postnotes_store_section:nn {<note id>} {<note content>}

54 \cs_new_protected:Npn \__postnotes_store_section:nn #1#2
55 {
56   \prop_new:c { \__postnotes_data_name:e {#1} }
57   \prop_gput:cnn { \__postnotes_data_name:e {#1} } { type } { section }
58   \cs_if_exist:cT { chapter }
59     {
60       \prop_gput:cne { \__postnotes_data_name:e {#1} }
61         { thechapter } { \thechapter }
62     }
63   \prop_gput:cne { \__postnotes_data_name:e {#1} } { thesection }

```

```

64     { \thesection }
65     \prop_gput:cnN { \__postnotes_data_name:e {#1} } { content } {#2}
66   }
67   \cs_generate_variant:Nn \__postnotes_store_section:nn { ne }

```

(End of definition for __postnotes_store_section:nn.)

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

```

\__postnotes_prop_item:nn
\__postnotes_prop_gclear:n
\__postnotes_prop_get:nnN {<note id>} {<property>} {<tl var to set>}
\__postnotes_prop_item:nn {<note id>} {<property>}
\__postnotes_prop_gclear:n {<note id>}
68 \cs_new_protected:Npn \__postnotes_prop_get:nnN #1#2#3
69   {
70     \prop_get:cnNF { \__postnotes_data_name:e {#1} } {#2} #3
71     { \tl_clear:N #3 }
72   }
73 \cs_new:Npn \__postnotes_prop_item:nn #1#2
74   { \prop_item:cn { \__postnotes_data_name:e {#1} } {#2} }
75 \cs_new_protected:Npn \__postnotes_prop_gclear:n #1
76   { \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 L^AT_EX 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>}
77 \tl_const:Nn \c__postnotes_ref_prefix_tl { postnote@r }
78 \cs_new_protected:Npe \post@note #1#2
79   { \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.

```

80 \AddToHook { begindocument }
81   {
82     \legacy_if:nT { @files }
83     {
84       \iow_now:Ne \@mainaux
85       { \token_to_str:N \providecommand \token_to_str:N \post@note [2]{ } }
86     }
87   }
88 \AddToHook { include/before }
89   {
90     \legacy_if:nT { @files }
91     {
92       \iow_now:Ne \@partaux
93       { \token_to_str:N \providecommand \token_to_str:N \post@note [2]{ } }
94     }
95   }

```

`__postnotes_set_label:nn` 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_mark_page_label:n
\__postnotes_set_text_page_label:n
\__postnotes_set_print_page_label:n

\__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>}

96 \cs_new_protected:Npn \__postnotes_set_label:nn #1#2
97   {
98     \legacy_if:nT { @filesw }
99     {
100       \iow_shipout_x:Nn \@auxout
101       { \token_to_str:N \post@note { #1 } { #2 } }
102     }
103   }
104 \cs_new_protected:Npn \__postnotes_set_mark_page_label:n #1
105   { \__postnotes_set_label:nn { mark@ #1 } { \thepage } }
106 \cs_generate_variant:Nn \__postnotes_set_mark_page_label:n { e }
107 \cs_new_protected:Npn \__postnotes_set_text_page_label:n #1
108   { \__postnotes_set_label:nn { text@ #1 } { \int_use:N \c@page } }
109 \cs_generate_variant:Nn \__postnotes_set_text_page_label:n { e }
110 \cs_new_protected:Npn \__postnotes_set_print_page_label:n #1
111   { \__postnotes_set_label:nn { print@ #1 } { \int_use:N \c@page } }
112 \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` Reference data extraction functions.

```

\__postnotes_extract_pageref:n

\__postnotes_get_pageref:Nn {<tl var to set>} {<label name>}
\__postnotes_extract_pageref:n {<label name>}

113 \cs_new_protected:Npn \__postnotes_get_pageref:Nn #1#2
114   {
115     \cs_if_exist:cTF { \c__postnotes_ref_prefix_tl @ #2 }
116     { \tl_set:Nv #1 { \c__postnotes_ref_prefix_tl @ #2 } }
117     { \tl_clear:N #1 }
118   }
119 \cs_generate_variant:Nn \__postnotes_get_pageref:Nn { Ne }
120 \cs_new:Npn \__postnotes_extract_pageref:n #1
121   {
122     \cs_if_exist:cTF { \c__postnotes_ref_prefix_tl @ #1 }
123     { \exp_not:v { \c__postnotes_ref_prefix_tl @ #1 } }
124     { \c_empty_tl }
125   }
126 \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

```

127 \keys_define:nn { postnotes/setup }
128 {
129   heading .cs_set_protected:Np = \pnheading ,
130   heading .value_required:n = true ,
131 }

```

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

```

132 \cs_if_exist:cTF { chapter }
133 {
134   \cs_new_protected:Npn \pnheading
135   {
136     \chapter*{\pntitle}
137     \@mkboth{\pnheaderdefault}{\pnheaderdefault}
138   }
139 }
140 {
141   \cs_new_protected:Npn \pnheading
142   {
143     \section*{\pntitle}
144     \@mkboth{\pnheaderdefault}{\pnheaderdefault}
145   }
146 }

```

(End of definition for `\pnheading`.)

format option

```

147 \tl_new:N \l__postnotes_print_format_tl
148 \keys_define:nn { postnotes/setup }
149 {
150   format .tl_set:N = \l__postnotes_print_format_tl ,
151   format .initial:n = { \small } ,
152   format .value_required:n = true ,
153 }

```

listenv option

```

154 \tl_new:N \l__postnotes_print_env_tl
155 \bool_new:N \l__postnotes_print_as_list_bool
156 \keys_define:nn { postnotes/setup }
157 {
158   listenv .code:n =
159   {
160     \tl_if_eq:nnTF {#1} { none }
161     {
162       \bool_set_false:N \l__postnotes_print_as_list_bool
163       \tl_set:Nn \l__postnotes_post_printnote_tl { \par }

```

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

```

164       \tl_set:Nn \l__postnotes_print_env_tl { itemize }
165     }
166     {
167       \bool_set_true:N \l__postnotes_print_as_list_bool
168       \tl_set:Nn \l__postnotes_print_env_tl {#1}
169     }

```

```

170     } ,
171     listenv .initial:n = { postnoteslist } ,
172     listenv .value_required:n = true ,
173 }

```

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 L^AT_EX Companion*).

```

174 \NewDocumentEnvironment { postnoteslist } { }
175 {
176   \list { }
177   {
178     \setlength { \leftmargin } { Opt }
179     \setlength { \labelwidth } { Opt }
180     \setlength { \itemindent } { .5\parindent }
181     \cs_set_eq:NN \makelabel \__postnotes_list_makelabel:n
182     \setlength { \rightmargin } { Opt }
183     \setlength { \listparindent } { \parindent }
184     \setlength { \parsep } { \parskip }
185     \setlength { \itemsep } { Opt }
186     \setlength { \topsep } { .5\topsep }
187     \setlength { \partopsep } { .5\partopsep }
188   }
189 }
190 { \endlist }
191 \NewDocumentEnvironment { postnoteslisthang } { }
192 {
193   \list { }
194   {
195     \setlength { \leftmargin } { 1em }
196     \setlength { \labelwidth } { -\leftmargin }
197     \setlength { \itemindent } { -2\leftmargin }
198     \cs_set_eq:NN \makelabel \__postnotes_list_makelabel:n
199     \setlength { \rightmargin } { Opt }
200     \setlength { \listparindent } { \parindent }
201     \setlength { \parsep } { \parskip }
202     \setlength { \itemsep } { Opt }
203     \setlength { \topsep } { .5\topsep }
204     \setlength { \partopsep } { .5\partopsep }
205   }
206 }
207 { \endlist }
208 \cs_new:Npn \__postnotes_list_makelabel:n #1
209 { \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.

```

210 \keys_define:nn { postnotes/setup }
211 {
212   makemark .cs_set:Np = \__postnotes_make_mark:nnn #1#2#3 ,
213   makemark .value_required:n = true ,

```

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

```

214 makemark .initial:n =
215   { #2 \hbox { \@textsuperscript { \normalfont #1 } } #3 } ,
216 maketextmark .cs_set:Np = \__postnotes_make_text_mark:nnn #1#2#3 ,
217 maketextmark .value_required:n = true ,
218 maketextmark .initial:n = { #2 #1 . #3 } ,
219 }

```

pretextmark, posttextmark, postprintnote options

```

220 \tl_new:N \l__postnotes_pre_textmark_tl
221 \tl_new:N \l__postnotes_post_textmark_tl
222 \tl_new:N \l__postnotes_post_printnote_tl
223 \keys_define:nn { postnotes/setup }
224 {
225   pretextmark .tl_set:N = \l__postnotes_pre_textmark_tl ,
226   pretextmark .value_required:n = true ,
227   posttextmark .tl_set:N = \l__postnotes_post_textmark_tl ,
228   posttextmark .value_required:n = true ,
229   postprintnote .tl_set:N = \l__postnotes_post_printnote_tl ,
230   postprintnote .value_required:n = true ,
231 }

```

hyperref and backlink options

```

232 \bool_new:N \l__postnotes_hyperlink_bool
233 \bool_new:N \l__postnotes_hyperref_warn_bool
234 \bool_new:N \l__postnotes_backlink_bool
235 \keys_define:nn { postnotes/setup }
236 {
237   hyperref .choice: ,
238   hyperref / auto .code:n =
239     {
240       \bool_set_true:N \l__postnotes_hyperlink_bool
241       \bool_set_false:N \l__postnotes_hyperref_warn_bool
242     } ,
243   hyperref / true .code:n =
244     {
245       \bool_set_true:N \l__postnotes_hyperlink_bool
246       \bool_set_true:N \l__postnotes_hyperref_warn_bool
247     } ,
248   hyperref / false .code:n =
249     {
250       \bool_set_false:N \l__postnotes_hyperlink_bool
251       \bool_set_false:N \l__postnotes_hyperref_warn_bool
252     } ,
253   hyperref .initial:n = auto ,
254   hyperref .default:n = true ,
255   backlink .bool_set:N = \l__postnotes_backlink_bool ,
256   backlink .initial:n = true ,
257   backlink .default:n = true ,
258 }
259 \AddToHook { begindocument }
260 {
261   \IfPackageLoadedTF { hyperref }

```

```

262     { }
263     {
264         \bool_if:NT \l__postnotes_hyperref_warn_bool
265             { \msg_warning:nn { postnotes } { missing-hyperref } }
266         \bool_set_false:N \l__postnotes_hyperlink_bool
267     }
268     \keys_define:nn { postnotes/setup }
269     {
270         hyperref .code:n =
271         {
272             \msg_warning:nnn { postnotes }
273                 { option-preamble-only } { hyperref }
274         } ,
275         backlink .code:n =
276         {
277             \msg_warning:nnn { postnotes }
278                 { option-preamble-only } { backlink }
279         } ,
280     }
281 }
282 \msg_new:nnn { postnotes } { option-preamble-only }
283 { Option~'#1'~only~available~in~the~preamble~\msg_line_context:. }
284 \msg_new:nnn { postnotes } { missing-hyperref }
285 { Missing~'hyperref'~package.~Setting~'hyperref=false'. }

```

sort option

```

286 \bool_new:N \l__postnotes_sort_bool
287 \keys_define:nn { postnotes/setup }
288 {
289     sort .bool_set:N = \l__postnotes_sort_bool ,
290     sort .initial:n = true ,
291     sort .default:n = true ,
292 }

```

style option

```

293 \keys_define:nn { postnotes/setup }
294 {
295     style .choice: ,
296     style / endnotes .meta:n =
297     {
298         listenv = none ,
299         format =
300         {
301             \footnotesize
302             \setlength { \rightskip } { Opt }
303             \setlength { \leftskip } { Opt }
304             \setlength { \parindent } { 1.8em }
305         } ,
306         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.

