

The zref-check package implementation*

Gustavo Barros[†]

2021-08-17

Contents

1	Initial setup	2
2	Dependencies	2
3	zref setup	2
4	Plumbing	3
4.1	Messages	3
4.2	Integer testing	4
4.3	Options	5
4.4	Position on page	9
4.5	Counter	11
4.6	Label formats	12
4.7	Property values	12
5	User interface	14
5.1	\zcheck	14
5.2	Targets	16
6	Checks	16
6.1	Setup	17
6.2	Running	19
6.3	Conditionals	21
6.3.1	This page	21
6.3.2	On page	22
6.3.3	Before / After	22
6.3.4	Pages	23
6.3.5	Close / Far	25
6.3.6	Chapter	26
6.3.7	Section	27
	Index	30

*This file describes v0.2.0, last revised 2021-08-17.

[†]<https://github.com/gusbrs/zref-check>

1 Initial setup

Start the DocStrip guards.

```
1 <*package>
   Identify the internal prefix (LATEX3 DocStrip convention).
2 <@@=zrefcheck>
   For the chapter and section checks, zref-check uses the new hook system in ltcmd-
   hooks, which was released with the 2021/06/01 LATEX kernel.
3 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
4 \IfFormatAtLeastTF{2021-06-01}
5 {}
6 {%
7   \PackageError{zref-check}{LaTeX kernel too old}
8   {%
9     'zref-check' requires a LaTeX kernel newer than 2021-06-01.%
10    \MessageBreak Loading will abort!%
11   }%
12   \endinput
13 }%
   Identify the package.
14 \ProvidesExplPackage {zref-check} {2021-08-17} {0.2.0}
15 {Flexible cross-references with contextual checks based on zref}
```

2 Dependencies

```
16 \RequirePackage { zref-user }
17 \RequirePackage { zref-abspage }
18 \RequirePackage { ifdraft }
```

3 zref setup

`\g__zrefcheck_abschap_int` Provide absolute counters for section and chapter, and respective zref properties, so that we can make checks about relation of chapters/sections regardless of internal counters, since we don't get those for the unnumbered (starred) ones. About the proper place to make the hooks for this purpose, see <https://tex.stackexchange.com/q/605533/105447> (thanks Ulrike Fischer).

```
19 \int_new:N \g__zrefcheck_abschap_int
20 \int_new:N \g__zrefcheck_abssec_int
```

(End definition for `\g__zrefcheck_abschap_int` and `\g__zrefcheck_abssec_int`.)

If the documentclass does not define `\chapter` the only thing that happens is that the chapter counter is never incremented, and the section one never reset.

```
21 \AddToHook { cmd / chapter / before }
22 {
23   \int_gincr:N \g__zrefcheck_abschap_int
24   \int_zero:N \g__zrefcheck_abssec_int
25 }
26 \zref@newprop { zc@abschap } [0] { \int_use:N \g__zrefcheck_abschap_int }
27 \zref@addprop \ZREF@mainlist { zc@abschap }
```

```

28 \AddToHook { cmd / section / before }
29   { \int_gincr:N \g__zrefcheck_abssec_int }
30 \zref@newprop { zc@abssec } [0] { \int_use:N \g__zrefcheck_abssec_int }
31 \zref@addprop \ZREF@mainlist { zc@abssec }

```

This is the list of properties to be used by zref-check, that is, the list of properties the references and targets store. This is the minimum set required, more properties may be added according to options.

```

32 \zref@newlist { zrefcheck }
33 \zref@addprops { zrefcheck }
34 {
35   page ,
36   abspage ,
37   zc@abschap ,
38   zc@abssec
39 }

```

4 Plumbing

4.1 Messages

```

\__zrefcheck_message:nnnn
\__zrefcheck_message:nnnx
40 \cs_new:Npn \__zrefcheck_message:nnnn #1#2#3#4
41 {
42   \use:c { msg_ \l__zrefcheck_msglevel_tl :nnnnn }
43   { zref-check } {#1} {#2} {#3} {#4}
44 }
45 \cs_generate_variant:Nn \__zrefcheck_message:nnnn { nnnx }

(End definition for \__zrefcheck_message:nnnn.)

46 \msg_new:nnn { zref-check } { check-failed }
47 {
48   Failed~check~'~#1'~for~label~'~#2' \iow_newline:
49   on~page~#3~on~input~line~\msg_line_number:.
50 }
51 \msg_new:nnn { zref-check } { double-check }
52 {
53   Double-check~'~#1'~for~label~'~#2' \iow_newline:
54   on~page~#3~on~input~line~\msg_line_number:.
55 }

56 \msg_new:nnn { zref-check } { check-missing }
57 { Check~'~#1'~not~defined~on~input~line~\msg_line_number:. }
58 \msg_new:nnn { zref-check } { property-undefined }
59 { Property~'~#1'~not~defined~on~input~line~\msg_line_number:. }
60 \msg_new:nnn { zref-check } { property-not-in-label }
61 { Label~'~#1'~has~no~property~'~#2'~on~input~line~\msg_line_number:. }
62 \msg_new:nnn { zref-check } { property-not-integer }
63 {
64   Property~'~#1'~for~label~'~#2'~not~an~integer \iow_newline:
65   on~input~line~\msg_line_number:.
66 }

