

The `postnotes` package^{*}

Code documentation

Gustavo Barros[†]

2022-04-21

Contents

1	Initial setup	2
2	Data	2
3	Options	5
4	<code>\postnote</code>	9
5	<code>\postnoteref</code>	15
6	<code>\postnotesection</code>	16
7	<code>\printpostnotes</code>	17
8	Headers	23
9	Compatibility	29
10	Languages	36
	Index	39

^{*}This file describes v0.1.1, released 2022-04-21.

[†]<https://github.com/gusbrs/postnotes>

1 Initial setup

Start the DocStrip guards.

```

1 <*package>
   Identify the internal prefix (LATEX3 DocStrip convention).
2 <@=postnotes>
   Require the new syntax for file/package hooks (ltnews34, ltfilehook).
3 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
4 \IfFormatAtLeastTF{2021-11-15}
5 {}
6 {%
7   \PackageError{postnotes}{LaTeX kernel too old}
8   {%
9     'postnotes' requires a LaTeX kernel 2021-11-15 or newer.%
10    \MessageBreak Loading will abort!%
11   }%
12 \endinput
13 }%
14 \ProvidesExplPackage {postnotes} {2022-04-21} {0.1.1}
15 {Endnotes for LaTeX}

```

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>}

16 \cs_new:Npn \__postnotes_data_name:n #1
17 { g__postnotes_ #1 _data_prop }
18 \cs_generate_variant:Nn \__postnotes_data_name:n { e }

```

(End definition for `__postnotes_data_name:n`.)

`__postnotes_store:nn` Stores the metadata and `<note content>` of `\postnote` with ID `<note id>`, from where it is called. The `postnotes/store/note` 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>}

19 \NewHook { postnotes/store/note }
20 \cs_new_protected:Npn \__postnotes_store:nn #1#2
21 {
22   \prop_new:c { \__postnotes_data_name:e {#1} }
23   \prop_gput:cnn { \__postnotes_data_name:e {#1} } { type } { note }
24   \prop_gput:cnx { \__postnotes_data_name:e {#1} } { mark }
25     { \l__postnotes_mark_tl }
26   \prop_gput:cnx { \__postnotes_data_name:e {#1} } { counter }
27     { \int_use:N \c@postnote }
28   \prop_gput:cnx { \__postnotes_data_name:e {#1} } { sortnum }
29   {

```

```

30         \bool_if:NTF \l__postnotes_manual_sortnum_bool
31         { \fp_use:N \l__postnotes_sort_num_fp }
32         { \int_use:N \c@postnote }
33     }
34     \cs_if_exist:cT { chapter }
35     {
36         \prop_gput:cnx { \__postnotes_data_name:e {#1} }
37         { thechapter } { \thechapter }
38     }
39     \prop_gput:cnx { \__postnotes_data_name:e {#1} } { thesection }
40     { \thesection }
41     \prop_gput:cnx { \__postnotes_data_name:e {#1} } { pnsectname }
42     { \g__postnotes_section_name_tl }
43     \prop_gput:cnx { \__postnotes_data_name:e {#1} } { pnsectid }
44     { \int_use:N \g__postnotes_sectid_int }
45     \prop_gput:cnx { \__postnotes_data_name:e {#1} } { multibool }
46     { \bool_to_str:N \l__postnotes_maybe_multi_bool }
47     \prop_gput:cnn { \__postnotes_data_name:e {#1} } { content } {#2}
48     \UseHook { postnotes/store/note }
49 }

```

(End definition for __postnotes_store:nn.)

__postnotes_store_section:nn Stores the metadata and $\langle note\ content \rangle$ of \postnotessection with ID $\langle note\ id \rangle$, from where it is called.

```

        \__postnotes_store_section:nn {\langle note id \rangle} {\langle note content \rangle}
50 \cs_new_protected:Npn \__postnotes_store_section:nn #1#2
51 {
52     \prop_new:c { \__postnotes_data_name:e {#1} }
53     \prop_gput:cnn { \__postnotes_data_name:e {#1} } { type } { section }
54     \cs_if_exist:cT { chapter }
55     {
56         \prop_gput:cnx { \__postnotes_data_name:e {#1} }
57         { thechapter } { \thechapter }
58     }
59     \prop_gput:cnx { \__postnotes_data_name:e {#1} } { thesection }
60     { \thesection }
61     \prop_gput:cnn { \__postnotes_data_name:e {#1} } { content } {#2}
62 }

```

(End 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 {\langle note id \rangle} {\langle property \rangle} {\langle tl var to set \rangle}
        \__postnotes_prop_item:nn {\langle note id \rangle} {\langle property \rangle}
        \__postnotes_prop_gclear:n {\langle note id \rangle}
63 \cs_new_protected:Npn \__postnotes_prop_get:nnN #1#2#3
64 {
65     \prop_get:cnNF { \__postnotes_data_name:e {#1} } {#2} #3
66     { \tl_clear:N #3 }
67 }

```

```

68 \cs_new:Npn \__postnotes_prop_item:nn #1#2
69 { \prop_item:cn { \__postnotes_data_name:e {#1} } {#2} }
70 \cs_new_protected:Npn \__postnotes_prop_gclear:n #1
71 { \prop_gclear:c { \__postnotes_data_name:e {#1} } }

```

(End 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>}

72 \tl_const:Nn \c__postnotes_ref_prefix_tl { postnote@r }
73 \cs_new_protected:Npx \post@note #1#2
74 { \exp_not:N \@newl@bel { \c__postnotes_ref_prefix_tl } {#1} {#2} }

```

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

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

```

\__postnotes_set_mark_page_label:n {<label name>}
\__postnotes_set_text_page_label:n {<label name>}
\__postnotes_set_print_page_label:n {<label name>}

75 \cs_new_protected:Npn \__postnotes_set_mark_page_label:n #1
76 {
77   \iow_shipout_x:Nn \@auxout
78   { \post@note { mark@ #1 } { \thepage } }
79 }
80 \cs_generate_variant:Nn \__postnotes_set_mark_page_label:n { x }
81 \cs_new_protected:Npn \__postnotes_set_text_page_label:n #1
82 {
83   \iow_shipout_x:Nn \@auxout
84   { \post@note { text@ #1 } { \int_use:N \c@page } }
85 }
86 \cs_generate_variant:Nn \__postnotes_set_text_page_label:n { x }
87 \cs_new_protected:Npn \__postnotes_set_print_page_label:n #1
88 {
89   \iow_shipout_x:Nn \@auxout
90   { \post@note { print@ #1 } { \int_use:N \c@page } }
91 }
92 \cs_generate_variant:Nn \__postnotes_set_print_page_label:n { x }

```

(End definition for `__postnotes_set_mark_page_label:n`, `__postnotes_set_text_page_label:n`, and `__postnotes_set_print_page_label:n`.)

`__postnotes_get_pageref:Nn` `__postnotes_extract_pageref:n` Reference data extraction functions.

```

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

```

```

93 \cs_new_protected:Npn \__postnotes_get_pageref:Nn #1#2
94 {
95   \cs_if_exist:cTF { \c__postnotes_ref_prefix_tl @ #2 }
96   { \tl_set:Nv #1 { \c__postnotes_ref_prefix_tl @ #2 } }
97   { \tl_clear:N #1 }
98 }
99 \cs_generate_variant:Nn \__postnotes_get_pageref:Nn { Nx }
100 \cs_new:Npn \__postnotes_extract_pageref:n #1
101 {
102   \cs_if_exist:cTF { \c__postnotes_ref_prefix_tl @ #1 }
103   { \exp_not:v { \c__postnotes_ref_prefix_tl @ #1 } }
104   { \c_empty_tl }
105 }
106 \cs_generate_variant:Nn \__postnotes_extract_pageref:n { e }

```

(End definition for __postnotes_get_pageref:Nn and __postnotes_extract_pageref:n.)

3 Options

heading option

```

107 \keys_define:nn { postnotes/setup }
108 {
109   heading .cs_set_protected:Np = \pnheading ,
110   heading .value_required:n = true ,
111 }

```

\pnheading Provide default value for \pnheading.

```

112 \cs_if_exist:cTF { chapter }
113 {
114   \cs_new_protected:Npn \pnheading
115   {
116     \chapter*{\pntitle}
117     \@mkboth{\pnheaderdefault}{\pnheaderdefault}
118   }
119 }
120 {
121   \cs_new_protected:Npn \pnheading
122   {
123     \section*{\pntitle}
124     \@mkboth{\pnheaderdefault}{\pnheaderdefault}
125   }
126 }

```

(End definition for \pnheading.)

format option

```

127 \tl_new:N \l__postnotes_print_format_tl
128 \keys_define:nn { postnotes/setup }
129 {
130   format .tl_set:N = \l__postnotes_print_format_tl ,
131   format .initial:n = { \small } ,

```

```

132     format .value_required:n = true ,
133 }

```

listenv option

```

134 \tl_new:N \l__postnotes_print_env_tl
135 \bool_new:N \l__postnotes_print_as_list_bool
136 \keys_define:nn { postnotes/setup }
137 {
138     listenv .code:n =
139     {
140         \tl_if_eq:nnTF {#1} { none }
141         {
142             \bool_set_false:N \l__postnotes_print_as_list_bool
143             \tl_set:Nn \l__postnotes_post_printnote_tl { \par }
144             \tl_set:Nn \l__postnotes_print_env_tl { itemize }
145         }
146         {
147             \bool_set_true:N \l__postnotes_print_as_list_bool
148             \tl_set:Nn \l__postnotes_print_env_tl {#1}
149         }
150     } ,
151     listenv .initial:n = { postnoteslist } ,
152     listenv .value_required:n = true ,
153 }

```

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

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*).

```

154 \NewDocumentEnvironment { postnoteslist } { }
155 {
156     \list { }
157     {
158         \setlength { \leftmargin } { 0pt }
159         \setlength { \labelwidth } { 0pt }
160         \setlength { \itemindent } { .5\parindent }
161         \cs_set_eq:NN \makelabel \__postnotes_list_makelabel:n
162         \setlength { \rightmargin } { 0pt }
163         \setlength { \listparindent } { \parindent }
164         \setlength { \parsep } { \parskip }
165         \setlength { \itemsep } { 0pt }
166         \setlength { \topsep } { .5\topsep }
167         \setlength { \partopsep } { .5\partopsep }
168     }
169 }
170 { \endlist }
171 \NewDocumentEnvironment { postnoteslisthang } { }
172 {
173     \list { }
174     {
175         \setlength { \leftmargin } { 1em }
176         \setlength { \labelwidth } { -\leftmargin }
177         \setlength { \itemindent } { -2\leftmargin }
178         \cs_set_eq:NN \makelabel \__postnotes_list_makelabel:n

```

```

179         \setlength { \rightmargin } { Opt }
180         \setlength { \listparindent } { \parindent }
181         \setlength { \parsep } { \parskip }
182         \setlength { \itemsep } { Opt }
183         \setlength { \topsep } { .5\topsep }
184         \setlength { \partopsep } { .5\partopsep }
185     }
186 }
187 { \endlist }
188 \cs_new:Npn \__postnotes_list_makelabel:n #1
189 { \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.