```

307     maketextmark =
308     {

```

```

309         \hbox_set:Nn \l__postnotes_tmpa_box
310         { \@textsuperscript { \normalfont ##1 } }
311         \skip_horizontal:n { - \box_wd:N \l__postnotes_tmpa_box }
312         ##2 \box_use:N \l__postnotes_tmpa_box ##3
313     } ,
314 } ,
315 style / pagenote .meta:n =
316 {
317     listenv = none ,
318     format = { } ,
319     pretextmark = { \par\noindent } ,
320     maketextmark = { { \normalfont ##2 ##1 . ##3 } } ,
321     posttextmark = { ~ } ,
322 } ,
323 }

```

\postnotesetup

\postnotesetup Provide \postnotesetup.

```
\postnotesetup{options}
```

```

324 \NewDocumentCommand \postnotesetup { m }
325 { \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 lose 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 be 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 $\langle label \rangle$ 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 occasional `\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 \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.
```

```

327 \int_new:N \g__postnotes_note_id_int
328 \tl_new:N \l_postnotes_note_id_tl
329 \tl_set:Nn \l_postnotes_note_id_tl { \int_use:N \g__postnotes_note_id_int }
330 \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}

331 \NewDocumentCommand \postnote { 0 { } +m }
332 { \__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}

333 \NewHook { postnotes/note/begin }
334 \cs_new_protected:Npn \__postnotes_note:nn #1#2
335 {
336   \group_begin:
337   \keys_set:nn { postnotes/note } {#1}
338   \__postnotes_inhibit_note:F
339   {
340     \int_gincr:N \g__postnotes_note_id_int
341     \tl_if_empty:NT \l__postnotes_mark_tl
342     {
343       \stepcounter { postnote }
344       \tl_set:Ne \l__postnotes_mark_tl { \thepostnote }
345     }
346     \seq_gput_right:Ne \g__postnotes_queue_seq
347     { \l_postnotes_note_id_tl }
348     \UseHook { postnotes/note/begin }
349     \cs_set:Npn \@currentcounter { postnote }
350     \cs_set:Npe \@currentlabel { \p@postnote \l__postnotes_mark_tl }
351     \MakeLinkTarget* { postnote. \l_postnotes_note_id_tl .mark }
352     \__postnotes_set_mark_page_label:e { \l_postnotes_note_id_tl }
353     \__postnotes_set_user_labels:
354     \bool_if:NF \l__postnotes_nomark_bool
355     {
356       \__postnotes_typeset_mark:eV
357       { \l_postnotes_note_id_tl } \l__postnotes_mark_tl
358     }
359     \__postnotes_store:nn { \l_postnotes_note_id_tl } {#2}
360   }
361   \group_end:
362 }

```

(End of definition for `_postnotes_note:nn`.)

Options for `\postnote`.

```

363 \tl_new:N \l__postnotes_mark_tl
364 \bool_new:N \l__postnotes_nomark_bool
365 \fp_new:N \l__postnotes_sort_num_fp
366 \tl_new:N \l__postnotes_note_label_tl
367 \bool_new:N \l__postnotes_manual_sortnum_bool
368 \bool_new:N \l__postnotes_maybe_multi_bool
369 \keys_define:nn { postnotes/note }
370 {
371   markstr .tl_set:N = \l__postnotes_mark_tl ,
372   markstr .value_required:n = true ,
373   sortnum .code:n =
374     {
375       \fp_set:Nn \l__postnotes_sort_num_fp {#1}
376       \bool_set_true:N \l__postnotes_manual_sortnum_bool
377     } ,
378   sortnum .value_required:n = true ,
379   mark .meta:n =
380     {
381       markstr = {#1} ,
382       sortnum = {#1} ,
383     } ,
384   mark .value_required:n = true ,
385   nomark .bool_set:N = \l__postnotes_nomark_bool ,
386   nomark .default:n = true ,
387   label .tl_set:N = \l__postnotes_note_label_tl ,
388   label .value_required:n = true ,
389 }

```

`_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.

```

390 \bool_new:N \l__postnotes_inhibit_note_bool
391 \bool_new:N \l__postnotes_print_plain_mark_bool
392 \bool_new:N \l__postnotes_print_plain_mark_stepcounter_bool
393 \NewHook { postnotes/note/inhibit }
394 \prg_new_protected_conditional:Npnn \_postnotes_inhibit_note: { F }
395 {
396   \bool_set_false:N \l__postnotes_inhibit_note_bool
397   \bool_set_false:N \l__postnotes_print_plain_mark_bool
398   \bool_set_false:N \l__postnotes_print_plain_mark_stepcounter_bool
399   \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.

```

400   \bool_if:NT \l__postnotes_print_plain_mark_bool
401   {
402     \tl_if_empty:NT \l__postnotes_mark_tl
403     {
404       \bool_if:NTF \l__postnotes_print_plain_mark_stepcounter_bool
405       {
406         \stepcounter { postnote }
407         \tl_set:Nc \l__postnotes_mark_tl { \thepostnote }
408       }
409       {
410         \group_begin:
411         \int_incr:N \c@postnote
412         \exp_args:NNNe
413         \group_end:
414         \tl_set:Nn \l__postnotes_mark_tl { \thepostnote }
415       }
416     }
417     \__postnotes_typeset_mark_wrapper:n
418     { \__postnotes_make_mark:nnn { \l__postnotes_mark_tl } { } { } }
419   }
420   \bool_if:NTF \l__postnotes_inhibit_note_bool
421   { \prg_return_true: }
422   { \prg_return_false: }
423 }

```

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

`__postnotes_typeset_mark:nn` 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>}

424 \cs_new_protected:Npn \__postnotes_typeset_mark:nn #1#2
425 {
426   \__postnotes_typeset_mark_wrapper:n
427   {
428     \bool_if:NTF \l__postnotes_hyperlink_bool
429     {
430       \__postnotes_make_mark:nnn {#2}
431       { \hyper@linkstart { link } { postnote. #1 .text } }
432       { \hyper@linkend }
433     }
434     { \__postnotes_make_mark:nnn {#2} { } { } }
435   }
436 }
437 \cs_generate_variant:Nn \__postnotes_typeset_mark:nn { eV }
438 \tl_new:N \l__postnotes_saved_spacefactor_tl
439 \cs_new_protected:Npn \__postnotes_typeset_mark_wrapper:n #1

```

```

440 {
441   \mode_leave_vertical:
442   \mode_if_horizontal:T
443   {
444     \tl_set:Nc \l__postnotes_saved_spacefactor_tl { \the\spacefactor }
445     \nobreak
446   }
447   #1
448   \mode_if_horizontal:T
449   { \spacefactor \l__postnotes_saved_spacefactor_tl }
450   \scan_stop:
451 }

```

(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`.

```

452 \cs_new_protected:Npn \__postnotes_set_user_labels:
453 {
454   \tl_if_empty:NF \l__postnotes_note_label_tl
455   { \exp_args:NV \label \l__postnotes_note_label_tl }
456   \tl_if_empty:NF \l__postnotes_note_zlabel_tl
457   { \exp_args:NV \zlabel \l__postnotes_note_zlabel_tl }
458 }

```

(End of definition for `__postnotes_set_user_labels:.`)

5 `\postnoteref`

`\postnoteref` Provide `\postnoteref`.

```

\postnoteref(*){\label}

459 \NewDocumentCommand \postnoteref { s m }
460 { \__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}

461 \cs_new_protected:Npn \__postnotes_note_ref:nn #1#2
462 {
463   \group_begin:
464   \__postnotes_typeset_mark_wrapper:n
465   {
466     \bool_lazy_and:nnTF
467     { ! #1 }
468     { \l__postnotes_hyperlink_bool }
469     {
470       \hyperref [#2]
471       { \__postnotes_make_mark:nmn { \ref*{#2} } { } { } }

```

```

472     }
473     { \_postnotes_make_mark:nnn { \ref*{#2} } { } { } }
474   }
475   \group_end:
476 }

```

(End of definition for _postnotes_note_ref:nn.)

6 \postnotesection

\postnotesection Provide \postnotesection and \postnotesectionx.
\postnotesectionx

```

\postnotesection[<options>]{<section content>}
\postnotesectionx[<options>]{<section content>}
477 \NewDocumentCommand \postnotesection { 0 { } +m }
478 { \_postnotes_section:nn {#1} {#2} }
479 \NewDocumentCommand \postnotesectionx { 0 { } +m }
480 {
481   % NOTE Command deprecated in 2022-12-27 for v0.2.0.
482   \msg_warning:nn { postnotes } { postnotesectionx-deprecated }
483   \postnotesection [ #1 , exp ] {#2}
484 }
485 \msg_new:nnn { postnotes } { postnotesectionx-deprecated }
486 {
487   ' \iow_char:N\ \postnotesectionx' ~is~deprecated~\msg_line_context:~
488   Use~the~'exp'~option~of~' \iow_char:N\ \postnotesection'~instead.
489 }

```

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

_postnotes_section:nn The internal version of \postnotesection.