```

```

67 \msg_new:nnn { zref-check } { hyperref-preamble-only }
68 {
69   Option~'hyperref'~only~available~in~the~preamble. \iow_newline:
70   Use~the~starred~version~of~'\noexpand\zcheck'~instead.
71 }
72 \msg_new:nnn { zref-check } { missing-hyperref }
73 { Missing~'hyperref'~package. \iow_newline: Setting~'hyperref=false'. }
74 \msg_new:nnn { zref-check } { ignore-document-only }
75 {
76   Option~'ignore'~only~available~in~the~document. \iow_newline:
77   Use~option~'msglevel'~instead.
78 }
79 \msg_new:nnn { zref-check } { option-preamble-only }
80 {
81   Option~'#1'~only~available~in~the~preamble \iow_newline:
82   on~input~line~\msg_line_number:.
83 }
84 \msg_new:nnn { zref-check } { closerange-not-positive-integer }
85 {
86   Option~'closerange'~not~a~positive~integer \iow_newline:
87   on~input~line~\msg_line_number:~Using~default~value.
88 }
89 \msg_new:nnn { zref-check } { labelcmd-undefined }
90 {
91   Control~sequence~named~'#1'~used~in~option~'labelcmd'~is~not~defined.~
92   Using~default~value.
93 }

```

4.2 Integer testing

```

\__zrefcheck_is_integer:n
\__zrefcheck_int_to_roman:w

```

From <https://tex.stackexchange.com/a/244405> (thanks Enrico Gregorio, aka ‘egreg’), also see <https://tex.stackexchange.com/a/19769>. Following the `l3styleguide`, I made a copy of `__int_to_roman:w`, since it is an internal function from the `int` module, but we still get a warning from `l3build doc`, complaining about it. And we’re using `\tl_if_empty:oTF` instead of `\tl_if_blank:oTF` as in egreg’s answer, since `\romannumeral` is defined so that “the expansion is empty if the number is zero or negative”, not “blank”. A couple of comments about this technique: the underlying `\romannumeral` ignores space tokens and explicit signs (+ and -) in the expansion and hence it can only be used to test positive integers; also the technique cannot distinguish whether it received an empty argument or if “the expansion was empty” as a result of receiving number as argument, so this must also be controlled for since, in our use case, this may happen.

```

94 \cs_new_eq:NN \__zrefcheck_int_to_roman:w \__int_to_roman:w
95 \prg_new_conditional:Npnn \__zrefcheck_is_integer:n #1 { p, T , F , TF }
96 {
97   \tl_if_empty:oTF {#1}
98   { \prg_return_false: }
99   {
100     \tl_if_empty:oTF { \__zrefcheck_int_to_roman:w -0#1 }
101     { \prg_return_true: }
102     { \prg_return_false: }
103   }
104 }

```

(End definition for `_zrefcheck_is_integer:n` and `_zrefcheck_int_to_roman:w`.)

`_zrefcheck_is_integer_rgx:n` A possible alternative to `_zrefcheck_is_integer:n` is to use a straightforward regexp match (see <https://tex.stackexchange.com/a/427559>). It does not suffer from the mentioned caveats from the `_int_to_roman:w` technique, however, while `_zrefcheck_is_integer:n` is expandable, `_zrefcheck_is_integer_rgx:n` is not. Also, `_zrefcheck_is_integer_rgx:n` is probably slower.

```

105 \prg_new_protected_conditional:Npnn \_zrefcheck_is_integer_rgx:n #1 { TF }
106 {
107   \regex_match:nnTF { \A\d+\Z } {#1}
108   { \prg_return_true: }
109   { \prg_return_false: }
110 }

```

(End definition for `_zrefcheck_is_integer_rgx:n`.)

4.3 Options

hyperref option

```

\l_zrefcheck_use_hyperref_bool
\l_zrefcheck_warn_hyperref_bool
111 \bool_new:N \l_zrefcheck_use_hyperref_bool
112 \bool_new:N \l_zrefcheck_warn_hyperref_bool
113 \keys_define:nn { zref-check }
114 {
115   hyperref .choice: ,
116   hyperref / auto .code:n =
117   {
118     \bool_set_true:N \l_zrefcheck_use_hyperref_bool
119     \bool_set_false:N \l_zrefcheck_warn_hyperref_bool
120   } ,
121   hyperref / true .code:n =
122   {
123     \bool_set_true:N \l_zrefcheck_use_hyperref_bool
124     \bool_set_true:N \l_zrefcheck_warn_hyperref_bool
125   } ,
126   hyperref / false .code:n =
127   {
128     \bool_set_false:N \l_zrefcheck_use_hyperref_bool
129     \bool_set_false:N \l_zrefcheck_warn_hyperref_bool
130   } ,
131   hyperref .initial:n = auto ,
132   hyperref .default:n = auto
133 }

```

(End definition for `\l_zrefcheck_use_hyperref_bool` and `\l_zrefcheck_warn_hyperref_bool`.)

```

134 \AddToHook { begindocument }
135 {
136   \@ifpackageloaded { hyperref }
137   {
138     \bool_if:NT \l_zrefcheck_use_hyperref_bool
139     {
140       \RequirePackage { zref-hyperref }
141       \zref@addprop { zrefcheck } { anchor }