```

190 \keys_define:nn { postnotes/setup }
191 {
192     makemark .cs_set:Np = \__postnotes_make_mark:nnn #1#2#3 ,
193     makemark .value_required:n = true ,

```

From the default kernel definition of \@makefnmark.

```

194     makemark .initial:n =
195     { #2 \hbox { \@textsuperscript { \normalfont #1 } } #3 } ,
196     maketextmark .cs_set:Np = \__postnotes_make_text_mark:nnn #1#2#3 ,
197     maketextmark .value_required:n = true ,
198     maketextmark .initial:n = { #2 #1 . #3 } ,
199 }

```

pretextmark, posttextmark, postprintnote options

```

200 \tl_new:N \l__postnotes_pre_textmark_tl
201 \tl_new:N \l__postnotes_post_textmark_tl
202 \tl_new:N \l__postnotes_post_printnote_tl
203 \keys_define:nn { postnotes/setup }
204 {
205     pretextmark .tl_set:N = \l__postnotes_pre_textmark_tl ,
206     pretextmark .value_required:n = true ,
207     posttextmark .tl_set:N = \l__postnotes_post_textmark_tl ,
208     posttextmark .value_required:n = true ,
209     postprintnote .tl_set:N = \l__postnotes_post_printnote_tl ,
210     postprintnote .value_required:n = true ,
211 }

```

hyperref and backlink options

```

212 \bool_new:N \l__postnotes_hyperlink_bool
213 \bool_new:N \l__postnotes_hyperref_warn_bool
214 \bool_new:N \l__postnotes_backlink_bool
215 \keys_define:nn { postnotes/setup }
216 {
217     hyperref .choice: ,
218     hyperref / auto .code:n =
219     {
220         \bool_set_true:N \l__postnotes_hyperlink_bool

```

```

221         \bool_set_false:N \l__postnotes_hyperref_warn_bool
222     } ,
223     hyperref / true .code:n =
224     {
225         \bool_set_true:N \l__postnotes_hyperlink_bool
226         \bool_set_true:N \l__postnotes_hyperref_warn_bool
227     } ,
228     hyperref / false .code:n =
229     {
230         \bool_set_false:N \l__postnotes_hyperlink_bool
231         \bool_set_false:N \l__postnotes_hyperref_warn_bool
232     } ,
233     hyperref .initial:n = auto ,
234     hyperref .default:n = true ,
235     backlink .bool_set:N = \l__postnotes_backlink_bool ,
236     backlink .initial:n = true ,
237     backlink .default:n = true ,
238 }
239 \AddToHook { begindocument }
240 {
241     \IfPackageLoadedTF { hyperref }
242     { }
243     {
244         \bool_if:NT \l__postnotes_hyperref_warn_bool
245         { \msg_warning:nn { postnotes } { missing-hyperref } }
246         \bool_set_false:N \l__postnotes_hyperlink_bool
247     }
248     \keys_define:nn { postnotes/setup }
249     {
250         hyperref .code:n =
251         {
252             \msg_warning:nnn { postnotes }
253             { option-preamble-only } { hyperref }
254         } ,
255         backlink .code:n =
256         {
257             \msg_warning:nnn { postnotes }
258             { option-preamble-only } { backlink }
259         } ,
260     }
261 }
262 \msg_new:nnn { postnotes } { option-preamble-only }
263 { Option~'#1'~only~available~in~the~preamble~\msg_line_context:. }
264 \msg_new:nnn { postnotes } { missing-hyperref }
265 { Missing~'hyperref'~package.~Setting~'hyperref=false'. }

```

sort option

```

266 \bool_new:N \l__postnotes_sort_bool
267 \keys_define:nn { postnotes/setup }
268 {
269     sort .bool_set:N = \l__postnotes_sort_bool ,
270     sort .initial:n = true ,
271     sort .default:n = true ,
272 }

```


style option

```

273 \keys_define:nn { postnotes/setup }
274 {
275   style .choice: ,
276   style / endnotes .meta:n =
277   {
278     listenv = none ,
279     format =
280     {
281       \footnotesize
282       \setlength { \rightskip } { 0pt }
283       \setlength { \leftskip } { 0pt }
284       \setlength { \parindent } { 1.8em }
285     } ,
286     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.

```

287     maketextmark =
288     {
289       \hbox_set:Nn \l_tmpa_box { \@textsuperscript { \normalfont ##1 } }
290       \skip_horizontal:n { - \box_wd:N \l_tmpa_box }
291       ##2 \box_use:N \l_tmpa_box ##3
292     } ,
293   } ,
294   style / pagenote .meta:n =
295   {
296     listenv = none ,
297     format = { } ,
298     pretextmark = { \par\noindent } ,
299     maketextmark = { { \normalfont ##2 ##1 . ##3 } } ,
300     posttextmark = { ~ } ,
301   } ,
302 }

```

\postnotesetup

\postnotesetup Provide \postnotesetup.

```

\postnotesetup{options}

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

```

(End 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 `tabulararray` also perform measuring passes of their tables; so does `csquotes`’ blockquotes; and certainly more that I’m unaware of. A number of these places offer some one or another way to mitigate the issue: `amsmath`, `tabularx`, `csquotes` and (optionally) `tabulararray` 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 has 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 *⟨label⟩* which we could use to identify their counterparts, but I was not able to go through with this, and the attempts all increased complexity considerably. It is not just a label/ref system, there’s got to be a one-to-one correspondence between the sets, uniqueness has to be ensured on both sides, and there cannot be “lone” marks or texts (a bijection). Besides, this label based system of identification would have to live side-by-side with the one based on the counter. So, even if we’d have unique IDs, we wouldn’t know beforehand in what form it comes. Considering the ID is used to build the variable name in which we store the note’s information, this would also complicate things.

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

The truth is there are some trade-offs, there’s no “ideal” solution. Still, all in all, my judgment is that the unique ID counter is worth more than the inconveniences of an 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.

```
305 \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 that to be processed by the next call of `\printpostnotes`.

```
306 \int_new:N \g__postnotes_note_id_int
307 \tl_new:N \l__postnotes_note_id_tl
308 \tl_set:Nn \l__postnotes_note_id_tl { \int_use:N \g__postnotes_note_id_int }
309 \seq_new:N \g__postnotes_queue_seq
```

(End definition for `\g__postnotes_note_id_int`, `\l__postnotes_note_id_tl`, and `\g__postnotes_queue_seq`.)

`\postnote` Provide `\postnote`.

```
\postnote [<options>] {<note text>}
```

```
310 \NewDocumentCommand \postnote { 0 { } +m }
311 { \__postnotes_note:nn {#1} {#2} }
```

(End 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>}
```

```
312 \NewHook { postnotes/note/begin }
313 \cs_new_protected:Npn \__postnotes_note:nn #1#2
314 {
315   \group_begin:
316   \keys_set:nn { postnotes/note } {#1}
317   \__postnotes_inhibit_note:F
318   {
319     \int_gincr:N \g__postnotes_note_id_int
320     \tl_if_empty:NT \l__postnotes_mark_tl
321     {
322       \stepcounter { postnote }
```

```

323         \tl_set:Nx \l__postnotes_mark_tl { \thepostnote }
324     }
325     \seq_gput_right:Nx \g__postnotes_queue_seq
326     { \l__postnotes_note_id_tl }
327     \UseHook { postnotes/note/begin }
328     \cs_set:Npn \@currentcounter { postnote }
329     \cs_set:Npx \@currentlabel { \p@postnote \l__postnotes_mark_tl }
330     \__postnotes_hyperref_make_currentHref:n
331     { postnote. \l__postnotes_note_id_tl .mark }
332     \__postnotes_set_mark_page_label:x { \l__postnotes_note_id_tl }
333     \__postnotes_set_user_labels:
334     \bool_if:NTF \l__postnotes_nomark_bool
335     {
336         \bool_if:NT \l__postnotes_hyperlink_bool
337         {
338             \__postnotes_hyperref_set_anchor:n
339             { postnote. \l__postnotes_note_id_tl .mark }
340         }
341     }
342     {
343         \__postnotes_typeset_mark:xV
344         { \l__postnotes_note_id_tl } \l__postnotes_mark_tl
345     }
346     \__postnotes_store:nn { \l__postnotes_note_id_tl } {#2}
347 }
348 \group_end:
349 }

```

(End definition for `__postnotes_note:nn`.)

Options for `\postnote`.

```

350 \tl_new:N \l__postnotes_mark_tl
351 \bool_new:N \l__postnotes_nomark_bool
352 \fp_new:N \l__postnotes_sort_num_fp
353 \tl_new:N \l__postnotes_note_label_tl
354 \bool_new:N \l__postnotes_manual_sortnum_bool
355 \bool_new:N \l__postnotes_maybe_multi_bool
356 \keys_define:nn { postnotes/note }
357 {
358     mark .tl_set:N = \l__postnotes_mark_tl ,
359     mark .value_required:n = true ,
360     nomark .bool_set:N = \l__postnotes_nomark_bool ,
361     nomark .default:n = true ,
362     sortnum .code:n =
363     {
364         \fp_set:Nn \l__postnotes_sort_num_fp {#1}
365         \bool_set_true:N \l__postnotes_manual_sortnum_bool
366     } ,
367     sortnum .value_required:n = true ,
368     label .tl_set:N = \l__postnotes_note_label_tl ,
369     label .value_required:n = true ,
370 }

```

`__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` and `\l__postnotes_print_plain_mark_bool` as appropriate to the case.

```

371 \bool_new:N \l__postnotes_inhibit_note_bool
372 \bool_new:N \l__postnotes_print_plain_mark_bool
373 \NewHook { postnotes/note/inhibit }
374 \prg_new_protected_conditional:Npnn \__postnotes_inhibit_note: { F }
375 {
376   \bool_set_false:N \l__postnotes_inhibit_note_bool
377   \bool_set_false:N \l__postnotes_print_plain_mark_bool
378   \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. Since we’d only print this mark in case of inhibition, when we don’t actually step the counter, 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.