```

\_postnotes_section:nn {<options>} {<content>}
490 \int_new:N \g__postnotes_sectid_int
491 \cs_new_protected:Npn \_postnotes_section:nn #1#2
492 {
493   \group_begin:
494   \int_gincr:N \g__postnotes_sectid_int
495   \int_gincr:N \g__postnotes_note_id_int
496   \seq_gput_right:Ne \g__postnotes_queue_seq { \l_postnotes_note_id_tl }
497   \tl_gclear:N \g__postnotes_section_name_tl
498   \keys_set:nn { postnotes/section } {#1}
499   \bool_if:NTF \l__postnotes_section_exp_bool
500     { \_postnotes_store_section:ne { \l_postnotes_note_id_tl } {#2} }
501     { \_postnotes_store_section:nn { \l_postnotes_note_id_tl } {#2} }
502   \group_end:
503 }

```

(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.

```

504 \tl_new:N \g__postnotes_section_name_tl
505 \bool_new:N \l__postnotes_section_exp_bool
506 \keys_define:nn { postnotes/section }
507   {
508     name .tl_gset:N = \g__postnotes_section_name_tl ,
509     name .value_required:n = true ,
510     exp .bool_set:N = \l__postnotes_section_exp_bool ,
511     exp .initial:n = false ,
512     exp .default:n = true ,
513   }

```

7 \printpostnotes

\printpostnotes Provide \printpostnotes.

```
\printpostnotes
```

```

514 \NewDocumentCommand \printpostnotes { }
515   { \__postnotes_print_notes: }

```

(End of definition for \printpostnotes.)

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

```

\pnthechapternextnote 516 \tl_new:N \pnthechapter
\pnthesectionnextnote 517 \tl_new:N \pnthesection
  \pnthechapternextnote 518 \tl_new:N \pnidnextnote
  \pnthesectionnextnote 519 \tl_new:N \pnthechapternextnote
  \pnidnextnote         520 \tl_new:N \pnthesectionnextnote
  \pnidnextnote         521 \tl_new:N \pnthechapter

```

(End of definition for \pnthechapter and others.)

\g__postnotes_print_postnotes_int Auxiliary variables for __postnotes_print_notes:.

```

\l__postnotes_print_note_id_tl 522 \int_new:N \g__postnotes_print_postnotes_int
\l__postnotes_print_note_id_next_tl 523 \tl_new:N \l__postnotes_print_note_id_tl
  \l__postnotes_print_counter_tl 524 \tl_new:N \l__postnotes_print_note_id_next_tl
\l__postnotes_print_mark_tl 525 \tl_new:N \l__postnotes_print_counter_tl
  \l__postnotes_print_type_curr_tl 526 \tl_new:N \l__postnotes_print_mark_tl
  \l__postnotes_print_type_next_tl 527 \tl_new:N \l__postnotes_print_type_curr_tl
  \l__postnotes_print_type_prev_tl 528 \tl_new:N \l__postnotes_print_type_next_tl
  \l__postnotes_print_content_tl 529 \tl_new:N \l__postnotes_print_type_prev_tl
  \l__postnotes_clear_queue_seq 530 \tl_new:N \l__postnotes_print_content_tl
  \l__postnotes_clear_queue_seq 531 \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.

```
532 \NewHook { postnotes/print/begin }
533 \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 L^AT_EX 2_ε counters, since they may be required externally. Their main intended use case is to support `zref-clever`, but in principle they can be of general use.

```
534 \newcounter { postnotetext }
535 \newcounter { postnotesection }
536 \setcounter { postnotesection } { 10000 }
```

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

```

    \__postnotes_print_notes:
537 \cs_new_protected:Npn \__postnotes_print_notes:
538   {
539     \group_begin:
540     \int_gincr:N \g__postnotes_print_postnotes_int
541     \seq_if_empty:NTF \g__postnotes_queue_seq
542       { \msg_warning:nn { postnotes } { empty-printpostnotes } }
543     {
544       \pnheading
545       \UseHook { postnotes/print/begin }
546       \tl_set:Nn \l__postnotes_print_type_prev_tl { open }
547       \seq_set_eq:NN \l__postnotes_clear_queue_seq \g__postnotes_queue_seq
548       \__postnotes_verify_multipass:N \g__postnotes_queue_seq
549       \bool_if:NT \l__postnotes_sort_bool
550         { \__postnotes_sort_queue:N \g__postnotes_queue_seq }
551       \bool_gset_true:N \g__postnotes_header_vars_next_bool
552       \__postnotes_get_headers_data:N \g__postnotes_queue_seq
553       \__postnotes_set_headers_vars_first:
```

Ensure the first note after a heading has paragraph indentation when `listenv` is `none`. `endnotes` uses a workaroundsish solution in `\noteheading`, 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).

```

554     \bool_if:NF \l__postnotes_print_as_list_bool
555     {
556         \cs_set_eq:NN \@afterindentfalse \@afterindenttrue
557         \@afterindenttrue
558     }
559     \bool_until_do:nn { \seq_if_empty_p:N \g__postnotes_queue_seq }
560     {
561         \seq_gpop_left:NN \g__postnotes_queue_seq
562         \l_postnotes_print_note_id_tl
563         \__postnotes_prop_get:nnN { \l_postnotes_print_note_id_tl }
564         { type } \l__postnotes_print_type_curr_tl
565         \tl_if_eq:NnTF \l__postnotes_print_type_curr_tl { section }
566         { % type_curr = 'section'
567             \seq_if_empty:NTF \g__postnotes_queue_seq
568             {
569                 \tl_set:Nn \l__postnotes_print_note_id_next_tl { noid }
570                 \tl_set:Nn \l__postnotes_print_type_next_tl { close }
571             }
572             {
573                 \seq_get_left:NN \g__postnotes_queue_seq
574                 \l__postnotes_print_note_id_next_tl
575                 \__postnotes_prop_get:nnN
576                 { \l__postnotes_print_note_id_next_tl }
577                 { type } \l__postnotes_print_type_next_tl
578             }
579         }
580     }

```

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

```

579     \tl_if_eq:NnTF \l__postnotes_print_type_next_tl { note }
580     {
581         \stepcounter { postnotessection }
582         \group_begin:
583         \__postnotes_prop_get:nnN
584         { \l_postnotes_print_note_id_tl }
585         { thechapter } \pnthechapter
586         \__postnotes_prop_get:nnN
587         { \l_postnotes_print_note_id_tl }
588         { thesection } \pnthesection
589         \tl_set:NV \pnidnextnote \l__postnotes_print_note_id_next_tl
590         \__postnotes_prop_get:nnN
591         { \l__postnotes_print_note_id_next_tl }
592         { thechapter } \pnthechapternextnote
593         \__postnotes_prop_get:nnN
594         { \l__postnotes_print_note_id_next_tl }
595         { thesection } \pnthesectionnextnote
596         \__postnotes_prop_get:nnN
597         { \l_postnotes_print_note_id_tl }
598         { content } \l__postnotes_print_content_tl
599         \l__postnotes_print_content_tl
600         \group_end:

```

Set `type_prev` for the next iteration.

```

601         \tl_set:NV \l__postnotes_print_type_prev_tl
602         \l__postnotes_print_type_curr_tl
603     }
604 }
605 { % type_curr = 'note'
606 \tl_if_eq:NnF \l__postnotes_print_type_prev_tl { note }
607 {
608     \bool_if:NTF \l__postnotes_print_as_list_bool
609     { \exp_args:Ne \begin { \l__postnotes_print_env_tl } }
610     { \group_begin: }
611     \l__postnotes_print_format_tl
612 }
613 \group_begin:
614 \UseHook { postnotes/print/note/begin }
615 \__postnotes_get_pageref:Ne \pnthepage
616 { mark@ \l_postnotes_print_note_id_tl }
617 \__postnotes_prop_get:nnN
618 { \l_postnotes_print_note_id_tl }
619 { mark } \l__postnotes_print_mark_tl
620 \__postnotes_prop_get:nnN
621 { \l_postnotes_print_note_id_tl }
622 { counter } \l__postnotes_print_counter_tl
623 \__postnotes_prop_get:nnN
624 { \l_postnotes_print_note_id_tl }
625 { content } \l__postnotes_print_content_tl
626 \cs_set:Npn \@currentcounter { postnotetext }
627 \int_set:Nn \c@postnotetext
628 { \l__postnotes_print_counter_tl }
629 \cs_set:Npe \@currentlabel
630 { \p@postnote \l__postnotes_print_mark_tl }
631 \__postnotes_text_mark_wrapper:n
632 {
633     \MakeLinkTarget*
634     { postnote. \l_postnotes_print_note_id_tl .text }
635     \__postnotes_set_text_page_label:e
636     { \l_postnotes_print_note_id_tl }
637     \__postnotes_typeset_text_mark:eV
638     { \l_postnotes_print_note_id_tl }
639     \l__postnotes_print_mark_tl
640 }
641 \l__postnotes_print_content_tl
642 \l__postnotes_post_printnote_tl
643 \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.

```

644 \seq_if_empty:NTF \g__postnotes_queue_seq
645 {
646     \tl_set:Nn \l__postnotes_print_note_id_next_tl { noid }
647     \tl_set:Nn \l__postnotes_print_type_next_tl { close }
648 }
649 {
650     \seq_get_left:NN \g__postnotes_queue_seq

```

```

651         \l__postnotes_print_note_id_next_tl
652         \__postnotes_prop_get:nnN
653         { \l__postnotes_print_note_id_next_tl }
654         { type } \l__postnotes_print_type_next_tl
655     }
656     \tl_if_eq:NnF \l__postnotes_print_type_next_tl { note }
657     {
658         \bool_if:NTF \l__postnotes_print_as_list_bool
659         { \exp_args:Ne \end { \l__postnotes_print_env_tl } }
660         { \group_end: }

```

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

```

661         \par
662     }

```

Set `type_prev` for the next iteration.

```

663         \tl_set:NV \l__postnotes_print_type_prev_tl
664         \l__postnotes_print_type_curr_tl
665     }
666 }
667 \AddToHookNext { shipout/after }
668 { \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.