```

```

142     }
143   }
144   {
145     \bool_if:NT \l__zrefcheck_warn_hyperref_bool
146     { \msg_warning:nn { zref-check } { missing-hyperref } }
147     \bool_set_false:N \l__zrefcheck_use_hyperref_bool
148   }
149   \keys_define:nn { zref-check }
150   {
151     hyperref .code:n =
152     { \msg_warning:nn { zref-check } { hyperref-preamble-only } }
153   }
154 }

```

msglevel option

`\l__zrefcheck_msglevel_tl`

```

155 \tl_new:N \l__zrefcheck_msglevel_tl
156 \keys_define:nn { zref-check }
157 {
158   msglevel .choice: ,
159   msglevel / warn .code:n =
160   { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } } ,
161   msglevel / info .code:n =
162   { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } } ,
163   msglevel / none .code:n =
164   { \tl_set:Nn \l__zrefcheck_msglevel_tl { none } } ,
165   msglevel / obeydraft .code:n =
166   {
167     \ifdraft
168     { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
169     { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
170   } ,
171   msglevel / obeyfinal .code:n =
172   {
173     \ifoptionfinal
174     { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
175     { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
176   } ,
177   msglevel .value_required:n = true ,
178   msglevel .initial:n = warn ,

```

`ignore` is a convenience alias for `msglevel=none`, but only for use in the document body.

```

179   ignore .code:n =
180   { \msg_warning:nn { zref-check } { ignore-document-only } } ,
181   ignore .value_forbidden:n = true
182 }

```

(End definition for `\l__zrefcheck_msglevel_tl`.)

```

183 \AddToHook { begindocument }
184 {
185   \keys_define:nn { zref-check }
186   { ignore .meta:n = { msglevel = none } }
187 }

```

onpage option

\l__zrefcheck_msgonpage_bool

```
188 \bool_new:N \l__zrefcheck_msgonpage_bool
189 \keys_define:nn { zref-check }
190 {
191   onpage .choice: ,
192   onpage / labelseq .code:n =
193     {
194       \bool_set_false:N \l__zrefcheck_msgonpage_bool
195     } ,
196   onpage / msg .code:n =
197     {
198       \bool_set_true:N \l__zrefcheck_msgonpage_bool
199     } ,
200   onpage / obeydraft .code:n =
201     {
202       \ifdraft
203         { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
204         { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
205       } ,
206   onpage / obeyfinal .code:n =
207     {
208       \ifoptionfinal
209         { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
210         { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
211       } ,
212   onpage .value_required:n = true ,
213   onpage .initial:n = labelseq
214 }
```

(End definition for \l__zrefcheck_msgonpage_bool.)

closerange option

\l__zrefcheck_close_range_int

```
215 \int_new:N \l__zrefcheck_close_range_int
216 \keys_define:nn { zref-check }
217 {
218   closerange .code:n =
219     {
220       \__zrefcheck_is_integer_rgx:nTF {#1}
221       { \int_set:Nn \l__zrefcheck_close_range_int { \int_eval:n {#1} } }
222       {
223         \msg_warning:nn { zref-check } { closerange-not-positive-integer }
224         \int_set:Nn \l__zrefcheck_close_range_int { 5 }
225       }
226     } ,
227   closerange .value_required:n = true ,
228   closerange .initial:n = 5
229 }
```

(End definition for \l__zrefcheck_close_range_int.)

labelcmd option

`\l_zrefcheck_target_label_tl` I'd love to receive the macro itself rather than it's name, but this would bring unwarranted complications: <https://tex.stackexchange.com/a/489570>.

```

230 \tl_new:N \l__zrefcheck_target_label_tl
231 \bool_new:N \l__zrefcheck_target_label_bool
232 \keys_define:nn { zref-check }
233 {
234   labelcmd .code:n =
235   {
236     \tl_set:Nn \l__zrefcheck_target_label_tl {#1}
237     \bool_set_true:N \l__zrefcheck_target_label_bool
238   } ,
239   labelcmd .value_required:n = true ,
240 }

```

(End definition for `\l__zrefcheck_target_label_tl`.)

`__zrefcheck_target_label:n` Default definition of the function for user label setting in `\zctarget` and `zcregion`. It may be redefined at `begindocument` according to option `labelcmd`.

```

241 \cs_new:Npn \__zrefcheck_target_label:n #1
242 { \zref@labelbylist {#1} { zrefcheck } }

```

(End definition for `__zrefcheck_target_label:n`.)

```

243 \AddToHook { begindocument }
244 {
245   \bool_if:NT \l__zrefcheck_target_label_bool
246   {
247     \tl_if_blank:VT \l__zrefcheck_target_label_tl
248     { \tl_clear:N \l__zrefcheck_target_label_tl }
249     \cs_if_exist:cTF { \l__zrefcheck_target_label_tl }
250     {
251       \cs_set:Npx \__zrefcheck_target_label:n #1
252       {
253         \exp_not:o
254         { \cs:w \l__zrefcheck_target_label_tl \cs_end: }
255         {#1}
256       }
257     }
258     {
259       \exp_args:NnnV \msg_warning:nnn { zref-check }
260       { labelcmd-undefined } { \l__zrefcheck_target_label_tl }
261     }
262   }
263   \keys_define:nn { zref-check }
264   {
265     labelcmd .code:n =
266     {
267       \msg_warning:nnn { zref-check }
268       { option-preamble-only } { labelcmd }
269     }
270   }
271 }

```


Package options

Process load-time package options (<https://tex.stackexchange.com/a/15840>).

```
272 \RequirePackage { l3keys2e }
273 \ProcessKeysOptions { zref-check }

\zrefchecksetup Provide \zrefchecksetup.
274 \NewDocumentCommand \zrefchecksetup { m }
275 { \keys_set:nn { zref-check } {#1} }
```

(End definition for \zrefchecksetup.)