```

379   \bool_if:NT \l__postnotes_print_plain_mark_bool
380   {
381     \tl_if_empty:NT \l__postnotes_mark_tl
382     {
383       \group_begin:
384       \int_incr:N \c@postnote
385       \exp_args:NNNx
386       \group_end:
387       \tl_set:Nn \l__postnotes_mark_tl { \thepostnote }
388     }
389     \__postnotes_typeset_mark_wrapper:n
390     { \__postnotes_make_mark:nnn { \l__postnotes_mark_tl } { } { } }
391   }
392   \bool_if:NTF \l__postnotes_inhibit_note_bool
393   { \prg_return_true: }
394   { \prg_return_false: }
395 }

```

(End 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>}

396 \cs_new_protected:Npn \__postnotes_typeset_mark:nn #1#2
397 {
398   \__postnotes_typeset_mark_wrapper:n
399   {
400     \bool_if:NTF \l__postnotes_hyperlink_bool
401     {
402       \__postnotes_hyperref_set_anchor:n { postnote. #1 .mark }
403       \__postnotes_make_mark:nnn {#2}

```

```

404         { \hyper@linkstart { link } { postnote. #1 .text } }
405         { \hyper@linkend }
406     }
407     { \__postnotes_make_mark:nnn {#2} { } { } }
408 }
409 }
410 \cs_generate_variant:Nn \__postnotes_typeset_mark:nn { xV }
411 \tl_new:N \l__postnotes_saved_spacefactor_tl
412 \cs_new_protected:Npn \__postnotes_typeset_mark_wrapper:n #1
413 {
414     \mode_leave_vertical:
415     \mode_if_horizontal:T
416     {
417         \tl_set:Nx \l__postnotes_saved_spacefactor_tl { \the\spacefactor }
418         \nobreak
419     }
420     #1
421     \mode_if_horizontal:T
422     { \spacefactor \l__postnotes_saved_spacefactor_tl }
423     \scan_stop:
424 }

```

(End 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`.

```

425 \cs_new_protected:Npn \__postnotes_set_user_labels:
426 {
427     \tl_if_empty:NF \l__postnotes_note_label_tl
428     { \exp_args:NV \label \l__postnotes_note_label_tl }
429     \tl_if_empty:NF \l__postnotes_note_zlabel_tl
430     { \exp_args:NV \zlabel \l__postnotes_note_zlabel_tl }
431 }

```

(End definition for __postnotes_set_user_labels:.)

5 \postnoteref

`\postnoteref` Provide `\postnoteref`.

```

\postnoteref{<*>}{<label>}

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

```

(End definition for \postnoteref.)

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

```

\__postnotes_note_ref:nn {<star bool>} {<label>}

```

```

434 \cs_new_protected:Npn \__postnotes_note_ref:nn #1#2
435 {
436   \group_begin:
437   \__postnotes_typeset_mark_wrapper:n
438   {
439     \bool_lazy_and:nnTF
440     { ! #1 }
441     { \l__postnotes_hyperlink_bool }
442     {
443       \hyperref [#2]
444       { \__postnotes_make_mark:nnn { \ref*{#2} } { } { } }
445     }
446     { \__postnotes_make_mark:nnn { \__postnotes_ref_star:n {#2} } { } { } }
447   }
448   \group_end:
449 }

```

(End definition for __postnotes_note_ref:nn.)

6 \postnotessection

\postnotessection Provide \postnotessection.

```

\postnotessection[<options>]{<section content>}

450 \NewDocumentCommand \postnotessection { 0 { } +m }
451 { \__postnotes_section:nn {#1} {#2} }

```

(End definition for \postnotessection.)

__postnotes_section:nn The internal version of \postnotessection.

```

\__postnotes_section:nn {<options>} {<content>}

452 \int_new:N \g__postnotes_sectid_int
453 \cs_new_protected:Npn \__postnotes_section:nn #1#2
454 {
455   \group_begin:
456   \int_gincr:N \g__postnotes_sectid_int
457   \int_gincr:N \g__postnotes_note_id_int
458   \seq_gput_right:Nx \g__postnotes_queue_seq { \l__postnotes_note_id_tl }
459   \tl_gclear:N \g__postnotes_section_name_tl
460   \keys_set:nn { postnotes/section } {#1}
461   \__postnotes_store_section:nn { \l__postnotes_note_id_tl } {#2}
462   \group_end:
463 }

```

(End definition for __postnotes_section:nn.)

Options for \postnotessection. 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.

```

464 \tl_new:N \g__postnotes_section_name_tl

```



```

465 \keys_define:nn { postnotes/section }
466 {
467     name .tl_gset:N = \g__postnotes_section_name_tl ,
468     name .value_required:n = true ,
469 }

```

7 \printpostnotes

`\printpostnotes` Provide `\printpostnotes`.

`\printpostnotes`

```

470 \NewDocumentCommand \printpostnotes { }
471 { \__postnotes_print_notes: }

```

(End definition for \printpostnotes.)

<code>\pnthechapter</code>	User facing variables, aimed at making available some of the notes' and sections' metadata
<code>\pnthesection</code>	for the user at specific contexts.
<code>\pnthechapternextnote</code>	472 \tl_new:N \pnthechapter
<code>\pnthesectionnextnote</code>	473 \tl_new:N \pnthesection
<code>\pnthepage</code>	474 \tl_new:N \pnthechapternextnote
	475 \tl_new:N \pnthesectionnextnote
	476 \tl_new:N \pnthepage

(End definition for \pnthechapter and others.)

<code>\g__postnotes_print_postnotes_int</code>	Auxiliary variables for <code>__postnotes_print_notes:</code> .
<code>\l__postnotes_print_note_id_tl</code>	477 \int_new:N \g__postnotes_print_postnotes_int
<code>\l__postnotes_print_note_id_next_tl</code>	478 \tl_new:N \l__postnotes_print_note_id_tl
<code>\l__postnotes_print_counter_tl</code>	479 \tl_new:N \l__postnotes_print_note_id_next_tl
<code>\l__postnotes_print_mark_tl</code>	480 \tl_new:N \l__postnotes_print_counter_tl
<code>\l__postnotes_print_type_curr_tl</code>	481 \tl_new:N \l__postnotes_print_mark_tl
<code>\l__postnotes_print_type_next_tl</code>	482 \tl_new:N \l__postnotes_print_type_curr_tl
<code>\l__postnotes_print_type_prev_tl</code>	483 \tl_new:N \l__postnotes_print_type_next_tl
<code>\l__postnotes_print_content_tl</code>	484 \tl_new:N \l__postnotes_print_type_prev_tl
<code>\l__postnotes_clear_queue_seq</code>	485 \tl_new:N \l__postnotes_print_content_tl
	486 \seq_new:N \l__postnotes_clear_queue_seq

(End definition for \g__postnotes_print_postnotes_int and others.)

`__postnotes_print_notes:` hooks. Both meant at providing points of entry for additional setup, specially to add support to packages and features which require it. The `postnotes/print/begin` hook is run early in `__postnotes_print_notes:` and only once per call, after the user options have been processed. The `postnotes/print/eachnote` 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.

```

487 \NewHook { postnotes/print/begin }
488 \NewHook { postnotes/print/eachnote }

```

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. Its 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, 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.

```

489 \newcounter { postnotetext }
490 \newcounter { postnotesection }
491 \setcounter { postnotesection } { 10000 }

```

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

```

\__postnotes_print_notes:

492 \cs_new_protected:Npn \__postnotes_print_notes:
493 {
494   \group_begin:
495   \int_gincr:N \g__postnotes_print_postnotes_int
496   \seq_if_empty:NTF \g__postnotes_queue_seq
497     { \msg_warning:nn { postnotes } { empty-printpostnotes } }
498     {
499       \pnheading
500       \UseHook { postnotes/print/begin }
501       \tl_set:Nn \l__postnotes_print_type_prev_tl { open }
502       \seq_set_eq:NN \l__postnotes_clear_queue_seq \g__postnotes_queue_seq
503       \__postnotes_verify_multipass:N \g__postnotes_queue_seq
504       \bool_if:NT \l__postnotes_sort_bool
505         { \__postnotes_sort_queue:N \g__postnotes_queue_seq }
506       \bool_gset_true:N \g__postnotes_header_vars_next_bool
507       \__postnotes_get_headers_data:N \g__postnotes_queue_seq
508       \__postnotes_set_headers_vars_first:

```

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

```

509     \bool_if:NF \l__postnotes_print_as_list_bool
510     {
511       \cs_set_eq:NN \@afterindentfalse \@afterindenttrue

```

```

512         \@afterindenttrue
513     }
514     \bool_until_do:nn { \seq_if_empty_p:N \g__postnotes_queue_seq }
515     {
516         \seq_gpop_left:NN \g__postnotes_queue_seq
517         \l__postnotes_print_note_id_tl
518         \__postnotes_prop_get:nnN { \l__postnotes_print_note_id_tl }
519         { type } \l__postnotes_print_type_curr_tl
520         \tl_if_eq:NnTF \l__postnotes_print_type_curr_tl { section }
521         { % type_curr = 'section'
522             \seq_if_empty:NNTF \g__postnotes_queue_seq
523             {
524                 \tl_set:Nn \l__postnotes_print_note_id_next_tl { noid }
525                 \tl_set:Nn \l__postnotes_print_type_next_tl { close }
526             }
527             {
528                 \seq_get_left:NN \g__postnotes_queue_seq
529                 \l__postnotes_print_note_id_next_tl
530                 \__postnotes_prop_get:nnN
531                 { \l__postnotes_print_note_id_next_tl }
532                 { type } \l__postnotes_print_type_next_tl
533             }
534         }

```

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

```

534         \tl_if_eq:NnTF \l__postnotes_print_type_next_tl { note }
535         {
536             \stepcounter { postnotessection }
537             \group_begin:
538             \__postnotes_prop_get:nnN
539             { \l__postnotes_print_note_id_tl }
540             { thechapter } \pnthechapter
541             \__postnotes_prop_get:nnN
542             { \l__postnotes_print_note_id_tl }
543             { thesection } \pnthesection
544             \__postnotes_prop_get:nnN
545             { \l__postnotes_print_note_id_next_tl }
546             { thechapter } \pnthechapternextnote
547             \__postnotes_prop_get:nnN
548             { \l__postnotes_print_note_id_next_tl }
549             { thesection } \pnthesectionnextnote
550             \__postnotes_prop_get:nnN
551             { \l__postnotes_print_note_id_tl }
552             { content } \l__postnotes_print_content_tl
553             \l__postnotes_print_content_tl
554             \group_end:

```

Set type_prev for the next iteration.

```

555         \tl_set:NV \l__postnotes_print_type_prev_tl
556         \l__postnotes_print_type_curr_tl
557     }
558 }
559 { % type_curr = 'note'
560     \tl_if_eq:NnTF \l__postnotes_print_type_prev_tl { note }
561     {
562         \bool_if:NNTF \l__postnotes_print_as_list_bool

```

```

563         { \exp_args:Nx \begin { \l__postnotes_print_env_tl } }
564         { \group_begin: }
565         \l__postnotes_print_format_tl
566     }
567     \group_begin:
568     \UseHook { postnotes/print/eachnote }
569     \__postnotes_get_pageref:Nx \pnthepage
570     { mark@ \l__postnotes_print_note_id_tl }
571     \__postnotes_prop_get:nnN
572     { \l__postnotes_print_note_id_tl }
573     { mark } \l__postnotes_print_mark_tl
574     \__postnotes_prop_get:nnN
575     { \l__postnotes_print_note_id_tl }
576     { counter } \l__postnotes_print_counter_tl
577     \__postnotes_prop_get:nnN
578     { \l__postnotes_print_note_id_tl }
579     { content } \l__postnotes_print_content_tl
580     \cs_set:Npn \@currentcounter { postnotetext }
581     \int_set:Nn \c@postnotetext
582     { \int_eval:n { \l__postnotes_print_counter_tl } }
583     \cs_set:Npx \@currentlabel
584     { \p@postnote \l__postnotes_print_mark_tl }
585     \__postnotes_hyperref_make_currentHref:n
586     { postnote. \l__postnotes_print_note_id_tl .text }
587     \__postnotes_text_mark_wrapper:n
588     {
589         \__postnotes_set_text_page_label:x
590         { \l__postnotes_print_note_id_tl }
591         \__postnotes_typeset_text_mark:eV
592         { \l__postnotes_print_note_id_tl }
593         \l__postnotes_print_mark_tl
594     }
595     \l__postnotes_print_content_tl
596     \l__postnotes_post_printnote_tl
597     \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.

```

598     \seq_if_empty:NTF \g__postnotes_queue_seq
599     {
600         \tl_set:Nn \l__postnotes_print_note_id_next_tl { noid }
601         \tl_set:Nn \l__postnotes_print_type_next_tl { close }
602     }
603     {
604         \seq_get_left:NN \g__postnotes_queue_seq
605         \l__postnotes_print_note_id_next_tl
606         \__postnotes_prop_get:nnN
607         { \l__postnotes_print_note_id_next_tl }
608         { type } \l__postnotes_print_type_next_tl
609     }
610     \tl_if_eq:NnF \l__postnotes_print_type_next_tl { note }
611     {
612         \bool_if:NTF \l__postnotes_print_as_list_bool

```

```

613             { \exp_args:Nx \end { \l__postnotes_print_env_tl } }
614             { \group_end: }
615         }

```

Set `type_prev` for the next iteration.

```

616             \tl_set:NV \l__postnotes_print_type_prev_tl
617             \l__postnotes_print_type_curr_tl
618         }
619     }
620     \AddToHookNext { shipout/after }
621     { \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.

```

622         \seq_map_inline:Nn \l__postnotes_clear_queue_seq
623         { \__postnotes_prop_gclear:n { ##1 } }
624     }
625     \group_end:
626 }

```

(End definition for `__postnotes_print_notes:.`)

```

627 \msg_new:nnn { postnotes } { empty-printpostnotes }
628 { 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>}
629 \cs_new_protected:Npn \__postnotes_typeset_text_mark:nn #1#2
630 {
631     \bool_if:NTF \l__postnotes_hyperlink_bool
632     {
633         \__postnotes_hyperref_set_anchor:n { postnote. #1 .text }
634         \bool_if:NTF \l__postnotes_backlink_bool
635         {
636             \__postnotes_make_text_mark:nnn {#2}
637             { \hyper@linkstart { link } { postnote. #1 .mark } }
638             { \hyper@linkend }
639         }
640         { \__postnotes_make_text_mark:nnn {#2} { } { } }
641     }
642     { \__postnotes_make_text_mark:nnn {#2} { } { } }
643 }
644 \cs_generate_variant:Nn \__postnotes_typeset_text_mark:nn { eV }
645 \cs_new_protected:Npn \__postnotes_text_mark_wrapper:n #1
646 {
647     \bool_if:NTF \l__postnotes_print_as_list_bool
648     {
649         \item
650         [ \l__postnotes_pre_textmark_tl #1 \l__postnotes_post_textmark_tl ]
651     }
652     { \l__postnotes_pre_textmark_tl #1 \l__postnotes_post_textmark_tl }
653 }

```

(End 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`, (at least) 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>
654 \cs_new_protected:Npn \_postnotes_verify_multipass:N #1
655 {
656   \group_begin:
657   \seq_clear:N \l_tmpa_seq
658   \seq_map_inline:Nn #1
659   {
660     \_postnotes_prop_get:nnN {##1} { multibool } \l_tmpa_tl
661     \tl_if_eq:NnTF \l_tmpa_tl { true }
662     {
663       \cs_if_exist:cT
664       { \c_postnotes_ref_prefix_tl @ mark@ ##1 }
665       { \seq_put_right:Nn \l_tmpa_seq {##1} }
666     }
667     { \seq_put_right:Nn \l_tmpa_seq {##1} }
668   }
669   \seq_gset_eq:NN #1 \l_tmpa_seq
670   \group_end:
671 }
```

(End definition for `_postnotes_verify_multipass:N`.)

`_postnotes_sort_queue:N` Sorting function for `_postnotes_print_notes:.`

```
\_postnotes_sort_queue:N <\g_postnotes_queue_seq>
672 \cs_new_protected:Npn \_postnotes_sort_queue:N #1
673 {
674   \group_begin:
675   \seq_gsort:Nn #1
```

```

676 {
677   \_postnotes_prop_get:nnN {##1} { pnsectid } \l_tmpa_tl
678   \_postnotes_prop_get:nnN {##2} { pnsectid } \l_tmpb_tl
679   \tl_if_eq:NNTF \l_tmpa_tl \l_tmpb_tl
680   {
681     \_postnotes_prop_get:nnN {##1} { type } \l_tmpa_tl
682     \_postnotes_prop_get:nnN {##2} { type } \l_tmpb_tl
683     \bool_lazy_and:nnTF
684     { \str_if_eq_p:Vn \l_tmpa_tl { note } }
685     { \str_if_eq_p:Vn \l_tmpb_tl { note } }
686     {
687       \_postnotes_prop_get:nnN {##1} { sortnum } \l_tmpa_tl
688       \_postnotes_prop_get:nnN {##2} { sortnum } \l_tmpb_tl
689       \fp_compare:nNnTF { \l_tmpa_tl } > { \l_tmpb_tl }
690       { \sort_return_swapped: }
691       { \sort_return_same: }
692     }
693     { \sort_return_same: }
694   }
695   { \sort_return_same: }
696 }
697 \group_end:
698 }

```

(End 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>	699 <code>\tl_new:N \pnhdpagefirst</code>
<code>\pnhdsectlast</code>	700 <code>\tl_new:N \pnhdpagelast</code>
<code>\pnhdnamefirst</code>	701 <code>\tl_new:N \pnhdchapfirst</code>
<code>\pnhdnamelast</code>	702 <code>\tl_new:N \pnhdchaplast</code>
	703 <code>\tl_new:N \pnhdsectfirst</code>
	704 <code>\tl_new:N \pnhdsectlast</code>
	705 <code>\tl_new:N \pnhdnamefirst</code>
	706 <code>\tl_new:N \pnhdnamelast</code>

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

Auxiliary variables for the headers infrastructure.

<code>\g__postnotes_header_page_first_prop</code>	707 <code>\prop_new:N \g__postnotes_header_page_first_prop</code>
<code>\g__postnotes_header_page_last_prop</code>	708 <code>\prop_new:N \g__postnotes_header_page_last_prop</code>
<code>\g__postnotes_header_chap_first_prop</code>	709 <code>\prop_new:N \g__postnotes_header_chap_first_prop</code>
<code>\g__postnotes_header_chap_last_prop</code>	710 <code>\prop_new:N \g__postnotes_header_chap_last_prop</code>
<code>\g__postnotes_header_sect_first_prop</code>	711 <code>\prop_new:N \g__postnotes_header_sect_first_prop</code>
<code>\g__postnotes_header_sect_last_prop</code>	712 <code>\prop_new:N \g__postnotes_header_sect_last_prop</code>
<code>\g__postnotes_header_name_first_prop</code>	713 <code>\prop_new:N \g__postnotes_header_name_first_prop</code>
<code>\g__postnotes_header_name_last_prop</code>	714 <code>\prop_new:N \g__postnotes_header_name_last_prop</code>
<code>\g__postnotes_header_prev_last_page_tl</code>	715 <code>\tl_new:N \g__postnotes_header_prev_last_page_tl</code>
<code>\g__postnotes_header_prev_last_chap_tl</code>	716 <code>\tl_new:N \g__postnotes_header_prev_last_chap_tl</code>
<code>\g__postnotes_header_prev_last_sect_tl</code>	717 <code>\tl_new:N \g__postnotes_header_prev_last_sect_tl</code>
<code>\g__postnotes_header_prev_last_name_tl</code>	718 <code>\tl_new:N \g__postnotes_header_prev_last_name_tl</code>
<code>\l__postnotes_prev_text_page_tl</code>	719 <code>\tl_new:N \l__postnotes_prev_text_page_tl</code>
<code>\l__postnotes_curr_text_page_tl</code>	720 <code>\tl_new:N \l__postnotes_curr_text_page_tl</code>
<code>\l__postnotes_prev_mark_page_tl</code>	721 <code>\tl_new:N \l__postnotes_prev_mark_page_tl</code>
<code>\l__postnotes_prev_mark_chap_tl</code>	722 <code>\tl_new:N \l__postnotes_prev_mark_chap_tl</code>
<code>\l__postnotes_prev_mark_sect_tl</code>	723 <code>\tl_new:N \l__postnotes_prev_mark_sect_tl</code>
<code>\l__postnotes_prev_mark_name_tl</code>	724 <code>\tl_new:N \l__postnotes_prev_mark_name_tl</code>

(End 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>`