```

669     \seq_map_inline:Nn \l__postnotes_clear_queue_seq
670     { \__postnotes_prop_gclear:n { ##1 } }
671 }
672 \group_end:
673 }

```

(End of definition for `__postnotes_print_notes:`)

```

674 \msg_new:nnn { postnotes } { empty-printpostnotes }
675 { 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>}
676 \cs_new_protected:Npn \__postnotes_typeset_text_mark:nn #1#2
677 {
678     \bool_lazy_and:nnTF
679     { \l__postnotes_hyperlink_bool }
680     { \l__postnotes_backlink_bool }
681     {
682         \__postnotes_make_text_mark:nnn {#2}
683         { \hyper@linkstart { link } { postnote. #1 .mark } }
684         { \hyper@linkend }
685     }
686     { \__postnotes_make_text_mark:nnn {#2} { } { } }
687 }

```

```

688 \cs_generate_variant:Nn \__postnotes_typeset_text_mark:nn { eV }
689 \cs_new_protected:Npn \__postnotes_text_mark_wrapper:n #1
690 {
691   \bool_if:NTF \l__postnotes_print_as_list_bool
692   {
693     \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.

```

694     \mode_leave_vertical:
695   }
696   { \l__postnotes_pre_textmark_tl #1 \l__postnotes_post_textmark_tl }
697 }

```

(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 its 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)
698 \cs_new_protected:Npn \__postnotes_verify_multipass:N #1
699 {
700   \group_begin:
701   \seq_clear:N \l__postnotes_tmpa_seq
702   \seq_map_inline:Nn #1
703   {
704     \__postnotes_prop_get:nnN {##1} { multibool } \l__postnotes_tmpa_tl
705     \str_if_eq:VnTF \l__postnotes_tmpa_tl { true }
706     {
707       \cs_if_exist:cT
708       { \c__postnotes_ref_prefix_tl @ mark@ ##1 }
709       { \seq_put_right:Nn \l__postnotes_tmpa_seq {##1} }
710     }
711     { \seq_put_right:Nn \l__postnotes_tmpa_seq {##1} }
712   }
713   \seq_gset_eq:NN #1 \l__postnotes_tmpa_seq

```

```

714     \group_end:
715   }

```

(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
716 \cs_new_protected:Npn \__postnotes_sort_queue:N #1
717 {
718   \group_begin:
719   \seq_gsort:Nn #1
720   {
721     \__postnotes_prop_get:nnN {##1} { psectid } \l__postnotes_tmpa_tl
722     \__postnotes_prop_get:nnN {##2} { psectid } \l__postnotes_tmpb_tl
723     \tl_if_eq:NNTF \l__postnotes_tmpa_tl \l__postnotes_tmpb_tl
724     {
725       \__postnotes_prop_get:nnN {##1} { type } \l__postnotes_tmpa_tl
726       \__postnotes_prop_get:nnN {##2} { type } \l__postnotes_tmpb_tl
727       \bool_lazy_and:nnTF
728         { \str_if_eq_p:Vn \l__postnotes_tmpa_tl { note } }
729         { \str_if_eq_p:Vn \l__postnotes_tmpb_tl { note } }
730       {
731         \__postnotes_prop_get:nnN {##1} { sortnum } \l__postnotes_tmpa_tl
732         \__postnotes_prop_get:nnN {##2} { sortnum } \l__postnotes_tmpb_tl
733         \fp_compare:nNnTF
734           { \l__postnotes_tmpa_tl } > { \l__postnotes_tmpb_tl }
735           { \sort_return_swapped: }
736           { \sort_return_same: }
737       }
738       { \sort_return_same: }
739     }
740     { \sort_return_same: }
741   }
742   \group_end:
743 }

```

(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.

<code>\pnhdpagefirst</code>	User facing variables, aimed at making available header data for the user. Setting these
<code>\pnhdpagelast</code>	variables with correct values at the moment the header gets typeset is <i>the</i> objective of
<code>\pnhdchapfirst</code>	the whole headers infrastructure of the package.
<code>\pnhdchaplast</code>	
<code>\pnhdsectfirst</code>	744 <code>\tl_new:N \pnhdpagefirst</code>
<code>\pnhdsectlast</code>	745 <code>\tl_new:N \pnhdpagelast</code>
<code>\pnhdnamefirst</code>	746 <code>\tl_new:N \pnhdchapfirst</code>
<code>\pnhdnamelast</code>	747 <code>\tl_new:N \pnhdchaplast</code>
	748 <code>\tl_new:N \pnhdsectfirst</code>
	749 <code>\tl_new:N \pnhdsectlast</code>
	750 <code>\tl_new:N \pnhdnamefirst</code>
	751 <code>\tl_new:N \pnhdnamelast</code>

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

<code>\g__postnotes_header_page_first_prop</code>	Auxiliary variables for the headers infrastructure.
<code>\g__postnotes_header_page_last_prop</code>	752 <code>\prop_new:N \g__postnotes_header_page_first_prop</code>
<code>\g__postnotes_header_chap_first_prop</code>	753 <code>\prop_new:N \g__postnotes_header_page_last_prop</code>
<code>\g__postnotes_header_chap_last_prop</code>	754 <code>\prop_new:N \g__postnotes_header_chap_first_prop</code>
<code>\g__postnotes_header_sect_first_prop</code>	755 <code>\prop_new:N \g__postnotes_header_chap_last_prop</code>
<code>\g__postnotes_header_sect_last_prop</code>	756 <code>\prop_new:N \g__postnotes_header_sect_first_prop</code>
<code>\g__postnotes_header_name_first_prop</code>	757 <code>\prop_new:N \g__postnotes_header_sect_last_prop</code>
<code>\g__postnotes_header_name_last_prop</code>	758 <code>\prop_new:N \g__postnotes_header_name_first_prop</code>
<code>\g__postnotes_header_prev_last_page_tl</code>	759 <code>\prop_new:N \g__postnotes_header_name_last_prop</code>
<code>\g__postnotes_header_prev_last_chap_tl</code>	760 <code>\tl_new:N \g__postnotes_header_prev_last_page_tl</code>
<code>\g__postnotes_header_prev_last_sect_tl</code>	761 <code>\tl_new:N \g__postnotes_header_prev_last_chap_tl</code>
<code>\g__postnotes_header_prev_last_name_tl</code>	762 <code>\tl_new:N \g__postnotes_header_prev_last_sect_tl</code>
<code>\l__postnotes_prev_text_page_tl</code>	
<code>\l__postnotes_curr_text_page_tl</code>	
<code>\l__postnotes_prev_mark_page_tl</code>	
<code>\l__postnotes_prev_mark_chap_tl</code>	
<code>\l__postnotes_prev_mark_sect_tl</code>	
<code>\l__postnotes_prev_mark_name_tl</code>	

```

763 \tl_new:N \g__postnotes_header_prev_last_name_tl
764 \tl_new:N \l__postnotes_prev_text_page_tl
765 \tl_new:N \l__postnotes_curr_text_page_tl
766 \tl_new:N \l__postnotes_prev_mark_page_tl
767 \tl_new:N \l__postnotes_prev_mark_chap_tl
768 \tl_new:N \l__postnotes_prev_mark_sect_tl
769 \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)
770 \cs_new_protected:Npn \__postnotes_get_headers_data:N #1
771 {
772   \group_begin:
773   \tl_gclear:N \pnhdpagefirst
774   \tl_gclear:N \pnhdpagelast
775   \tl_gclear:N \pnhdchapfirst
776   \tl_gclear:N \pnhdchaplast
777   \tl_gclear:N \pnhdsectfirst
778   \tl_gclear:N \pnhdsectlast
779   \tl_gclear:N \pnhdnamefirst
780   \tl_gclear:N \pnhdnamelast
781   \prop_gclear:N \g__postnotes_header_page_first_prop
782   \prop_gclear:N \g__postnotes_header_page_last_prop
783   \prop_gclear:N \g__postnotes_header_chap_first_prop
784   \prop_gclear:N \g__postnotes_header_chap_last_prop
785   \prop_gclear:N \g__postnotes_header_sect_first_prop
786   \prop_gclear:N \g__postnotes_header_sect_last_prop
787   \prop_gclear:N \g__postnotes_header_name_first_prop
788   \prop_gclear:N \g__postnotes_header_name_last_prop
789   \tl_gclear:N \g__postnotes_header_prev_last_page_tl
790   \tl_gclear:N \g__postnotes_header_prev_last_chap_tl
791   \tl_gclear:N \g__postnotes_header_prev_last_sect_tl
792   \tl_gclear:N \g__postnotes_header_prev_last_name_tl
793   \tl_clear:N \l__postnotes_prev_text_page_tl
794   \tl_clear:N \l__postnotes_curr_text_page_tl
795   \tl_clear:N \l__postnotes_prev_mark_page_tl
796   \tl_clear:N \l__postnotes_prev_mark_chap_tl
797   \tl_clear:N \l__postnotes_prev_mark_sect_tl
798   \tl_clear:N \l__postnotes_prev_mark_name_tl
799   \seq_map_inline:Nn #1
800   {
801     \exp_args:Ne \tl_if_eq:nnT
802     { \__postnotes_prop_item:n {##1} { type } }
803     { note }
804     {
805       \__postnotes_get_pageref:Nn
806       \l__postnotes_curr_text_page_tl { text@ ##1 }
807       \tl_if_empty:NF \l__postnotes_curr_text_page_tl
808       {
809         \tl_if_eq:NNTF

```

```

810         \l__postnotes_prev_text_page_tl
811         \l__postnotes_curr_text_page_tl
812     {

```

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

```

813         \__postnotes_get_pageref:Nn
814         \l__postnotes_prev_mark_page_tl { mark@ ##1 }
815         \__postnotes_prop_get:nnN {##1} { thechapter }
816         \l__postnotes_prev_mark_chap_tl
817         \__postnotes_prop_get:nnN {##1} { thesection }
818         \l__postnotes_prev_mark_sect_tl
819         \__postnotes_prop_get:nnN {##1} { pnsectname }
820         \l__postnotes_prev_mark_name_tl
821     }
822 {

```

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).

```

823         \tl_if_empty:NF \l__postnotes_prev_text_page_tl
824         {
825             \prop_gput:Nee \g__postnotes_header_page_last_prop
826             { \l__postnotes_prev_text_page_tl }
827             { \l__postnotes_prev_mark_page_tl }
828             \prop_gput:Nee \g__postnotes_header_chap_last_prop
829             { \l__postnotes_prev_text_page_tl }
830             { \l__postnotes_prev_mark_chap_tl }
831             \prop_gput:Nee \g__postnotes_header_sect_last_prop
832             { \l__postnotes_prev_text_page_tl }
833             { \l__postnotes_prev_mark_sect_tl }
834             \prop_gput:Nee \g__postnotes_header_name_last_prop
835             { \l__postnotes_prev_text_page_tl }
836             { \l__postnotes_prev_mark_name_tl }
837         }

```

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

```

838         \prop_gput:Nee \g__postnotes_header_page_first_prop
839         { \l__postnotes_curr_text_page_tl }
840         { \__postnotes_extract_pageref:n { mark@ ##1 } }
841         \prop_gput:Nee \g__postnotes_header_chap_first_prop
842         { \l__postnotes_curr_text_page_tl }
843         { \__postnotes_prop_item:nn {##1} { thechapter } }
844         \prop_gput:Nee \g__postnotes_header_sect_first_prop
845         { \l__postnotes_curr_text_page_tl }
846         { \__postnotes_prop_item:nn {##1} { thesection } }
847         \prop_gput:Nee \g__postnotes_header_name_first_prop
848         { \l__postnotes_curr_text_page_tl }
849         { \__postnotes_prop_item:nn {##1} { pnsectname } }

```

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

```

850         \__postnotes_get_pageref:Nn

```

```

851         \l__postnotes_prev_mark_page_tl { mark@ ##1 }
852         \__postnotes_prop_get:nnN {##1} { thechapter }
853         \l__postnotes_prev_mark_chap_tl
854         \__postnotes_prop_get:nnN {##1} { thesection }
855         \l__postnotes_prev_mark_sect_tl
856         \__postnotes_prop_get:nnN {##1} { pnsectname }
857         \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).