4.4 Position on page

Method for determining relative position within the page: the sequence in which the labels get shipped out, inferred from the sequence in which the labels occur in the .aux file.

Some relevant info about the sequence of things: <https://tex.stackexchange.com/a/120978> and `texdoc lthooks`, section “Hooks provided by `\begin{document}`”.

One first attempt at this was to use `\zref@newlabel`, which is the macro in which `zref` stores the label information in the aux file. When the .aux file is read at the beginning of the compilation, this macro is expanded for each of the labels. So, by redefining this macro we can feed a variable (a L3 sequence), and then do what it usually does, which is to define each label with the internal macro `\@newl@bel`, when the .aux file is read.

Patching this macro for this is not possible. First, `\zref@newlabel` is one of those “commands that look ahead” mentioned in `ltxcmdhooks` documentation. Indeed, `\@newl@bel` receives 3 arguments, and `\zref@newlabel` just passes the first, the following two will be scanned ahead. Second, the `ltxcmdhooks` hooks are not actually available when the .aux file is read, they come only after `\begin{document}`. Hence, redefinition would be the only alternative. My attempts at this ended up registered at <https://tex.stackexchange.com/a/604744>. But the best result in these lines was:

```
\ZREF@Robust\edef\zref@newlabel#1{
  \noexpand\seq_gput_right:Nn \noexpand\g__zrefcheck_auxfile_lblseq_seq {#1}
  \noexpand\@newl@bel{\ZREF@RefPrefix}{#1}
}
```

However, better than the above is to just read it from the .aux file directly, which relieves us from hacking into any internals. That’s what David Carlisle’s answer at <https://tex.stackexchange.com/a/147705> does. This answer has actually been converted into the package `listbls` by Norbert Melzer, but it is made to work with regular labels, not with `zref`’s. And it also does not really expose the information in a retrievable way (as far as I can tell). So, the below is adapted from Carlisle’s answer’s technique (a poor man’s version of it...).

There is some subtlety here as to whether this approach makes it safe for us to read the labels at this point without `\zref@wrapper@babel`. The common wisdom is that `babel`’s shorthands are only active after `\begin{document}` (e.g., <https://tex.stackexchange.com/a/98897>). Alas, it is more complicated than that. `Babel`’s documentation says (in section 9.5 Shorthands): “To prevent problems with the loading of other packages after `babel` we reset the catcode of the character to the original one at the end of the package and of each language file (except with `KeepShorthandsActive`). It is re-activate[d] again at `\begin{document}`”. We also need to make

sure that the shorthands are active during the processing of the .aux file. Otherwise some citations may give unexpected results in the printout when a shorthand was used in the optional argument of \bibitem for example.” This is done with \if@files\write\@mainaux{...}. In other words, the catcode change is written in the .aux file itself! Indeed, if you inspect the file, you’ll find them there. Besides, there is still the ominous “except with KeepShorthandsActive”.

However, the *method* we’re using here is not quite the same as the usual run of the .aux file, because we’re actively discarding the lines for which the first token is not equal to \zref@newlabel. I have tested the famous sensitive case for this: **babel french** and labels with colons. And things worked as expected. Well, *if* KeepShorthandsActive is enabled *with french* and we load the package *after babel* things do break, but not quite because of the colons in the labels. Even siunitx breaks in the same conditions...

For reference: About what are valid characters for use in labels: <https://tex.stackexchange.com/a/18312>. About some problems with active colons: <https://tex.stackexchange.com/a/89470>. About the difference between L3 strings and token lists, see <https://tex.stackexchange.com/a/446381>, in particular Joseph Wright’s comment: “Strings are for data that will never be typeset, for example file names, identifiers, etc.: if the material may be used in typesetting, it should be a token list.” See also moewe’s (CW) answer in the same lines. Which suggests using L3 strings for the reference labels might be a good catch all approach, and possibly more robust. David Carlisle’s comment about inputenc and how the strings work is a caveat (see https://tex.stackexchange.com/q/446123#comment1516961_446381, thanks David Carlisle). Still... let’s stick to tradition as long as it works, zref already does a great job in this regard anyway.

\g_zrefcheck_auxfile_lblseq_prop

276 \prop_new:N \g_zrefcheck_auxfile_lblseq_prop

(End definition for \g_zrefcheck_auxfile_lblseq_prop.)

277 \tl_set:Nn \g_tmpa_tl { \c_sys_jobname_str .aux }

278 \file_if_exist:nT { \g_tmpa_tl }

279 {

Retrieve the information from the .aux file, and store it in a property list, so that the sequence can be retrieved in key-value fashion.