```

725 \cs_new_protected:Npn \__postnotes_get_headers_data:N #1
726 {
727   \group_begin:
728   \tl_gclear:N \pnhdpagefirst
729   \tl_gclear:N \pnhdpagelast
730   \tl_gclear:N \pnhdchapfirst
731   \tl_gclear:N \pnhdchaplast
732   \tl_gclear:N \pnhdsectfirst
733   \tl_gclear:N \pnhdsectlast
734   \tl_gclear:N \pnhdnamefirst
735   \tl_gclear:N \pnhdnamelast
736   \prop_gclear:N \g__postnotes_header_page_first_prop
737   \prop_gclear:N \g__postnotes_header_page_last_prop
738   \prop_gclear:N \g__postnotes_header_chap_first_prop
739   \prop_gclear:N \g__postnotes_header_chap_last_prop
740   \prop_gclear:N \g__postnotes_header_sect_first_prop
741   \prop_gclear:N \g__postnotes_header_sect_last_prop
742   \prop_gclear:N \g__postnotes_header_name_first_prop
743   \prop_gclear:N \g__postnotes_header_name_last_prop
744   \tl_gclear:N \g__postnotes_header_prev_last_page_tl
745   \tl_gclear:N \g__postnotes_header_prev_last_chap_tl
746   \tl_gclear:N \g__postnotes_header_prev_last_sect_tl
747   \tl_gclear:N \g__postnotes_header_prev_last_name_tl
748   \tl_clear:N \l__postnotes_prev_text_page_tl
749   \tl_clear:N \l__postnotes_curr_text_page_tl
750   \tl_clear:N \l__postnotes_prev_mark_page_tl
751   \tl_clear:N \l__postnotes_prev_mark_chap_tl
752   \tl_clear:N \l__postnotes_prev_mark_sect_tl
753   \tl_clear:N \l__postnotes_prev_mark_name_tl
754   \seq_map_inline:Nn #1
755   {
756     \exp_args:Nx \tl_if_eq:nnT
757     { \__postnotes_prop_item:nn {##1} { type } }
758     { note }
759     {
760       \__postnotes_get_pageref:Nn
761       \l__postnotes_curr_text_page_tl { text@ ##1 }
762       \tl_if_empty:NF \l__postnotes_curr_text_page_tl
763       {
764         \tl_if_eq:NNTF
765         \l__postnotes_prev_text_page_tl
766         \l__postnotes_curr_text_page_tl
767         {

```

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

```

768       \__postnotes_get_pageref:Nn
769       \l__postnotes_prev_mark_page_tl { mark@ ##1 }
770       \__postnotes_prop_get:nnN {##1} { thechapter }
771       \l__postnotes_prev_mark_chap_tl
772       \__postnotes_prop_get:nnN {##1} { thesection }
773       \l__postnotes_prev_mark_sect_tl
774       \__postnotes_prop_get:nnN {##1} { pnsectname }
775       \l__postnotes_prev_mark_name_tl
776     }

```

777

{

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).

```

778         \tl_if_empty:NF \l__postnotes_prev_text_page_tl
779         {
780             \prop_gput:Nxx \g__postnotes_header_page_last_prop
781             { \l__postnotes_prev_text_page_tl }
782             { \l__postnotes_prev_mark_page_tl }
783             \prop_gput:Nxx \g__postnotes_header_chap_last_prop
784             { \l__postnotes_prev_text_page_tl }
785             { \l__postnotes_prev_mark_chap_tl }
786             \prop_gput:Nxx \g__postnotes_header_sect_last_prop
787             { \l__postnotes_prev_text_page_tl }
788             { \l__postnotes_prev_mark_sect_tl }
789             \prop_gput:Nxx \g__postnotes_header_name_last_prop
790             { \l__postnotes_prev_text_page_tl }
791             { \l__postnotes_prev_mark_name_tl }
792         }

```

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

```

793         \prop_gput:Nxx \g__postnotes_header_page_first_prop
794         { \l__postnotes_curr_text_page_tl }
795         { \__postnotes_extract_pageref:n { mark@ ##1 } }
796         \prop_gput:Nxx \g__postnotes_header_chap_first_prop
797         { \l__postnotes_curr_text_page_tl }
798         { \__postnotes_prop_item:nn {##1} { thechapter } }
799         \prop_gput:Nxx \g__postnotes_header_sect_first_prop
800         { \l__postnotes_curr_text_page_tl }
801         { \__postnotes_prop_item:nn {##1} { thesection } }
802         \prop_gput:Nxx \g__postnotes_header_name_first_prop
803         { \l__postnotes_curr_text_page_tl }
804         { \__postnotes_prop_item:nn {##1} { pnsectname } }

```

Store prev_mark data for the first note on the page.

```

805         \__postnotes_get_pageref:Nn
806         \l__postnotes_prev_mark_page_tl { mark@ ##1 }
807         \__postnotes_prop_get:nnN {##1} { thechapter }
808         \l__postnotes_prev_mark_chap_tl
809         \__postnotes_prop_get:nnN {##1} { thesection }
810         \l__postnotes_prev_mark_sect_tl
811         \__postnotes_prop_get:nnN {##1} { pnsectname }
812         \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).

```

813         \tl_set:NV \l__postnotes_prev_text_page_tl
814         \l__postnotes_curr_text_page_tl
815     }
816 }

```

```

817     }
818 }

```

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.

```

819 \tl_if_empty:NF \l__postnotes_prev_text_page_tl
820 {
821   \prop_gput:Nxx \g__postnotes_header_page_last_prop
822   { \l__postnotes_prev_text_page_tl }
823   { \l__postnotes_prev_mark_page_tl }
824   \prop_gput:Nxx \g__postnotes_header_chap_last_prop
825   { \l__postnotes_prev_text_page_tl }
826   { \l__postnotes_prev_mark_chap_tl }
827   \prop_gput:Nxx \g__postnotes_header_sect_last_prop
828   { \l__postnotes_prev_text_page_tl }
829   { \l__postnotes_prev_mark_sect_tl }
830   \prop_gput:Nxx \g__postnotes_header_name_last_prop
831   { \l__postnotes_prev_text_page_tl }
832   { \l__postnotes_prev_mark_name_tl }
833 }
834 \group_end:
835 }

```

(End 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}}

836 \cs_new_protected:Npn \__postnotes_set_headers_vars:n #1
837 {
838   \group_begin:
839   \prop_get:NnNTF \g__postnotes_header_page_first_prop
840   {#1} \l_tmpa_tl
841   { \tl_gset:NV \pnhdpagefirst \l_tmpa_tl }
842   { \tl_gset:NV \pnhdpagefirst \g__postnotes_header_prev_last_page_tl }
843   \prop_get:NnNTF \g__postnotes_header_page_last_prop
844   {#1} \l_tmpa_tl
845   {
846     \tl_gset:NV \pnhdpagelast \l_tmpa_tl
847     \tl_gset:NV \g__postnotes_header_prev_last_page_tl \l_tmpa_tl
848   }
849   { \tl_gset:NV \pnhdpagelast \g__postnotes_header_prev_last_page_tl }

```

```

850 \prop_get:NnNTF \g__postnotes_header_chap_first_prop
851   {#1} \l_tmpa_tl
852   { \tl_gset:NV \pnhdchapfirst \l_tmpa_tl }
853   { \tl_gset:NV \pnhdchapfirst \g__postnotes_header_prev_last_chap_tl }
854 \prop_get:NnNTF \g__postnotes_header_chap_last_prop
855   {#1} \l_tmpa_tl
856   {
857     \tl_gset:NV \pnhdchaplast \l_tmpa_tl
858     \tl_gset:NV \g__postnotes_header_prev_last_chap_tl \l_tmpa_tl
859   }
860   { \tl_gset:NV \pnhdchaplast \g__postnotes_header_prev_last_chap_tl }
861 \prop_get:NnNTF \g__postnotes_header_sect_first_prop
862   {#1} \l_tmpa_tl
863   { \tl_gset:NV \pnhdsectfirst \l_tmpa_tl }
864   { \tl_gset:NV \pnhdsectfirst \g__postnotes_header_prev_last_sect_tl }
865 \prop_get:NnNTF \g__postnotes_header_sect_last_prop
866   {#1} \l_tmpa_tl
867   {
868     \tl_gset:NV \pnhdsectlast \l_tmpa_tl
869     \tl_gset:NV \g__postnotes_header_prev_last_sect_tl \l_tmpa_tl
870   }
871   { \tl_gset:NV \pnhdsectlast \g__postnotes_header_prev_last_sect_tl }
872 \prop_get:NnNTF \g__postnotes_header_name_first_prop
873   {#1} \l_tmpa_tl
874   { \tl_gset:NV \pnhdnamefirst \l_tmpa_tl }
875   { \tl_gset:NV \pnhdnamefirst \g__postnotes_header_prev_last_name_tl }
876 \prop_get:NnNTF \g__postnotes_header_name_last_prop
877   {#1} \l_tmpa_tl
878   {
879     \tl_gset:NV \pnhdnamelast \l_tmpa_tl
880     \tl_gset:NV \g__postnotes_header_prev_last_name_tl \l_tmpa_tl
881   }
882   { \tl_gset:NV \pnhdnamelast \g__postnotes_header_prev_last_name_tl }
883 \group_end:
884 }
885 \cs_generate_variant:Nn \__postnotes_set_headers_vars:n { x }

```

(End 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.

```

886 \AddToHook { shipout/before } [ postnotes/header ]
887   { \__postnotes_set_headers_vars_next: }
888 \bool_new:N \g__postnotes_header_vars_next_bool
889 \cs_new_protected:Npn \__postnotes_set_headers_vars_next:
890   {
891     \bool_if:NT \g__postnotes_header_vars_next_bool
892       { \__postnotes_set_headers_vars:x { \int_eval:n { \c@page + 1 } } }
893   }

```

```

894 \cs_new_protected:Npn \__postnotes_set_headers_vars_first:
895 {
896   \__postnotes_set_print_page_label:x
897   { \int_use:N \g__postnotes_print_postnotes_int }
898   \__postnotes_set_headers_vars:x
899   {
900     \__postnotes_extract_pageref:e
901     { print@ \int_use:N \g__postnotes_print_postnotes_int }
902   }
903 }

```

(End 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

904 \NewDocumentCommand \pnheaderdefault {}
905 {
906   \tl_if_eq:NNTF \pnhdpagefirst \pnhdpagelast
907   { \pnhdnotes{} ~ \pnhdtopage{} ~ \pnhdpagefirst }
908   { \pnhdnotes{} ~ \pnhdtopages{} ~ \pnhdpagefirst -- \pnhdpagelast }
909 }

```

(End definition for `\pnheaderdefault`.)