```

858         \tl_set:NV \l__postnotes_prev_text_page_tl
859         \l__postnotes_curr_text_page_tl
860     }
861 }
862 }
863 }

```

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.

```

864     \tl_if_empty:NF \l__postnotes_prev_text_page_tl
865     {
866         \prop_gput:Nee \g__postnotes_header_page_last_prop
867         { \l__postnotes_prev_text_page_tl }
868         { \l__postnotes_prev_mark_page_tl }
869         \prop_gput:Nee \g__postnotes_header_chap_last_prop
870         { \l__postnotes_prev_text_page_tl }
871         { \l__postnotes_prev_mark_chap_tl }
872         \prop_gput:Nee \g__postnotes_header_sect_last_prop
873         { \l__postnotes_prev_text_page_tl }
874         { \l__postnotes_prev_mark_sect_tl }
875         \prop_gput:Nee \g__postnotes_header_name_last_prop
876         { \l__postnotes_prev_text_page_tl }
877         { \l__postnotes_prev_mark_name_tl }
878     }
879     \group_end:
880 }

```

(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>}

```

```

881 \cs_new_protected:Npn \__postnotes_set_headers_vars:n #1
882 {
883   \group_begin:
884   \prop_get:NnNTF \g__postnotes_header_page_first_prop
885     {#1} \l__postnotes_tmpa_tl
886     { \tl_gset:NV \pnhdpagefirst \l__postnotes_tmpa_tl }
887     { \tl_gset:NV \pnhdpagefirst \g__postnotes_header_prev_last_page_tl }
888   \prop_get:NnNTF \g__postnotes_header_page_last_prop
889     {#1} \l__postnotes_tmpa_tl
890     {
891       \tl_gset:NV \pnhdpagelast \l__postnotes_tmpa_tl
892       \tl_gset:NV \g__postnotes_header_prev_last_page_tl
893         \l__postnotes_tmpa_tl
894     }
895     { \tl_gset:NV \pnhdpagelast \g__postnotes_header_prev_last_page_tl }
896   \prop_get:NnNTF \g__postnotes_header_chap_first_prop
897     {#1} \l__postnotes_tmpa_tl
898     { \tl_gset:NV \pnhdchapfirst \l__postnotes_tmpa_tl }
899     { \tl_gset:NV \pnhdchapfirst \g__postnotes_header_prev_last_chap_tl }
900   \prop_get:NnNTF \g__postnotes_header_chap_last_prop
901     {#1} \l__postnotes_tmpa_tl
902     {
903       \tl_gset:NV \pnhdchaplast \l__postnotes_tmpa_tl
904       \tl_gset:NV \g__postnotes_header_prev_last_chap_tl
905         \l__postnotes_tmpa_tl
906     }
907     { \tl_gset:NV \pnhdchaplast \g__postnotes_header_prev_last_chap_tl }
908   \prop_get:NnNTF \g__postnotes_header_sect_first_prop
909     {#1} \l__postnotes_tmpa_tl
910     { \tl_gset:NV \pnhdsectfirst \l__postnotes_tmpa_tl }
911     { \tl_gset:NV \pnhdsectfirst \g__postnotes_header_prev_last_sect_tl }
912   \prop_get:NnNTF \g__postnotes_header_sect_last_prop
913     {#1} \l__postnotes_tmpa_tl
914     {
915       \tl_gset:NV \pnhdsectlast \l__postnotes_tmpa_tl
916       \tl_gset:NV \g__postnotes_header_prev_last_sect_tl
917         \l__postnotes_tmpa_tl
918     }
919     { \tl_gset:NV \pnhdsectlast \g__postnotes_header_prev_last_sect_tl }
920   \prop_get:NnNTF \g__postnotes_header_name_first_prop
921     {#1} \l__postnotes_tmpa_tl
922     { \tl_gset:NV \pnhdnamefirst \l__postnotes_tmpa_tl }
923     { \tl_gset:NV \pnhdnamefirst \g__postnotes_header_prev_last_name_tl }
924   \prop_get:NnNTF \g__postnotes_header_name_last_prop
925     {#1} \l__postnotes_tmpa_tl
926     {
927       \tl_gset:NV \pnhdnamelast \l__postnotes_tmpa_tl
928       \tl_gset:NV \g__postnotes_header_prev_last_name_tl
929         \l__postnotes_tmpa_tl
930     }
931     { \tl_gset:NV \pnhdnamelast \g__postnotes_header_prev_last_name_tl }
932   \group_end:
933 }
934 \cs_generate_variant:Nn \__postnotes_set_headers_vars:n { e }

```

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

`__postnotes_set_headers_vars_next:` 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.

```
935 \AddToHook { shipout/before } [ postnotes/header ]
936   { \__postnotes_set_headers_vars_next: }
937 \bool_new:N \g__postnotes_header_vars_next_bool
938 \cs_new_protected:Npn \__postnotes_set_headers_vars_next:
939   {
940     \bool_if:NT \g__postnotes_header_vars_next_bool
941       { \__postnotes_set_headers_vars:e { \int_eval:n { \c@page + 1 } } }
942   }
943 \cs_new_protected:Npn \__postnotes_set_headers_vars_first:
944   {
945     \__postnotes_set_print_page_label:e
946     { \int_use:N \g__postnotes_print_postnotes_int }
947     \__postnotes_set_headers_vars:e
948     {
949       \__postnotes_extract_pageref:e
950       { print@ \int_use:N \g__postnotes_print_postnotes_int }
951     }
952   }
```

(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`

```
953 \NewDocumentCommand \pnheaderdefault {}
954   {
955     \tl_if_eq:NNTF \pnhdpagefirst \pnhdpagelast
956     { \pnhdnotes{} ~ \pnhdtopage{} ~ \pnhdpagefirst }
957     { \pnhdnotes{} ~ \pnhdtopages{} ~ \pnhdpagefirst -- \pnhdpagelast }
958   }
```

(End of definition for `\pnheaderdefault`.)

9 Compatibility

A dedicated temp variable for restoring data.

```
959 \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.

```
960 \AddToHook { postnotes/note/begin } [ postnotes ]
961   {
962     \cs_if_exist:NT \@capttype
963       { \bool_set_true:N \l__postnotes_maybe_multi_bool }
964   }
```

biblatex

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

```
965 \AddToHook { package/biblatex/after }
966   {
```

Let `biblatex` know we are in a “notes” context. See <https://tex.stackexchange.com/a/304464>, including comments.

```
967     \AddToHook { postnotes/print/begin } [ postnotes ]
968     { \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.

```
969     \AddToHook { begindocument/before } [ postnotes/mkbibendnote ]
970     {
971       \cs_set:Npn \blx@theendnote { \postnote }
972       \cs_set:Npn \blx@theendnotetext
973         { \blx@err@endnote \footnotetext }
974     }
975   }
976 <*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):

```
977 \AddToHook { package/biblatex/after }
978 {
```

Store biblatex variables for each note.

```
979   \AddToHook { postnotes/note/store } [ postnotes ]
980   {
981     \prop_gput:cne { \__postnotes_data_name:e { \l_postnotes_note_id_tl } }
982     { biblatex@refsection } { \int_use:N \c@refsection }
983     \prop_gput:cne { \__postnotes_data_name:e { \l_postnotes_note_id_tl } }
984     { biblatex@refsegment } { \int_use:N \c@refsegment }
985     \prop_gput:cne { \__postnotes_data_name:e { \l_postnotes_note_id_tl } }
986     { biblatex@refcontextbool }
987     { \iftoggle { blx@refcontext } { true } { false } }
988     \prop_gput:cnV { \__postnotes_data_name:e { \l_postnotes_note_id_tl } }
989     { biblatex@refcontext } \blx@refcontext@context
990   }
```

biblatex setup, once for \printpostnotes call.

```
991   \AddToHook { postnotes/print/begin } [ postnotes ]
992   {
993     \__postnotes_biblatex_endrefcontext_local:
994     \__postnotes_biblatex_citereset_local:
995   }
```

Restore biblatex variables for each note.

```
996   \AddToHook { postnotes/print/note/begin } [ postnotes ]
997   {
998     \__postnotes_prop_get:nnN { \l_postnotes_print_note_id_tl }
999     { biblatex@refsection } \l__postnotes_restore_tmp_tl
1000   \int_set:Nn \c@refsection { \l__postnotes_restore_tmp_tl }
1001   \__postnotes_prop_get:nnN { \l_postnotes_print_note_id_tl }
1002   { biblatex@refsegment } \l__postnotes_restore_tmp_tl
1003   \int_set:Nn \c@refsegment { \l__postnotes_restore_tmp_tl }
1004   \__postnotes_prop_get:nnN { \l_postnotes_print_note_id_tl }
1005   { biblatex@refcontextbool } \l__postnotes_restore_tmp_tl
1006   \use:c { toggle \l__postnotes_restore_tmp_tl } { blx@refcontext }
1007   \__postnotes_prop_get:nnN { \l_postnotes_print_note_id_tl }
1008   { biblatex@refcontext } \l__postnotes_restore_tmp_tl
1009   \blx@edef@refcontext { \l__postnotes_restore_tmp_tl }
1010 }
```

Auxiliary functions.

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

```
1011   \cs_new_protected:Npn \__postnotes_biblatex_endrefcontext_local:
1012   {
1013     \togglefalse { blx@refcontext }
1014     \tl_clear:N \blx@refcontext@labelprefix
1015     \tl_clear:N \blx@refcontext@labelprefix@real
1016     \tl_set:Ne \blx@refcontext@sortingtemplatename { \blx@sorting }
1017     \tl_set:Nn \blx@refcontext@sortingnamekeytemplatename { global }
1018     \tl_set:Nn \blx@refcontext@uniquenametemplatename { global }
1019     \tl_set:Nn \blx@refcontext@labelalphanametemplatename { global }
```

```

1020     \blx@edef@refcontext
1021     {
1022         \blx@refcontext@sortingtemplatename /
1023         \blx@refcontext@sortingnamekeytemplatename /
1024         /
1025         \blx@refcontext@uniquenametemplatename /
1026         \blx@refcontext@labelalphanametemplatename
1027     }
1028 }

```

(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.