280 \ior_open:Nn \g_tmpa_ior { \g_tmpa_tl }

281 \group_begin:

282 \int_zero:N \l_tmpa_int

283 \tl_clear:N \l_tmpa_tl

284 \tl_clear:N \l_tmpb_tl

285 \bool_set_false:N \l_tmpa_bool

286 \ior_map_variable:NNn \g_tmpa_ior \l_tmpa_tl

287 {

288 \tl_map_variable:NNn \l_tmpa_tl \l_tmpb_tl

289 {

290 \tl_if_eq:NnTF \l_tmpb_tl { \zref@newlabel }

291 {

Found a \zref@label, signal it.

292 \bool_set_true:N \l_tmpa_bool

293 }

294 {

```

295         \bool_if:NTF \l_tmpa_bool
296         {
297             \bool_set_false:N \l_tmpa_bool
298             \int_incr:N \l_tmpa_int
299             \prop_gput:Nxx \g_zrefcheck_auxfile_lblseq_prop
300             { \l_tmpb_tl } { \int_use:N \l_tmpa_int }
301         }
302     {

```

If there is not a match of the first token with `\zref@newlabel`, break the loop and discard the rest of the line, to ensure no babel calls to `\catcode` in the `.aux` file get expanded. This also breaks the loop and discards the rest of the `\zref@newlabel` lines after we got the label we wanted, since we reset `\l_tmpa_bool` in the T branch.

```

303         \tl_map_break:
304     }
305 }
306 }
307 }
308 \group_end:
309 \ior_close:N \g_tmpa_ior
310 }

```

The alternate method I had considered (more than that...) for this was using `yx` coordinates supplied by `zref`’s `savepos` module. However, this approach brought in a number of complexities, including the need to patch either `\zref@label` or `\ZREF@label`. In addition, the technique was at the bottom fundamentally flawed. Ulrike Fischer was very much right when she said that “structure and position are two different beasts” (<https://github.com/ho-tex/zref/issues/12#issuecomment-880022576>). It is true that the checks based on it behaved decently, in normal circumstances, and except for outrageous label placement by the user, it would return the expected results. We don’t really need exact coordinates to decide “above/below”. Besides, it would do an exact job for the dedicated target macros of this package. It is also true that the “page” for `\pageref` is stored with the value of where the `\label` is placed, wherever that may be. However, I could not conceive a situation where the `yx` criterion would perform clearly better than the `labelseq` one. And, if that’s the case, and considering the complications it brings, this check was a slippery slope. All in all, I’ve decided to drop it.

4.5 Counter

We need a dedicated counter for the labels generated by the checks and targets. The value of the counter is not relevant, we just need it to be able to set proper anchors with `\refstepcounter`. And, since I couldn’t find a `\refstepcounter` equivalent in L3, we use a standard 2e counter here. I’m also using the technique to ensure the counter is never reset that is used by `zref-abspage.sty` and `\zref@require@unique`. Indeed, the requirements are the same, we need numbers ensured to be *unique* in the counter.

```

311 \begingroup
312 \let \@addtoreset \ltx@gobbletwo
313 \newcounter { zrefcheck }
314 \endgroup
315 \setcounter { zrefcheck } { 0 }

```

4.6 Label formats

```

\__zrefcheck_check_lblfmt:n      \__zrefcheck_check_lblfmt:n {\check id int}}
316 \cs_new:Npn \__zrefcheck_check_lblfmt:n #1 { zrefcheck@ \int_use:N #1 }

(End definition for \__zrefcheck_check_lblfmt:n.)

\__zrefcheck_end_lblfmt:n      \__zrefcheck_end_lblfmt:n {\label}}
317 \cs_new:Npn \__zrefcheck_end_lblfmt:n #1 { #1 @zrefcheck }

(End definition for \__zrefcheck_end_lblfmt:n.)

```

4.7 Property values

`\zrefcheck_get_astl:nnn` A convenience function to retrieve property values from labels. Uses `\g__zrefcheck_auxfile_lblseq_prop` for `lblseq`, and calls `\zref@extractdefault` for everything else.

We cannot use the “return value” of `__zrefcheck_get_astl:nnn` or `__zrefcheck_get_asint:nnn` directly, because we need to use the retrieved property values as arguments in the checks, however we use here a number of non-expandable operations. Hence, we receive a local `tl/int` variable as third argument and set that, so that it is available (and expandable) at the place of use. For this reason, we do not group here, because we are passing a local variable around, but it is expected this function will be called within a group.

We’re returning `\c_empty_tl` in case of failure to find the intended property value (explicitly in `\zref@extractdefault`, but that is also what `\tl_clear:N` does).

```

\zrefcheck_get_astl:nnn {\label}} {\prop}} {\tl var}}

318 \cs_new:Npn \zrefcheck_get_astl:nnn #1#2#3
319 {
320   \tl_clear:N #3
321   \tl_if_eq:nnTF {#2} { lblseq }
322   {
323     \prop_get:NnNF \g__zrefcheck_auxfile_lblseq_prop {#1} #3
324     {
325       \msg_warning:nnnn { zref-check }
326       { property-not-in-label } {#1} {#2}
327     }
328   }
329   {

```

There are three things we need to check to ensure the information we are trying to retrieve here exists: the existence of `{\label}}`, the existence of `{\prop}}`, and whether the particular label being queried actually contains the property. If that’s all in place, the value is passed to the checks, and it’s their responsibility to verify the consistency of this value.