9 Compatibility

`\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) call `\postnote[nomark]{\label{mynote}}` right before the caption, then use `\postnoteref{mynote}` inside the caption.

```

910 \AddToHook { postnotes/note/begin } [ postnotes ]
911 {
912   \cs_if_exist:NT \@capttype
913   { \bool_set_true:N \l__postnotes_maybe_multi_bool }
914 }

```

hyperref

```

915 \bool_new:N \g__postnotes_hyperref_loaded_bool
916 \AddToHook { package/hyperref/after }
917 { \bool_gset_true:N \g__postnotes_hyperref_loaded_bool }

```

```

\__postnotes_hyperref_make_currentHref:n
\__postnotes_hyperref_set_anchor:n
\__postnotes_ref_star:n

```

Auxiliary functions for hyperref support.

```

    \_postnotes_hyperref_make_currentHref:n {\langle anchor/destination \rangle}
    \_postnotes_hyperref_set_anchor:n {\langle anchor/destination \rangle}
    \_postnotes_ref_star:n {\langle label \rangle}

918 \cs_new_protected:Npn \_postnotes_hyperref_make_currentHref:n #1
919 {
920     \bool_if:NT \g__postnotes_hyperref_loaded_bool
921     { \Hy@MakeCurrentHref {#1} }
922 }
923 \cs_new_protected:Npn \_postnotes_hyperref_set_anchor:n #1
924 {
925     \bool_if:NT \g__postnotes_hyperref_loaded_bool
926     { \Hy@raisedlink { \hyper@anchor {#1} } }
927 }
928 \cs_new_protected:Npn \_postnotes_ref_star:n #1
929 {
930     \bool_if:NTF \g__postnotes_hyperref_loaded_bool
931     { \ref*{#1} }
932     { \ref{#1} }
933 }

```

(End definition for `_postnotes_hyperref_make_currentHref:n`, `_postnotes_hyperref_set_anchor:n`, and `_postnotes_ref_star:n`.)

biblatex

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

We can make biblatex’s `refsegments` and `refcontexts` work, but `refsections` are more complicated. Currently, `refsections` are only supported if `\printpostnotes` is called within each `refsection`, one cannot “accumulate” the notes from all `refsections` and print them at the end. Well, one can, but they will be considered part of the current `refsection` of wherever `\printpostnotes` is placed (unless they were also cited in the original `refsection` out of a `\postnote` which, of course, is not something to rely on).

Note that support for these features of biblatex is *experimental*.

```

934 \AddToHook { package/biblatex/after }
935 {

```

Store biblatex variables for each note.

```

936     \AddToHook { postnotes/store/note } [ postnotes ]
937     {
938         \prop_gput:cnx { \_postnotes_data_name:e { \l__postnotes_note_id_tl } }
939         { biblatex@refsection } { \int_use:N \c@refsection }
940         \prop_gput:cnx { \_postnotes_data_name:e { \l__postnotes_note_id_tl } }
941         { biblatex@refsegment } { \int_use:N \c@refsegment }
942         \prop_gput:cnx { \_postnotes_data_name:e { \l__postnotes_note_id_tl } }
943         { biblatex@refcontextbool }
944         { \iftoggle { blx@refcontext } { true } { false } }
945         \prop_gput:cnV { \_postnotes_data_name:e { \l__postnotes_note_id_tl } }
946         { biblatex@refcontext } \blx@refcontext@context
947     }

```

biblatex setup, once for `\printpostnotes` call.

```

948 \AddToHook { postnotes/print/begin } [ postnotes ]
949 {
950   \__postnotes_biblatex_endrefcontext_local:
951   \__postnotes_biblatex_citereset_local:

```

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

```

952   \toggletrue { blx@footnote }
953 }

```

Restore biblatex variables for each note.

```

954 \tl_new:N \l__postnotes_biblatex_restore_tl
955 \AddToHook { postnotes/print/eachnote } [ postnotes ]
956 {
957   \__postnotes_prop_get:nnN { \l__postnotes_print_note_id_tl }
958   { biblatex@refsection } \l__postnotes_biblatex_restore_tl
959   \int_set:Nn \c@refsection { \l__postnotes_biblatex_restore_tl }
960   \__postnotes_prop_get:nnN { \l__postnotes_print_note_id_tl }
961   { biblatex@refsegment } \l__postnotes_biblatex_restore_tl
962   \int_set:Nn \c@refsegment { \l__postnotes_biblatex_restore_tl }
963   \__postnotes_prop_get:nnN { \l__postnotes_print_note_id_tl }
964   { biblatex@refcontextbool } \l__postnotes_biblatex_restore_tl
965   \use:c { toggle \l__postnotes_biblatex_restore_tl } { blx@refcontext }
966   \__postnotes_prop_get:nnN { \l__postnotes_print_note_id_tl }
967   { biblatex@refcontext } \l__postnotes_biblatex_restore_tl
968   \blx@edef@refcontext { \l__postnotes_biblatex_restore_tl }
969 }

```

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.

```

970 \AddToHook { begindocument/before } [ postnotes/mkbibendnote ]
971 {
972   \cs_set_nopar:Npn \blx@theendnote { \postnote }
973   \cs_set_nopar:Npn \blx@theendnotetext
974   { \blx@err@endnote \footnotetext }
975 }

```

Auxiliary functions.

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

```

976 \cs_new_protected:Npn \__postnotes_biblatex_endrefcontext_local:
977 {
978   \togglefalse { blx@refcontext }
979   \tl_clear:N \blx@refcontext@labelprefix
980   \tl_clear:N \blx@refcontext@labelprefix@real
981   \tl_set:Nx \blx@refcontext@sortingtemplatename { \blx@sorting }
982   \tl_set:Nn \blx@refcontext@sortingnamekeytemplatename { global }
983   \tl_set:Nn \blx@refcontext@uniquenametemplatename { global }
984   \tl_set:Nn \blx@refcontext@labelalphanametemplatename { global }
985   \blx@edef@refcontext
986   {

```

```

987         \blx@refcontext@sortingtemplatename /
988         \blx@refcontext@sortingnamekeytemplatename /
989         /
990         \blx@refcontext@uniquenametemplatename /
991         \blx@refcontext@labelalphanametemplatename
992     }
993 }

```

(End 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.

```

994     \cs_new_protected:Npn \_postnotes_biblatex_citereset_local:
995     {
\global\cslet{blx@bsee@the\c@refsection}\empty
\global\cslet{blx@fsee@the\c@refsection}\empty
996         \tl_clear:c { blx@bsee@ \int_use:N \c@refsection }
997         \tl_clear:c { blx@fsee@ \int_use:N \c@refsection }
\blx@ibidreset@force
998         \undef \blx@lastkey@text
999         \undef \blx@lastkey@foot
\blx@idemreset@force
1000         \undef \blx@lasthash@text
1001         \undef \blx@lasthash@foot
\blx@opcitreset@force
1002         \clist_map_inline:Nn \blx@trackhash@text
1003         { \csundef { blx@lastkey@text@ ##1 } }
1004         \tl_clear:N \blx@trackhash@text
1005         \clist_map_inline:Nn \blx@trackhash@foot
1006         { \csundef { blx@lastkey@foot@ ##1 } }
1007         \tl_clear:N \blx@trackhash@foot
\blx@loccitreset@force
1008         \clist_map_inline:Nn \blx@trackkeys@text
1009         { \csundef { blx@lastnote@text@ ##1 } }
1010         \tl_clear:N \blx@trackkeys@text
1011         \clist_map_inline:Nn \blx@trackkeys@foot
1012         { \csundef { blx@lastnote@foot@ ##1 } }
1013         \tl_clear:N \blx@trackkeys@foot
and all of them do:
1014         \cs_set_eq:NN \blx@lastmpfn \z@
1015     }

```

(End definition for _postnotes_biblatex_citereset_local:.)

```

1016 }

```


`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. Given the complexity of this machinery, it would be madness not to use `biblatex`'s infrastructure directly, and try to “emulate” it. 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 (excluded from the package) 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`.

```

1017  $\langle *gobble \rangle$ 
1018 \AddToHook { package/biblatex/after }
1019 {
1020   \tl_new:N \l__postnotes_biblatex_orig_refsection_tl
1021   \tl_new:N \g__postnotes_biblatex_prev_refsection_tl
1022   \AddToHook { postnotes/print/begin } [ postnotes ]
1023   {
1024     \tl_set:Nx \l__postnotes_biblatex_orig_refsection_tl
1025       { \int_use:N \c@refsection }
1026     \tl_gset:Nx \g__postnotes_biblatex_prev_refsection_tl
1027       \l__postnotes_biblatex_orig_refsection_tl
1028   }
1029   \AddToHook { postnotes/print/eachnote } [ postnotes ]
1030   {
1031     \__postnotes_prop_get:nnN { \l__postnotes_print_note_id_tl }
1032       { biblatex@refsection } \l__postnotes_biblatex_restore_tl
1033     \tl_if_eq:NNF
1034       \l__postnotes_biblatex_restore_tl
1035       \g__postnotes_biblatex_prev_refsection_tl
1036     {
1037       \int_set:Nn \c@blx@maxsection
1038         { \l__postnotes_biblatex_restore_tl - 1 }
1039       \tl_gset_eq:NN \g__postnotes_biblatex_prev_refsection_tl
1040         \l__postnotes_biblatex_restore_tl
1041       \group_begin:
1042       \cs_set_eq:NN \label \use_none:n
1043       \cs_set_eq:NN \blx@info \use_none:n
1044       \blx@endrefsection
1045       \refsection
1046       \group_end:
1047     }
1048   }
1049 }
1050  $\langle /gobble \rangle$ 

```

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:`.

```
1051 \tl_new:N \l__postnotes_note_zlabel_tl
```

(End definition for \l__postnotes_note_zlabel_tl.)

```
1052 \AddToHook { package/zref-user/after }
1053 {
```

Provide `zlabel` option.

```
1054   \keys_define:nn { postnotes/note }
1055   {
1056     zlabel .tl_set:N = \l__postnotes_note_zlabel_tl ,
1057     zlabel .value_required:n = true ,
1058   }
```

`\postnotezref` Provide `\postnotezref`.

```
\postnotezref(*){<label>}
```

```
1059 \NewDocumentCommand \postnotezref { s m }
1060 { \__postnotes_note_zref:nn {#1} {#2} }
```

(End definition for \postnotezref.)