```

1029     \cs_new_protected:Npn \__postnotes_biblatex_citereset_local:
1030     {
1031         \global\cslet{blx@bsee@\the\c@refsection}\@empty
1032         \global\cslet{blx@fsee@\the\c@refsection}\@empty
1033         \tl_clear:c { blx@bsee@ \int_use:N \c@refsection }
1034         \tl_clear:c { blx@fsee@ \int_use:N \c@refsection }
1035         \blx@ibidreset@force
1036         \undef \blx@lastkey@text
1037         \undef \blx@lastkey@foot
1038         \blx@idemreset@force
1039         \undef \blx@lasthash@text
1040         \undef \blx@lasthash@foot
1041         \blx@opcitreset@force
1042         \clist_map_inline:Nn \blx@trackhash@text
1043         { \csundef { blx@lastkey@text@ ##1 } }
1044         \tl_clear:N \blx@trackhash@text
1045         \clist_map_inline:Nn \blx@trackhash@foot
1046         { \csundef { blx@lastkey@foot@ ##1 } }
1047         \tl_clear:N \blx@trackhash@foot
1048         \blx@loccitreset@force
1049         \clist_map_inline:Nn \blx@trackkeys@text
1050         { \csundef { blx@lastnote@text@ ##1 } }
1051         \tl_clear:N \blx@trackkeys@text
1052         \clist_map_inline:Nn \blx@trackkeys@foot
1053         { \csundef { blx@lastnote@foot@ ##1 } }
1054         \tl_clear:N \blx@trackkeys@foot
1055         and all of them do:
1056         \cs_set_eq:NN \blx@lastmpfn \z@
1057     }
1058 }

```

(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 $\langle resources \rangle$) 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`.

```

1052 \AddToHook { package/biblatex/after }
1053 {
1054   \tl_new:N \l__postnotes_biblatex_orig_refsection_tl
1055   \tl_new:N \g__postnotes_biblatex_prev_refsection_tl
1056   \AddToHook { postnotes/print/begin } [ postnotes ]
1057   {
1058     \tl_set:Nc \l__postnotes_biblatex_orig_refsection_tl
1059     { \int_use:N \c@refsection }
1060     \tl_gset:Nc \g__postnotes_biblatex_prev_refsection_tl
1061     { \l__postnotes_biblatex_orig_refsection_tl }
1062   }
1063   \AddToHook { postnotes/print/note/begin } [ postnotes ]
1064   {
1065     \__postnotes_prop_get:nnN { \l__postnotes_print_note_id_tl }
1066     { biblatex@refsection } \l__postnotes_restore_tmp_tl
1067     \tl_if_eq:N NF
1068     \l__postnotes_restore_tmp_tl
1069     \g__postnotes_biblatex_prev_refsection_tl
1070     {
1071       \int_set:Nn \c@blx@maxsection
1072       { \l__postnotes_restore_tmp_tl - 1 }
1073       \tl_gset_eq:NN \g__postnotes_biblatex_prev_refsection_tl
1074       \l__postnotes_restore_tmp_tl
1075       \group_begin:
1076       \cs_set_eq:NN \label \use_none:n
1077       \cs_set_eq:NN \blx@info \use_none:n
1078       \blx@endrefsection
1079       \refsection
1080       \group_end:
1081     }
1082   }
1083 }
1084 \</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:`.

```
1085 \tl_new:N \l__postnotes_note_zlabel_tl
(End of definition for \l__postnotes_note_zlabel_tl.)
```

```
1086 \AddToHook { package/zref-user/after }
1087 {
```

Provide `zlabel` option.

```
1088   \keys_define:nn { postnotes/note }
1089   {
1090     zlabel .tl_set:N = \l__postnotes_note_zlabel_tl ,
1091     zlabel .value_required:n = true ,
1092   }
```

`\postnotezref` Provide `\postnotezref`.

```
   \postnotezref*{<label>}
1093   \NewDocumentCommand \postnotezref { s m }
1094   { \__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>}
1095   \cs_new_protected:Npn \__postnotes_note_zref:nn #1#2
1096   {
1097     \group_begin:
1098     \__postnotes_typeset_mark_wrapper:n
1099     {
1100       \bool_lazy_all:nTF
1101       {
1102         { ! #1 }
1103         { \l__postnotes_hyperlink_bool }
1104         { \l__postnotes_zrefhyperref_bool }
1105       }
1106       {
1107         \hyperlink
1108         { \zref@extractdefault {#2} { anchor } { } }
1109         { \__postnotes_make_mark:nnn { \zref{#2} } { } { } }
1110       }
1111       { \__postnotes_make_mark:nnn { \zref{#2} } { } { } }
1112     }
1113     \group_end:
1114   }
```

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

```
1115 }
1116 \bool_new:N \l__postnotes_zrefhyperref_bool
1117 \AddToHook { package/zref-hyperref/after }
1118 { \bool_set_true:N \l__postnotes_zrefhyperref_bool }
```

zref-clever

```
1119 \AddToHook { package/zref-clever/after }
1120 {
1121   \zcsetup
1122   {
1123     countertype = { postnote = endnote } ,
1124     countertype = { postnotetext = endnote } ,
1125   }
1126   \AddToHook { postnotes/print/begin } [ postnotes ]
1127   { \zcsetup { counterresetby = { postnotetext = postnotesection } } }
1128 }
```

zref-check

```
1129 \AddToHook { package/zref-check/after }
1130 {
1131   \IfPackageAtLeastTF { zref-check } { 2022-07-05 }
1132   {
1133     \AddToHook { postnotes/note/store } [ postnotes ]
1134     {
1135       \prop_gput:cne { \_postnotes_data_name:e { \l_postnotes_note_id_tl } }
1136       { zref-check@abschap } { \int_use:N \c@zc@abschap }
1137       \prop_gput:cne { \_postnotes_data_name:e { \l_postnotes_note_id_tl } }
1138       { zref-check@abssec } { \int_use:N \c@zc@abssec }
1139     }
1140     \AddToHook { postnotes/print/note/begin } [ postnotes ]
1141     {
1142       \__postnotes_prop_get:nnN { \l_postnotes_print_note_id_tl }
1143       { zref-check@abschap } \l__postnotes_restore_tmp_tl
1144       \int_set:Nn \c@zc@abschap { \l__postnotes_restore_tmp_tl }
1145       \__postnotes_prop_get:nnN { \l_postnotes_print_note_id_tl }
1146       { zref-check@abssec } \l__postnotes_restore_tmp_tl
1147       \int_set:Nn \c@zc@abssec { \l__postnotes_restore_tmp_tl }
1148     }
1149   }
1150 }
1151 }
```

amsmath

```
1152 \AddToHook { package/amsmath/after }
1153 {
```

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

```
1154   \AddToHook { postnotes/note/inhibit } [ postnotes ]
1155   {
1156     \legacy_if:nT { measuring@ }
1157     {
1158       \bool_set_true:N \l__postnotes_inhibit_note_bool
1159       \bool_set_true:N \l__postnotes_print_plain_mark_bool
1160       \bool_set_true:N \l__postnotes_print_plain_mark_stepcounter_bool
1161     }
1162 }
```

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.)”

```

1163 \AddToHook { postnotes/note/begin } [ postnotes ]
1164 {
1165     \legacy_if:nF { firstchoice@ }
1166     { \bool_set_true:N \l__postnotes_maybe_multi_bool }
1167 }
1168 }

```

csquotes

```

1169 \AddToHook { package/csquotes/after }
1170 {
1171     \bool_new:N \l__postnotes_csquotes_measuring_bool
1172     \BlockquoteDisable
1173     { \bool_set_true:N \l__postnotes_csquotes_measuring_bool }
1174     \AddToHook { postnotes/note/inhibit } [ postnotes ]
1175     {
1176         \bool_if:NT \l__postnotes_csquotes_measuring_bool
1177         {
1178             \bool_set_true:N \l__postnotes_inhibit_note_bool
1179             \bool_set_true:N \l__postnotes_print_plain_mark_bool
1180             \bool_set_true:N \l__postnotes_print_plain_mark_stepcounter_bool
1181         }
1182     }
1183 }

```

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>.

```

1184 \AddToHook { package/tabularx/after }
1185 {
1186     \bool_new:N \l__postnotes_tabularx_inside_env_bool
1187     \AddToHook { env/tabularx/begin } [ postnotes ]
1188     {
1189         \bool_set_true:N \l__postnotes_tabularx_inside_env_bool
1190         \cs_set_eq:NN \__postnotes_tabularx_saved_write:Nn \write
1191     }
1192     \AddToHook { postnotes/note/inhibit } [ postnotes ]
1193     {
1194         \bool_lazy_and:nnT

```

```

1195     { \l__postnotes_tabularx_inside_env_bool }
1196     { ! \cs_if_eq_p:NN \write \__postnotes_tabularx_saved_write:Nn }
1197     {
1198         \bool_set_true:N \l__postnotes_inhibit_note_bool
1199         \bool_set_true:N \l__postnotes_print_plain_mark_bool
1200         \bool_set_true:N \l__postnotes_print_plain_mark_stepcounter_bool
1201     }
1202 }
1203 }

```

tabularray

```

1204 \AddToHook { package/tabularray/after }
1205 {

```

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/lvjlr/tabularray/issues/179> (thanks Ulrike Fischer).

```

1206     \bool_if_exist:NTF \lTblrMeasuringBool
1207     {

```

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.

```

1208     \AddToHook { postnotes/note/inhibit } [ postnotes ]
1209     {
1210         \bool_if:NT \lTblrMeasuringBool
1211         {
1212             \bool_set_true:N \l__postnotes_inhibit_note_bool
1213             \bool_set_true:N \l__postnotes_print_plain_mark_bool
1214             \bool_set_true:N \l__postnotes_print_plain_mark_stepcounter_bool
1215         }
1216     }
1217 }
1218 {

```

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

```

1219     \clist_map_inline:nn
1220     {
1221         tblr , longtblr , talltblr , booktabs ,
1222         longtabs , talltabs , +array
1223     }
1224     {
1225         \AddToHook { env/#1/begin } [ postnotes ]
1226         { \bool_set_true:N \l__postnotes_maybe_multi_bool }
1227     }
1228 }