The existence of the label is an user facing issue, and a warning for this is placed in `__zrefcheck_zcheck:nnnnn` (and done with `\zref@refused`). We do check here though for definition with `\zref@ifrefundefined` and silently do nothing if it is undefined, to reduce irrelevant warnings in a fresh compilation round. The other two are more “internal” problems, either some problem with the checks, or with the configuration of `zref` for their consumption.

```

330     \zref@ifrefundefined {#1}
331     {}
332     {
333         \zref@ifpropundefined {#2}
334         { \msg_warning:nnnn { zref-check } { property-undefined } {#2} }
335         {
336             \zref@ifrefcontainsprop {#1} {#2}
337             {
338                 \tl_set:Nx #3
339                 { \zref@extractdefault {#1} {#2} { \c_empty_tl } }
340             }
341             {
342                 \msg_warning:nnnn
343                 { zref-check } { property-not-in-label } {#1} {#2}
344             }
345         }
346     }
347 }
348 }

```

(End definition for \zrefcheck_get_astl:nnn.)

\l__zrefcheck_integer_bool \zrefcheck_get_asint:nnn is a very convenient wrapper around the more general \zrefcheck_get_astl:nnn, since almost always we'll be wanting to compare numbers in the checks. However, it is quite hard for it to ensure an integer is *always* returned in the case of errors. And those do occur, even in a well structured document (e.g., in a first round of compilation). To complicate things, the L3 integer predicates are *very* sensitive to receiving any other kind of data, and they *scream*. To handle this \zrefcheck_get_asint:nnn uses \l__zrefcheck_integer_bool to signal if an integer could not be returned. To use this function always set \l__zrefcheck_integer_bool to true first, then call it as much as you need. If any of these calls got is returning anything which is not an integer, \l__zrefcheck_integer_bool will have been set to false, and you should check that this hasn't happened before actually comparing the integers (\bool_lazy_and:nnTF is your friend).

```

349 \bool_new:N \l__zrefcheck_integer_bool

```

(End definition for \l__zrefcheck_integer_bool.)

\l__zrefcheck_propval_tl

```

350 \tl_new:N \l__zrefcheck_propval_tl

```

(End definition for \l__zrefcheck_propval_tl.)

\zrefcheck_get_asint:nnn

```

\zrefcheck_get_asint:nnn {<label>} {<prop>} {<int var>}
351 \cs_new:Npn \zrefcheck_get_asint:nnn #1#2#3
352 {
353     \zrefcheck_get_astl:nnn {#1} {#2} { \l__zrefcheck_propval_tl }
354     \__zrefcheck_is_integer:nTF { \l__zrefcheck_propval_tl }
355     {

```

Make it an integer data type.

```

356     \int_set:Nn #3 { \int_eval:n { \l__zrefcheck_propval_tl } }
357     }
358     {

```

```

359         \bool_set_false:N \l__zrefcheck_integer_bool
360         \zref@ifrefundefined {#1}

```

Keep silent if ref is undefined to reduce irrelevant warnings in a fresh compilation round. Again, this is also not the point to check for undefined references, that's a task for `__zrefcheck_zcheck:nnnn`.

```

361     { }
362     {
363         \msg_warning:nnnn { zref-check }
364         { property-not-integer } {#2} {#1}
365     }
366 }
367 }

```

(End definition for `\zrefcheck_get_asint:nnn`.)

5 User interface

5.1 `\zcheck`

`\zcheck` The `{\text}` argument of `\zcheck` should not be long, since `\hyperlink` cannot receive a long argument. Besides, there is no reason for it to be. Note, also, that hyperlinks crossing page boundaries have some known issues: <https://tex.stackexchange.com/a/182769>, <https://tex.stackexchange.com/a/54607>, <https://tex.stackexchange.com/a/179907>.

```

\zcheck*[\checks/options]{\labels}{\text}

```

```

368 \NewDocumentCommand \zcheck
369 { s O { } } > { \SplitList { , } } m m }
370 { \zref@wrapper@babel \__zrefcheck_zcheck:nnnn {#3} {#1} {#2} {#4} }

```

(End definition for `\zcheck`.)

```

\g__zrefcheck_id_int
\l__zrefcheck_checkbeg_tl
\l__zrefcheck_checkend_tl
\l__zrefcheck_link_label_tl
\l__zrefcheck_link_anchor_tl
\l__zrefcheck_link_star_bool
371 \int_new:N \g__zrefcheck_id_int
372 \tl_new:N \l__zrefcheck_checkbeg_tl
373 \tl_new:N \l__zrefcheck_checkend_tl
374 \tl_new:N \l__zrefcheck_link_label_tl
375 \tl_new:N \l__zrefcheck_link_anchor_tl
376 \bool_new:N \l__zrefcheck_link_star_bool

```

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

`__zrefcheck_zcheck:nnnn` An intermediate internal function, which does the actual heavy lifting, and places `{\labels}` as first argument, so that it can be protected by `\zref@wrapper@babel` in `\zcheck`. This is the same procedure as the one used in the definition of `\zref` in `zref-user.sty` for protection of babel active characters.

```

\__zrefcheck_zcheck:nnnn {\labels} {\(*)} {\checks/options} {\text}