`__postnotes_note_zref:nn` The internal version of `\postnotezref`.

```
\__postnotes_note_zref:nn {<star bool>} {<label>}
```

```
1061 \cs_new_protected:Npn \__postnotes_note_zref:nn #1#2
1062 {
1063   \group_begin:
1064   \__postnotes_typeset_mark_wrapper:n
1065   {
1066     \bool_lazy_and:nnTF
1067     { ! #1 }
1068     { \l__postnotes_hyperlink_bool }
1069     {
1070       \hyperlink
1071       { \zref@extractdefault {#2} { anchor } { } }
1072       { \__postnotes_make_mark:nnn { \zref{#2} } { } { } }
1073     }
1074     { \__postnotes_make_mark:nnn { \zref{#2} } { } { } }
1075   }
1076   \group_end:
1077 }
```

(End definition for __postnotes_note_zref:nn.)

```
1078 }
```

zref-clever

```
1079 \AddToHook { package/zref-clever/after }
1080 {
1081   \zcsetup
1082   {
1083     countertype = { postnote = endnote } ,
1084     countertype = { postnotetext = endnote } ,
1085   }
1086   \AddToHook { postnotes/print/begin } [ postnotes ]
1087   { \zcsetup { counterresetby = { postnotetext = postnotessection } } }
1088 }
```

amsmath

```
1089 \AddToHook { package/amsmath/after }
1090 {
```

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

```
1091   \AddToHook { postnotes/note/inhibit } [ postnotes ]
1092   {
1093     \legacy_if:nT { measuring@ }
1094     {
1095       \bool_set_true:N \l__postnotes_inhibit_note_bool
1096       \bool_set_true:N \l__postnotes_print_plain_mark_bool
1097     }
1098   }
```

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.

```
1099   \AddToHook { postnotes/note/begin } [ postnotes ]
1100   {
1101     \legacy_if:nF { firstchoice@ }
1102     { \bool_set_true:N \l__postnotes_maybe_multi_bool }
1103   }
1104 }
```

csquotes

```
1105 \AddToHook { package/csquotes/after }
1106 {
1107   \bool_new:N \l__postnotes_csquotes_measuring_bool
1108   \BlockquoteDisable
1109   { \bool_set_true:N \l__postnotes_csquotes_measuring_bool }
1110   \AddToHook { postnotes/note/inhibit } [ postnotes ]
1111   {
1112     \bool_if:NT \l__postnotes_csquotes_measuring_bool
1113     {
1114       \bool_set_true:N \l__postnotes_inhibit_note_bool
```

```

1115         \bool_set_true:N \l__postnotes_print_plain_mark_bool
1116     }
1117 }
1118 }

```

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>.

```

1119 \AddToHook { package/tabularx/after }
1120 {
1121     \bool_new:N \l__postnotes_tabularx_inside_env_bool
1122     \AddToHook { env/tabularx/begin } [ postnotes ]
1123     {
1124         \bool_set_true:N \l__postnotes_tabularx_inside_env_bool
1125         \cs_set_eq:NN \__postnotes_tabularx_saved_write:Nn \write
1126     }
1127     \AddToHook { postnotes/note/inhibit } [ postnotes ]
1128     {
1129         \bool_lazy_and:nnT
1130         { \l__postnotes_tabularx_inside_env_bool }
1131         { ! \cs_if_eq_p:NN \write \__postnotes_tabularx_saved_write:Nn }
1132         {
1133             \bool_set_true:N \l__postnotes_inhibit_note_bool
1134             \bool_set_true:N \l__postnotes_print_plain_mark_bool
1135         }
1136     }
1137 }

```

tabularray

I’ve tried, but I could not find any “handle” to distinguish in `tabularray` a trial/measure pass from the final one. So we use `__postnotes_verify_multipass:N` for it.

```

1138 \AddToHook { package/tabularray/after }
1139 {
1140     \clist_map_inline:nn
1141     { tblr , longtblr , talltblr , booktabs , longtabs , talltabs , +array }
1142     {
1143         \AddToHook { env/#1/begin } [ postnotes ]
1144         { \bool_set_true:N \l__postnotes_maybe_multi_bool }
1145     }
1146 }

```

10 Languages

<code>\pntitle</code> <code>\pnhdnotes</code> <code>\pnhdtopage</code> <code>\pnhdtopages</code>	Set of language specific user variables. They are used in the default value of the <code>heading</code> option and in <code>\pnheaderdefault</code> which, ultimately, is also used in the same place.
1147 1148 1149	<code>\tl_new:N \pntitle</code> <code>\tl_new:N \pnhdnotes</code> <code>\tl_new:N \pnhdtopage</code>

```

1150 \tl_new:N \pnhdtopages
1151 \tl_set:Nn \pntitle { Notes }
1152 \tl_set:Nn \pnhdnotes { Notes }
1153 \tl_set:Nn \pnhdtopage { to~page }
1154 \tl_set:Nn \pnhdtopages { to~pages }

```

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

`__postnotes_define_language:nn` Defines language specific values for $\langle postnote language \rangle$ by storing a set of assignments for the language specific variables in $\langle setup \rangle$. $\langle postnote language \rangle$ 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 {\langle postnote language \rangle} {\langle setup \rangle}

1155 \cs_new_protected:Npn \__postnotes_define_language:nn #1#2
1156 {
1157   \tl_new:c { g__postnotes_language_ #1 _t1 }
1158   \tl_gset:cn { g__postnotes_language_ #1 _t1 } {#2}
1159 }

```

(End 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/\langle language \rangle/beforeextras` so that users can override it with a traditional `\addto\extras\langle language \rangle`.

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 – `\langle language \rangle\langle caption \rangle`, which is meant to be set with `\setlocalecaption`, and not suitable for our needs. The `\extras\langle language \rangle` 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 $\langle babel language \rangle$ to execute the setup defined by `__postnotes_define_language:nn` for $\langle postnote language \rangle$ at the `babel/\langle language \rangle/beforeextras` hook.

```

\__postnotes_set_babel_language:nn {\langle babel language \rangle} {\langle postnote language \rangle}

1160 \cs_new_protected:Npn \__postnotes_set_babel_language:nn #1#2
1161 {
1162   \ActivateGenericHook { babel/#1/beforeextras }
1163   \exp_args:Nnv \AddToHook { babel/#1/beforeextras }
1164   { g__postnotes_language_ #2 _t1 }
1165 }

```

(End 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)}

1166 \cs_new_protected:Npn \_postnotes_set_polyglossia_language:nn #1#2
1167 {
1168   \AddToHook { package/polyglossia/after }
1169   {
1170     \exp_args:Nnv \csgappto { captions #1 }
1171     { g__postnotes_language_ #2 _tl }
1172   }
1173 }
```

(End definition for `_postnotes_set_polyglossia_language:nn`.)

English

```

1174 \_postnotes_define_language:nn { english }
1175 {
1176   \tl_set:Nn \pntitle      { Notes }
1177   \tl_set:Nn \pnhdnotes    { Notes }
1178   \tl_set:Nn \pnhdtopage   { to~page }
1179   \tl_set:Nn \pnhdtopages  { to~pages }
1180 }
1181 \_postnotes_set_babel_language:nn { english }    { english }
1182 \_postnotes_set_babel_language:nn { british }    { english }
1183 \_postnotes_set_babel_language:nn { american }    { english }
1184 \_postnotes_set_babel_language:nn { canadian }    { english }
1185 \_postnotes_set_babel_language:nn { australian } { english }
1186 \_postnotes_set_babel_language:nn { newzealand } { english }
1187 \_postnotes_set_babel_language:nn { UKenglish }   { english }
1188 \_postnotes_set_babel_language:nn { USenglish }   { english }
1189 \_postnotes_set_polyglossia_language:nn { english } { english }
```