```

```
1229 }
```

10 Languages

```
\pntitle Set of language specific user variables. They are used in the default value of the heading
\pnhdnotes option and in \pnheaderdefault which, ultimately, is also used in the same place.
\pnhdtopage
\pnhdtopages 1230 \tl_new:N \pntitle
1231 \tl_new:N \pnhdnotes
1232 \tl_new:N \pnhdtopage
1233 \tl_new:N \pnhdtopages
1234 \tl_set:Nn \pntitle { Notes }
1235 \tl_set:Nn \pnhdnotes { Notes }
1236 \tl_set:Nn \pnhdtopage { to-page }
1237 \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)}
1238 \cs_new_protected:Npn \__postnotes_define_language:nn #1#2
1239 {
1240   \tl_new:c { g__postnotes_language_ #1 _t1 }
1241   \tl_gset:cn { g__postnotes_language_ #1 _t1 } {#2}
1242 }
```

(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⟩}
1243 \cs_new_protected:Npn \__postnotes_set_babel_language:nn #1#2
1244 {
1245   \ActivateGenericHook { babel/#1/beforeextras }
1246   \exp_args:Nnv \AddToHook { babel/#1/beforeextras }
1247   { g__postnotes_language_ #2 _tl }
1248 }

```

(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⟩}
1249 \cs_new_protected:Npn \__postnotes_set_polyglossia_language:nn #1#2
1250 {
1251   \AddToHook { package/polyglossia/after }
1252   {
1253     \exp_args:Nnv \csgappto { captions #1 }
1254     { g__postnotes_language_ #2 _tl }
1255   }
1256 }

```

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

English

```

1257 \__postnotes_define_language:nn { english }
1258 {
1259   \tl_set:Nn \pntitle { Notes }
1260   \tl_set:Nn \pnhdnotes { Notes }
1261   \tl_set:Nn \pnhdtopage { to~page }
1262   \tl_set:Nn \pnhdtopages { to~pages }
1263 }
1264 \__postnotes_set_babel_language:nn { english } { english }
1265 \__postnotes_set_babel_language:nn { british } { english }
1266 \__postnotes_set_babel_language:nn { american } { english }
1267 \__postnotes_set_babel_language:nn { canadian } { english }
1268 \__postnotes_set_babel_language:nn { australian } { english }

```

```

1269 \_postnotes_set_babel_language:nn { newzealand } { english }
1270 \_postnotes_set_babel_language:nn { UKenglish } { english }
1271 \_postnotes_set_babel_language:nn { USenglish } { english }
1272 \_postnotes_set_polyglossia_language:nn { english } { english }

```

Portuguese

```

1273 \_postnotes_define_language:nn { portuguese }
1274 {
1275   \tl_set:Nn \pntitle      { Notas }
1276   \tl_set:Nn \pnhdnotes   { Notas }
1277   \tl_set:Nn \pnhdtopage  { da-página }
1278   \tl_set:Nn \pnhdtopages { das-páginas }
1279 }
1280 \_postnotes_set_babel_language:nn { portuguese } { portuguese }
1281 \_postnotes_set_babel_language:nn { brazilian } { portuguese }
1282 \_postnotes_set_babel_language:nn { portuges } { portuguese }
1283 \_postnotes_set_babel_language:nn { brazil } { portuguese }
1284 \_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`.

```

1285 \_postnotes_define_language:nn { french }
1286 {
1287   \tl_set:Nn \pntitle      { Notes }
1288   \tl_set:Nn \pnhdnotes   { Notes }
1289   \tl_set:Nn \pnhdtopage  { de-la-page }
1290   \tl_set:Nn \pnhdtopages { des-pages }
1291 }
1292 \_postnotes_set_babel_language:nn { french } { french }
1293 \_postnotes_set_babel_language:nn { acadian } { french }
1294 \_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`.

```

1295 \_postnotes_define_language:nn { german }
1296 {
1297   \tl_set:Nn \pntitle      { Endnoten }
1298   \tl_set:Nn \pnhdnotes   { Endnoten }
1299   \tl_set:Nn \pnhdtopage  { zu-Seite }
1300   \tl_set:Nn \pnhdtopages { zu-Seiten }
1301 }
1302 \_postnotes_set_babel_language:nn { german } { german }
1303 \_postnotes_set_babel_language:nn { ngerman } { german }
1304 \_postnotes_set_babel_language:nn { austrian } { german }
1305 \_postnotes_set_babel_language:nn { naustrian } { german }
1306 \_postnotes_set_babel_language:nn { swissgerman } { german }
1307 \_postnotes_set_babel_language:nn { nswissgerman } { german }