```

```

377 \cs_new:Npn \__zrefcheck_zcheck:nnnn #1#2#3#4
378 {
379     \group_begin:

```

Set checks keys.

```
380 \__zrefcheck_set_checks_keys:
```

Process local options and checks.

```
381 \keys_set:nn { zref-check } {#3}
```

Names of the labels for this zrefcheck call.

```
382 \int_gincr:N \g__zrefcheck_id_int
383 \tl_set:Nx \l__zrefcheck_checkbeg_tl
384 { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
385 \tl_set:Nx \l__zrefcheck_checkend_tl
386 { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
```

Set checkbeg label.

```
387 \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrefcheck }
```

Typeset $\langle\text{text}\rangle$, with hyperlink when appropriate. Even though the first argument can receive a list of labels, there is no meaningful way to set links to multiple targets. Hence, only the first one is considered for hyperlinking.

```
388 \tl_set:Nn \l__zrefcheck_link_label_tl { \tl_head:n {#1} }
389 \bool_set:Nn \l__zrefcheck_link_star_bool {#2}
390 \zref@ifrefundefined { \l__zrefcheck_link_label_tl }
```

If the reference is undefined, just typeset.

```
391 {#4}
392 {
393   \bool_if:nTF
394   {
395     \l__zrefcheck_use_hyperref_bool &&
396     ! \l__zrefcheck_link_star_bool
397   }
398   {
399     \exp_args:Nx \zrefcheck_get_astl:nnn
400     { \l__zrefcheck_link_label_tl }
401     { anchor } { \l__zrefcheck_link_anchor_tl }
402     \hyperlink { \l__zrefcheck_link_anchor_tl } {#4}
403   }
404   {#4}
405 }
```

Set checkend label.

```
406 \zref@labelbylist { \l__zrefcheck_checkend_tl } { zrefcheck }
```

Check if $\langle\text{labels}\rangle$ are defined.

```
407 \tl_map_function:nN {#1} \zref@refused
```

Run the checks.

```
408 \__zrefcheck_run_checks:nnv
409 { \l__zrefcheck_zcheck_checks_seq } {#1} { \l__zrefcheck_checkbeg_tl }
410 \group_end:
411 }
```

(End definition for $\backslash_zrefcheck_zcheck:nnnn$.)

5.2 Targets

```

\zctarget      \zctarget{<label>}{<text>}
412 \NewDocumentCommand \zctarget { m +m }
413 {
Group contents of \zctarget to avoid leaking the effects of \refstepcounter over
\@currentlabel. The same care is not needed for zcregion, since the environment
is already grouped.
414 \group_begin:
415 \refstepcounter { zrefcheck }
416 \zref@wrapper@babel \_zrefcheck_target_label:n {#1}
417 #2
418 \zref@wrapper@babel
419 \zref@labelbylist { \_zrefcheck_end_lblfmt:n {#1} } { zrefcheck }
420 \group_end:
421 }

```

(End definition for \zctarget.)

```

zcregion      \begin{zcregion}{<label>}
...
\end{zcregion}
422 \NewDocumentEnvironment {zcregion} { m }
423 {
424 \refstepcounter { zrefcheck }
425 \zref@wrapper@babel \_zrefcheck_target_label:n {#1}
426 }
427 {
428 \zref@wrapper@babel
429 \zref@labelbylist { \_zrefcheck_end_lblfmt:n {#1} } { zrefcheck }
430 }

```

(End definition for zcregion.)

6 Checks

What is needed define a zref-check check?

First, a conditional function defined with:

```
\prg_new_conditional:Npnn \_zrefcheck_check_<check>:nn #1#2 { F }
```

where $\langle check \rangle$ is the name of the check, the first argument is the $\{\langle label \rangle\}$ and the second the $\{\langle reference \rangle\}$. The existence of the check is verified by the existence of the function with this name-scheme (and signatures). As usual, this function must return either `\prg_return_true:` or `\prg_return_false:`. Of course, you can define other variants if you need them internally, it is just that what the package does expect and verifies is the existence of the `:nnF` variant.

Note that the naming convention of the checks adopts the perspective of the $\langle reference \rangle$. That is, the “before” check should return true if the $\langle label \rangle$ occurs before the “reference”.

The check conditionals are expected to retrieve zref’s label information with `\zrefcheck_get_astl:nnn` or `\zrefcheck_get_asint:nnn`. Also, technically speaking, the $\langle reference \rangle$ argument is also a label, actually a pair of them, as set by `\zcheck`. For

the “labels”, any zref property in zref’s main list is available, the “references” store the properties in the zrefcheck list. Besides those, there is also the lblseq (fake) property (for either “labels” or “references”), stored in `\g__zrefcheck_auxfile_lblseq_prop`.

Second, the required properties of labels and references must be duly registered for zref. This can be done with `\zref@newprop`, `\zref@addprop` and friends, as usual.

Third, the check must be registered as a key which gets setup in `\zcheck` by `__zrefcheck_set_checks_keys:`.

6.1 Setup

`\l__zrefcheck_zcheck_checks_seq`

```
431 \seq_new:N \l__zrefcheck_zcheck_checks_seq
```

(End definition for `\l__zrefcheck_zcheck_checks_seq`.)