Portuguese

```

1190 \_postnotes_define_language:nn { portuguese }
1191 {
1192   \tl_set:Nn \pntitle      { Notas }
1193   \tl_set:Nn \pnhdnotes    { Notas }
1194   \tl_set:Nn \pnhdtopage   { da~página }
```

```

1195 \tl_set:Nn \pnhdtopages { das-páginas }
1196 }
1197 \__postnotes_set_babel_language:nn { portuguese } { portuguese }
1198 \__postnotes_set_babel_language:nn { brazilian } { portuguese }
1199 \__postnotes_set_babel_language:nn { portuges } { portuguese }
1200 \__postnotes_set_babel_language:nn { brazil } { portuguese }
1201 \__postnotes_set_polyglossia_language:nn { portuguese } { portuguese }
1202 </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	1162
\addto	37
\addtocounter	35
\AddToHook	239, 886, 910, 916, 934, 936, 948, 955, 970, 1018, 1022, 1029, 1052, 1079, 1086, 1089, 1091, 1099, 1105, 1110, 1119, 1122, 1127, 1138, 1143, 1163, 1168
\AddToHookNext	620
B	
\begin	563
\BlockquoteDisable	1108
bool commands:	
\bool_gset_false:N	621
\bool_gset_true:N	506, 917
\bool_if:NTF	30, 244, 334, 336, 379, 392, 400, 504, 509, 562, 612, 631, 634, 647, 891, 920, 925, 930, 1112
\bool_lazy_and:nnTF	439, 683, 1066, 1129
\bool_new:N	135, 212, 213, 214, 266, 351, 354, 355, 371, 372, 888, 915, 1107, 1121
\bool_set_false:N	142, 221, 230, 231, 246, 376, 377
\bool_set_true:N	147, 220, 225, 226, 365, 913, 1095, 1096, 1102, 1109, 1114, 1115, 1124, 1133, 1134, 1144
\bool_to_str:N	46
\bool_until_do:nn	514
box commands:	
\box_use:N	291
\box_wd:N	290
\l_tmpa_box	289, 290, 291
C	
\caption	10, 11, 29
\chapter	116
\citereset	32
clist commands:	
\clist_map_inline:Nn	1002, 1005, 1008, 1011
\clist_map_inline:nn	1140
\counterwithin	12
cs commands:	
\cs_generate_variant:Nn	18, 80, 86, 92, 99, 106, 410, 644, 885
\cs_if_eq_p:NN	1131
\cs_if_exist:NTF	34, 54, 95, 102, 112, 663, 912
\cs_new:Npn	16, 68, 100, 188
\cs_new_protected:Npn	20, 50, 63, 70, 75, 81, 87, 93, 114, 121, 313, 396, 412, 425, 434, 453, 492, 629, 645, 654, 672, 725, 836, 889, 894, 918, 923, 928, 976, 994, 1061, 1155, 1160, 1166
\cs_new_protected:Npx	73
\cs_set:Npn	328, 580
\cs_set:Npx	329, 583
\cs_set_eq:NN	161, 178, 511, 1014, 1042, 1043, 1125
\cs_set_nopar:Npn	972, 973
\csgappto	1170
\csundef	1003, 1006, 1009, 1012
E	
\end	613
\endgroup	20
\endinput	12
\endlist	170, 187
\endnotemark	10
\endnotetext	10
\endrefcontext	31
\enoteheading	18

`\pnhdnamelast` 699, 735, 879, 882
`\pnhdnotes` 907, 908, 1147, 1177, 1193
`\pnhdpagefirst`
 699, 728, 841, 842, 906, 907, 908
`\pnhdpagelast` . 699, 729, 846, 849, 906, 908
`\pnhdsectfirst` 699, 732, 863, 864
`\pnhdsectlast` 699, 733, 868, 871
`\pnhdtopage` 907, 1147, 1178, 1194
`\pnhdtopages` 908, 1147, 1179, 1195
`\pnheaderdefault` ... 29, 36, 117, 124, 904
`\pnheading` 5, 109, 112, 499
`\pnthechapter` 472, 540
`\pnthechapternextnote` 472, 546
`\pnthepage` 472, 569
`\pnthesection` 472, 543
`\pnthesectionnextnote` 472, 549
`\pntitle` 116, 123, 1147, 1176, 1192
`\postnote` 2, 9,
 11–13, 15, 22, 23, 29–31, 35, 310, 972
`\postnotemark` 11
`\postnoteref` 11, 15, 35, 432
postnotes internal commands:
 `\l_postnotes_backlink_bool`
 214, 235, 634
 `_postnotes_biblatex_citereset_-`
 local: 951, 994
 `_postnotes_biblatex_endrefcontext_-`
 local: 950, 976
 `\l_postnotes_biblatex_orig_-`
 refsection_tl . 33, 1020, 1024, 1027
 `\g_postnotes_biblatex_prev_-`
 refsection_tl 1021, 1026, 1035, 1039
 `\l_postnotes_biblatex_restore_-`
 tl . 954, 958, 959, 961, 962, 964,
 965, 967, 968, 1032, 1034, 1038, 1040
 `\l_postnotes_clear_queue_seq` ...
 21, 477, 502, 622
 `\l_postnotes_csquotes_measuring_-`
 bool 1107, 1109, 1112
 `\l_postnotes_curr_text_page_tl` .
 26, 707, 749,
 761, 762, 766, 794, 797, 800, 803, 814
 `_postnotes_data_name:n`
 2, 12, 16, 22, 23, 24,
 26, 28, 36, 39, 41, 43, 45, 47, 52, 53,
 56, 59, 61, 65, 69, 71, 938, 940, 942, 945
 `_postnotes_define_language:nn` .
 37, 38, 1155, 1174, 1190
 `_postnotes_extract_pageref:n` ..
 4, 93, 795, 900
 `_postnotes_get_headers_data:N` .
 23, 24, 27, 507, 725
 `_postnotes_get_pageref:Nn`
 4, 93, 569, 760, 768, 805
 `\g_postnotes_header_chap_first_-`
 prop 707, 738, 796, 850
 `\g_postnotes_header_chap_last_-`
 prop 707, 739, 783, 824, 854
 `\g_postnotes_header_name_first_-`
 prop 707, 742, 802, 872
 `\g_postnotes_header_name_last_-`
 prop 707, 743, 789, 830, 876
 `\g_postnotes_header_page_first_-`
 prop 707, 736, 793, 839
 `\g_postnotes_header_page_last_-`
 prop 707, 737, 780, 821, 843
 `\g_postnotes_header_prev_last_-`
 chap_tl 707, 745, 853, 858, 860
 `\g_postnotes_header_prev_last_-`
 name_tl 707, 747, 875, 880, 882
 `\g_postnotes_header_prev_last_-`
 page_tl 707, 744, 842, 847, 849
 `\g_postnotes_header_prev_last_-`
 sect_tl 707, 746, 864, 869, 871
 `\g_postnotes_header_sect_first_-`
 prop 707, 740, 799, 861
 `\g_postnotes_header_sect_last_-`
 prop 707, 741, 786, 827, 865
 `\g_postnotes_header_vars_next_-`
 bool 28, 506, 621, 888, 891
 `\l_postnotes_hyperlink_bool` ...
 212, 220,
 225, 230, 246, 336, 400, 441, 631, 1068
 `\g_postnotes_hyperref_loaded_-`
 bool 915, 917, 920, 925, 930
 `_postnotes_hyperref_make_-`
 currentHref:n ... 30, 330, 585, 918
 `_postnotes_hyperref_set_-`
 anchor:n 30, 338, 402, 633, 918
 `\l_postnotes_hyperref_warn_bool`
 213, 221, 226, 231, 244
 `_postnotes_inhibit_note`: 374
 `_postnotes_inhibit_note:TF` ...
 22, 317, 371
 `\l_postnotes_inhibit_note_bool` .
 .. 14, 371, 376, 392, 1095, 1114, 1133
 `_postnotes_list_makelabel:n` ...
 161, 178, 188
 `_postnotes_make_mark:nnn` . 192,
 390, 403, 407, 444, 446, 1072, 1074
 `_postnotes_make_text_mark:nnn` .
 196, 636, 640, 642
 `\l_postnotes_manual_sortnum_-`
 bool 30, 354, 365
 `\l_postnotes_mark_tl` .. 25, 320,
 323, 329, 344, 350, 358, 381, 387, 390
 `\l_postnotes_maybe_multi_bool` ..
 22, 35, 46, 355, 913, 1102, 1144

\l__postnotes_nomark_bool	\l__postnotes_print_plain_mark_-
. 334, 351, 360	bool
__postnotes_note:nn 14, 372, 377, 379, 1096, 1115, 1134
. 12, 14, 15, 311, 312	\g__postnotes_print_postnotes_-
\g__postnotes_note_id_int	int 477, 495, 897, 901
. 12, 306, 319, 457	\l__postnotes_print_type_curr_tl
\l__postnotes_note_id_tl 477, 519, 520, 556, 617
. 12, 306, 326, 331, 332, 339,	\l__postnotes_print_type_next_tl
344, 346, 458, 461, 938, 940, 942, 945 477, 525, 532, 534, 601, 608, 610
\l__postnotes_note_label_tl	\l__postnotes_print_type_prev_tl
. 353, 368, 427, 428 477, 501, 555, 560, 616
__postnotes_note_ref:nn 15, 433, 434	__postnotes_prop_gclear:n 3, 63, 623
\l__postnotes_note_zlabel_tl	__postnotes_prop_get:nnN . 3, 63,
. 34, 429, 430, 1051, 1056	518, 530, 538, 541, 544, 547, 550,
__postnotes_note_zref:nn	571, 574, 577, 606, 660, 677, 678,
. 34, 1060, 1061	681, 682, 687, 688, 770, 772, 774,
\l__postnotes_post_printnote_tl	807, 809, 811, 957, 960, 963, 966, 1031
. 143, 202, 209, 596	__postnotes_prop_item:nn
\l__postnotes_post_textmark_tl 3, 63, 757, 798, 801, 804
. 201, 207, 650, 652	\g__postnotes_queue_seq 12,
\l__postnotes_pre_textmark_tl	22, 24, 306, 325, 458, 496, 502, 503,
. 200, 205, 650, 652	505, 507, 514, 516, 522, 528, 598, 604
\l__postnotes_prev_mark_chap_tl	\c__postnotes_ref_prefix_tl
. 707, 751, 771, 785, 808, 826 4, 72, 74, 95, 96, 102, 103, 664
\l__postnotes_prev_mark_name_tl	__postnotes_ref_star:n . 30, 446, 918
. 707, 753, 775, 791, 812, 832	\l__postnotes_saved_spacefactor_-
\l__postnotes_prev_mark_page_tl	tl 411, 417, 422
. 707, 750, 769, 782, 806, 823	\g__postnotes_sectid_int 44, 452, 456
\l__postnotes_prev_mark_sect_tl	__postnotes_section:nn . 16, 451, 452
. 707, 752, 773, 788, 810, 829	\g__postnotes_section_name_tl
\l__postnotes_prev_text_page_tl 42, 459, 464, 467
. 26, 707, 748, 765, 778, 781, 784,	__postnotes_set_babel_language:nn
787, 790, 813, 819, 822, 825, 828, 831 37, 1160,
\l__postnotes_print_as_list_bool	1181, 1182, 1183, 1184, 1185, 1186,
. . . . 135, 142, 147, 509, 562, 612, 647	1187, 1188, 1197, 1198, 1199, 1200
\l__postnotes_print_content_tl	__postnotes_set_headers_vars:n
. 477, 552, 553, 579, 595 24, 27, 28, 836, 892, 898
\l__postnotes_print_counter_tl	__postnotes_set_headers_vars_-
. 477, 576, 582	first: 24, 28, 508, 886
\l__postnotes_print_env_tl	__postnotes_set_headers_vars_-
. 134, 144, 148, 563, 613	next: 24, 28, 886
\l__postnotes_print_format_tl	__postnotes_set_mark_page_-
. 127, 130, 565	label:n 4, 23, 75, 332
\l__postnotes_print_mark_tl	__postnotes_set_polyglossia_-
. 477, 573, 584, 593	language:nn . . . 38, 1166, 1189, 1201
\l__postnotes_print_note_id_-	__postnotes_set_print_page_-
next_tl 477,	label:n 4, 23, 75, 896
524, 529, 531, 545, 548, 600, 605, 607	__postnotes_set_text_page_-
\l__postnotes_print_note_id_tl	label:n 4, 23, 75, 589
. 477, 517, 518,	__postnotes_set_user_labels:
539, 542, 551, 570, 572, 575, 578, 34, 333, 425
586, 590, 592, 957, 960, 963, 966, 1031	\l__postnotes_sort_bool 266, 269, 504
__postnotes_print_notes:	\l__postnotes_sort_num_fp 31, 352, 364
. 17, 18, 21-23, 471, 492	__postnotes_sort_queue:N 22, 505, 672