```

```

1308 \_postnotes_set_polyglossia_language:nn { german } { german }
1309 </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.

A	
\ActivateGenericHook	1245
\addto	<i>39</i>
\addtocounter	<i>37</i>
\AddToHook	80, 88, 259, 935, 960, 965, 967, 969, 977, 979, 991, 996, 1052, 1056, 1063, 1086, 1117, 1119, 1126, 1129, 1133, 1140, 1152, 1154, 1163, 1169, 1174, 1184, 1187, 1192, 1204, 1208, 1225, 1246, 1251
\AddToHookNext	667
B	
\begin	609
\BlockquoteDisable	1172
bool commands:	
\bool_gset_false:N	668
\bool_gset_true:N	551
\bool_if:NTF	34, 264, 354, 400, 404, 420, 428, 499, 549, 554, 608, 658, 691, 940, 1176, 1210
\bool_if_exist:NTF	1206
\bool_lazy_all:nTF	1100
\bool_lazy_and:nnTF	466, 678, 727, 1194
\bool_new:N	155, 232, 233, 234, 286, 364, 367, 368, 390, 391, 392, 505, 937, 1116, 1171, 1186
\bool_set_false:N	162, 241, 250, 251, 266, 396, 397, 398
\bool_set_true:N	167, 240, 245, 246, 376, 963, 1118, 1158, 1159, 1160, 1166, 1173, 1178, 1179, 1180, 1189, 1198, 1199, 1200, 1212, 1213, 1214, 1226
\bool_to_str:N	50
\bool_until_do:nn	559
box commands:	
\box_new:N	19
\box_use:N	312
\box_wd:N	311
C	
\caption	10–12, 31
\chapter	136
\citereset	<i>33</i>
clist commands:	
\clist_map_inline:Nn	1037, 1040, 1043, 1046
\clist_map_inline:nn	1219
\counterwithin	12
cs commands:	
\cs_generate_variant:Nn	22, 67, 106, 109, 112, 119, 126, 437, 688, 934
\cs_if_eq_p:NN	1196
\cs_if_exist:NTF	38, 58, 115, 122, 132, 707, 962
\cs_new:Npn	20, 73, 120, 208
\cs_new_protected:Npe	78
\cs_new_protected:Npn	24, 54, 68, 75, 96, 104, 107, 110, 113, 134, 141, 334, 424, 439, 452, 461, 491, 537, 676, 689, 698, 716, 770, 881, 938, 943, 1011, 1029, 1095, 1238, 1243, 1249
\cs_set:Npe	350, 629
\cs_set:Npn	349, 626, 971, 972
\cs_set_eq:NN	181, 198, 556, 1049, 1076, 1077, 1190
\csgappto	1253
\csundef	1038, 1041, 1044, 1047
D	
\def	3
E	
\end	659
\endgroup	21
\endlist	190, 207
\endnotemark	11
\endnotetext	11
\endrefcontext	32
\enoteheading	19
exp commands:	
\exp_args:Ne	609, 659, 801
\exp_args:NNNe	412
\exp_args:Nnv	1246, 1253
\exp_args:NV	455, 457
\exp_not:N	79
\exp_not:n	123

F		K	
\fmtversion	5	keys commands:	
\footnote	37	\keys_define:nn	127, 148, 156, 210, 223, 235, 268, 287, 293, 369, 506, 1088
\footnotemark	10–12	\keys_set:nn	325, 337, 498
\footnotesize	301	L	
\footnotetext	10, 11, 973	\label	12, 455, 1076
fp commands:		\labelep	209
\fp_compare:nNnTF	733	\labelwidth	179, 196
\fp_new:N	365	\leftmargin	178, 195, 196, 197
\fp_set:Nn	375	\leftskip	303
\fp_use:N	35	legacy commands:	
G		\legacy_if:nTF	82, 90, 98, 1156, 1165
\gappto	39	\list	176, 193
group commands:		\listparindent	183, 200
\group_begin:	336, 410, 463, 493, 539, 582, 610, 613, 700, 718, 772, 883, 1075, 1097	\lTblrMeasuringBool	38, 1206, 1210
\group_end:	361, 413, 475, 502, 600, 643, 660, 672, 714, 742, 879, 932, 1080, 1113	M	
H		\makeatletter	40
\hbox	215	\makeatother	40
hbox commands:		\makelabel	181, 198
\hbox_set:Nn	309	\MakeLinkTarget	2, 351, 633
\hspace	209	\mathchoice	11, 37
\hyperlink	1107	\mbox	10
\hyperref	470	\mkbibendnote	31
I		mode commands:	
\IfFormatAtLeastTF	5, 6	\mode_if_horizontal:TF	442, 448
\IfPackageAtLeastTF	1131	\mode_leave_vertical:	441, 694
\IfPackageLoadedTF	261	msg commands:	
\iftoggle	987	\msg_line_context:	283, 487, 675
int commands:		\msg_new:nnn	282, 284, 485, 674
\int_eval:n	941	\msg_warning:nn	265, 482, 542
\int_gincr:N	340, 494, 495, 540	\msg_warning:nnn	272, 277
\int_incr:N	411	N	
\int_new:N	327, 490, 522	\NeedsTeXFormat	4
\int_set:Nn	627, 1000, 1003, 1071, 1144, 1147	\newcounter	326, 534, 535
\int_use:N	31, 36, 48, 108, 111, 329, 946, 950, 982, 984, 1031, 1032, 1059, 1136, 1138	\NewDocumentCommand	324, 331, 459, 477, 479, 514, 953, 1093
iow commands:		\NewDocumentEnvironment	174, 191
\iow_char:N	487, 488, 675	\NewHook	2, 23, 333, 393, 532, 533
\iow_now:Nn	84, 92	\newlabel	4
\iow_shipout_x:Nn	5, 100	\NewTblrEnviron	38
\item	21, 23, 693	\nobreak	445
\itemindent	180, 197	\noindent	319
\itemsep	185, 202	\normalfont	209, 215, 310, 320
P		P	
		\PackageError	9
		\par	22, 163, 306, 319, 661
		\parindent	180, 183, 200, 304
		\parsep	184, 201
		\parskip	184, 201
		\partopsep	187, 204

`\pnhdchapfirst` 744, 775, 898, 899
`\pnhdchaplast` 744, 776, 903, 907
`\pnhdnamefirst` 744, 779, 922, 923
`\pnhdnamelast` 744, 780, 927, 931
`\pnhdnotes`
956, 957, 1230, 1260, 1276, 1288, 1298
`\pnhdpagefirst`
. 744, 773, 886, 887, 955, 956, 957
`\pnhdpagelast` . 744, 774, 891, 895, 955, 957
`\pnhdsectfirst` 744, 777, 910, 911
`\pnhdsectlast` 744, 778, 915, 919
`\pnhdtopage` 956, 1230, 1261, 1277, 1289, 1299
`\pnhdtopages`
. 957, 1230, 1262, 1278, 1290, 1300
`\pnheaderdefault` . . . 30, 39, 137, 144, 953
`\pnheading` 6, 129, 132, 544
`\pnidnextnote` 516, 589
`\pnthechapter` 516, 585
`\pnthechapternextnote` 516, 592
`\pnthepage` 516, 615
`\pnthesection` 516, 588
`\pnthesectionnextnote` 516, 595
`\pntitle`
136, 143, 1230, 1259, 1275, 1287, 1297
`\postnote` 2, 3, 10,
12–14, 16, 23, 24, 31, 37, 38, 331, 971
`\postnotemark` 11, 12
`\postnoteref` 12, 16, 37, 459
postnotes commands:
`\l_postnotes_note_id_tl` 12,
327, 347, 351, 352, 357, 359, 496,
500, 501, 981, 983, 985, 988, 1135, 1137
`\l_postnotes_print_note_id_tl`
. 522, 562, 563, 584, 587, 597,
616, 618, 621, 624, 634, 636, 638,
998, 1001, 1004, 1007, 1065, 1142, 1145
postnotes internal commands:
`\l__postnotes_backlink_bool`
. 234, 255, 680
`__postnotes_biblatex_citereset_-
local:` 994, 1029, 1029
`__postnotes_biblatex_endrefcontext_-
local:` 993, 1011, 1011
`\l__postnotes_biblatex_orig_-
refsection_tl` . 34, 1054, 1058, 1061
`\g__postnotes_biblatex_prev_-
refsection_tl` 1055, 1060, 1069, 1073
`\l__postnotes_clear_queue_seq`
. 22, 522, 547, 669
`\l__postnotes_csquotes_measuring_-
bool` 1171, 1173, 1176
`\l__postnotes_curr_text_page_tl`
. 28, 752, 794,
806, 807, 811, 839, 842, 845, 848, 859
`__postnotes_data_name:n` . . . 2, 12,
20, 20, 22, 26, 27, 28, 30, 32, 40, 43,
45, 47, 49, 51, 56, 57, 60, 63, 65, 70,
74, 76, 981, 983, 985, 988, 1135, 1137
`__postnotes_define_language:nm`
. 39,
40, 1238, 1238, 1257, 1273, 1285, 1295
`__postnotes_extract_pageref:n`
. 5, 113, 120, 126, 840, 949
`__postnotes_get_headers_data:N`
. 25, 26, 28, 552, 770, 770
`__postnotes_get_pageref:Nn`
. 5, 113, 113, 119, 615, 805, 813, 850
`\g__postnotes_header_chap_first_-
prop` 752, 783, 841, 896
`\g__postnotes_header_chap_last_-
prop` 752, 784, 828, 869, 900
`\g__postnotes_header_name_first_-
prop` 752, 787, 847, 920
`\g__postnotes_header_name_last_-
prop` 752, 788, 834, 875, 924
`\g__postnotes_header_page_first_-
prop` 752, 781, 838, 884
`\g__postnotes_header_page_last_-
prop` 752, 782, 825, 866, 888
`\g__postnotes_header_prev_last_-
chap_tl` 752, 790, 899, 904, 907
`\g__postnotes_header_prev_last_-
name_tl` 752, 792, 923, 928, 931
`\g__postnotes_header_prev_last_-
page_tl` 752, 789, 887, 892, 895
`\g__postnotes_header_prev_last_-
sect_tl` 752, 791, 911, 916, 919
`\g__postnotes_header_sect_first_-
prop` 752, 785, 844, 908
`\g__postnotes_header_sect_last_-
prop` 752, 786, 831, 872, 912
`\g__postnotes_header_vars_next_-
bool` 30, 551, 668, 937, 940
`\l__postnotes_hyperlink_bool` 232,
240, 245, 250, 266, 428, 468, 679, 1103
`\l__postnotes_hyperref_warn_bool`
. 233, 241, 246, 251, 264
`__postnotes_inhibit_note:` 394
`__postnotes_inhibit_note:TF`
. 23, 338, 390
`\l__postnotes_inhibit_note_bool`
. 14,
390, 396, 420, 1158, 1178, 1198, 1212
`__postnotes_list_makelabel:n`
. 181, 198, 208
`__postnotes_make_mark:nmn` . 212,
418, 430, 434, 471, 473, 1109, 1111

`__postnotes_make_text_mark:nnn` 216, 682, 686
`\l__postnotes_manual_sortnum_`
 `bool` 34, 367, 376
`\l__postnotes_mark_tl` 29, 341, 344,
 350, 357, 363, 371, 402, 407, 414, 418
`\l__postnotes_maybe_multi_bool` . .
 23, 37, 50, 368, 963, 1166, 1226
`\l__postnotes_nomark_bool`
 354, 364, 385
`__postnotes_note:nn`
 13, 15, 16, 332, 333, 334
`\g__postnotes_note_id_int`
 12, 327, 340, 495
`\l__postnotes_note_label_tl`
 366, 387, 454, 455
`__postnotes_note_ref:nn`
 16, 460, 461, 461
`\l__postnotes_note_zlabel_tl`
 35, 456, 457, 1085, 1090
`__postnotes_note_zref:nn`
 35, 1094, 1095, 1095
`\l__postnotes_post_printnote_tl`
 163, 222, 229, 642
`\l__postnotes_post_textmark_tl`
 221, 227, 693, 696
`\l__postnotes_pre_textmark_tl`
 220, 225, 693, 696
`\l__postnotes_prev_mark_chap_tl`
 752, 796, 816, 830, 853, 871
`\l__postnotes_prev_mark_name_tl`
 752, 798, 820, 836, 857, 877
`\l__postnotes_prev_mark_page_tl`
 752, 795, 814, 827, 851, 868
`\l__postnotes_prev_mark_sect_tl`
 752, 797, 818, 833, 855, 874
`\l__postnotes_prev_text_page_tl`
 27, 28, 752, 793, 810, 823, 826, 829,
 832, 835, 858, 864, 867, 870, 873, 876
`\l__postnotes_print_as_list_bool`
 155, 162, 167, 554, 608, 658, 691
`\l__postnotes_print_content_tl`
 522, 598, 599, 625, 641
`\l__postnotes_print_counter_tl`
 522, 622, 628
`\l__postnotes_print_env_tl`
 154, 164, 168, 609, 659
`\l__postnotes_print_format_tl`
 147, 150, 611
`\l__postnotes_print_mark_tl`
 522, 619, 630, 639
`\l__postnotes_print_note_id_`
 `next_tl` 522, 569,
 574, 576, 589, 591, 594, 646, 651, 653
`__postnotes_print_notes:`
 18, 19, 22, 24, 25, 515, 537, 537
`\l__postnotes_print_plain_mark_`
 `bool` 14,
 391, 397, 400, 1159, 1179, 1199, 1213
`\l__postnotes_print_plain_mark_`
 `stepcounter_bool` 14, 38,
 392, 398, 404, 1160, 1180, 1200, 1214
`\g__postnotes_print_postnotes_`
 `int` 522, 540, 946, 950
`\l__postnotes_print_type_curr_tl`
 522, 564, 565, 602, 664
`\l__postnotes_print_type_next_tl`
 522, 570, 577, 579, 647, 654, 656
`\l__postnotes_print_type_prev_tl`
 522, 546, 601, 606, 663
`__postnotes_prop_gclear:n`
 4, 68, 75, 670
`__postnotes_prop_get:nnN`
 4, 68, 68, 563, 575, 583,
 586, 590, 593, 596, 617, 620, 623,
 652, 704, 721, 722, 725, 726, 731,
 732, 815, 817, 819, 852, 854, 856,
 998, 1001, 1004, 1007, 1065, 1142, 1145
`__postnotes_prop_item:nn`
 4, 68, 73, 802, 843, 846, 849
`\g__postnotes_queue_seq` 12, 23,
 24, 26, 327, 346, 496, 541, 547, 548,
 550, 552, 559, 561, 567, 573, 644, 650
`\c__postnotes_ref_prefix_tl`
 4, 77, 79, 115, 116, 122, 123, 708
`\l__postnotes_restore_tmp_tl`
 959, 999, 1000, 1002, 1003,
 1005, 1006, 1008, 1009, 1066, 1068,
 1072, 1074, 1143, 1144, 1146, 1147
`\l__postnotes_saved_spacefactor_`
 `tl` 438, 444, 449
`\g__postnotes_sectid_int` 48, 490, 494
`__postnotes_section:nn`
 17, 478, 490, 491
`\l__postnotes_section_exp_bool`
 499, 505, 510
`\g__postnotes_section_name_tl`
 46, 497, 504, 508
`__postnotes_set_babel_language:nn`
 40, 1243, 1243, 1264, 1265,
 1266, 1267, 1268, 1269, 1270, 1271,
 1280, 1281, 1282, 1283, 1292, 1293,
 1302, 1303, 1304, 1305, 1306, 1307
`__postnotes_set_headers_vars:n`
 26, 28, 30, 881, 881, 934, 941, 947
`__postnotes_set_headers_vars_`
 `first:` 25, 30, 553, 935, 943

751, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 959, 1054, 1055, 1085, 1230, 1231, 1232, 1233, 1240	
\tl_set:Nn	116, 163, 164, 168, 329, 344, 407, 414, 444, 546, 569, 570, 589, 601, 646, 647, 663, 858, 1016, 1017, 1018, 1019, 1058, 1234, 1235, 1236, 1237, 1259, 1260, 1261, 1262, 1275, 1276, 1277, 1278, 1287, 1288, 1289, 1290, 1297, 1298, 1299, 1300
\togglefalse	1013
\toggletrue	968
token commands:	
\token_to_str:N	85, 93, 101
\topsep	186, 203
	U
\undef	1033, 1034, 1035, 1036
use commands:	
\use:N	1006
\use_none:n	1076, 1077
\UseHook	52, 348, 399, 545, 614
	W
\write	1190, 1196
	Z
\zcsetup	1121, 1127
\zlabel	457
\zref	1109, 1111