`__zrefcheck_set_checks_keys:`

```
432 \cs_new:Npn \__zrefcheck_set_checks_keys:
433 {
434   \keys_define:nn { zref-check }
435   {
436     thispage .code:n =
437       { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { thispage } } ,
438     thispage .value_forbidden:n = true ,
439
440     prevpage .code:n =
441       { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { prevpage } } ,
442     prevpage .value_forbidden:n = true ,
443
444     nextpage .code:n =
445       { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { nextpage } } ,
446     nextpage .value_forbidden:n = true ,
447
448     facing .code:n =
449       { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { facing } } ,
450     facing .value_forbidden:n = true ,
451
452     above .code:n =
453       { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { above } } ,
454     above .value_forbidden:n = true ,
455
456     below .code:n =
457       { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { below } } ,
458     below .value_forbidden:n = true ,
459
460     pagesbefore .code:n =
461       { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { pagesbefore } } ,
462     pagesbefore .value_forbidden:n = true ,
463
464     ppbefore .code:n =
465       { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { ppbefore } } ,
466     ppbefore .value_forbidden:n = true ,
467
468     pagesafter .code:n =
```

```

469     { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { pagesafter } } ,
470     pagesafter .value_forbidden:n = true ,
471
472     ppafter .code:n =
473     { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { ppafter } } ,
474     ppafter .value_forbidden:n = true ,
475
476     before .code:n =
477     { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { before } } ,
478     before .value_forbidden:n = true ,
479
480     after .code:n =
481     { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { after } } ,
482     after .value_forbidden:n = true ,
483
484     thischap .code:n =
485     { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { thischap } } ,
486     thischap .value_forbidden:n = true ,
487
488     prevchap .code:n =
489     { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { prevchap } } ,
490     prevchap .value_forbidden:n = true ,
491
492     nextchap .code:n =
493     { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { nextchap } } ,
494     nextchap .value_forbidden:n = true ,
495
496     chapsbefore .code:n =
497     { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { chapsbefore } } ,
498     chapsbefore .value_forbidden:n = true ,
499
500     chapsafter .code:n =
501     { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { chapsafter } } ,
502     chapsafter .value_forbidden:n = true ,
503
504     thissec .code:n =
505     { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { thissec } } ,
506     thissec .value_forbidden:n = true ,
507
508     prevsec .code:n =
509     { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { prevsec } } ,
510     prevsec .value_forbidden:n = true ,
511
512     nextsec .code:n =
513     { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { nextsec } } ,
514     nextsec .value_forbidden:n = true ,
515
516     secsbefore .code:n =
517     { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { secsbefore } } ,
518     secsbefore .value_forbidden:n = true ,
519
520     secsafter .code:n =
521     { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { secsafter } } ,
522     secsafter .value_forbidden:n = true ,

```

```

523
524         close .code:n =
525             { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { close } } ,
526         close .value_forbidden:n = true ,
527
528         far .code:n =
529             { \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq { far } } ,
530         far .value_forbidden:n = true ,
531     }
532 }

```

(End definition for __zrefcheck_set_checks_keys:.)

6.2 Running

```

\__zrefcheck_run_checks:nnn    \__zrefcheck_run_checks:nnn {<checks>} {<labels>} {<reference>}
\__zrefcheck_run_checks:nnv    <checks> are expected to be received as a sequence variable.

```

```

533 \cs_new:Npn \__zrefcheck_run_checks:nnn #1#2#3
534 {
535     \group_begin:
536     \tl_map_inline:nn {#2}
537     {
538         \seq_map_inline:Nn #1
539         { \__zrefcheck_do_check:nnn {####1} {##1} {#3} }
540     }
541     \group_end:
542 }
543 \cs_generate_variant:Nn \__zrefcheck_run_checks:nnn { nnv }

```

(End definition for __zrefcheck_run_checks:nnn.)

```

\l__zrefcheck_passedcheck_bool
\l__zrefcheck_onpage_bool
\c__zrefcheck_onpage_checks_seq
544 \bool_new:N \l__zrefcheck_passedcheck_bool
545 \bool_new:N \l__zrefcheck_onpage_bool
546 \seq_new:N \c__zrefcheck_onpage_checks_seq
547 \seq_set_from_clist:Nn \c__zrefcheck_onpage_checks_seq
548 { above , below , before , after }

```

(End definition for \l__zrefcheck_passedcheck_bool, \l__zrefcheck_onpage_bool, and \c__zrefcheck_onpage_checks_seq.)

Variant not provided by expl3.

```

549 \cs_generate_variant:Nn \exp_args:Nnno { Nnoo }

```

```

\__zrefcheck_do_check:nnn    \__zrefcheck_do_check:nnn {<check>} {<label beg>} {<reference beg>}
550 \cs_new:Npn \__zrefcheck_do_check:nnn #1#2#3
551 {
552     \group_begin:

```

<label beg> may be defined or not, it is arbitrary user input. Whether this is the case is checked in __zrefcheck_zcheck:nnnnn, and due warning already ensues. And there is no point in checking “relative position” of an undefined label. Hence, in the absence of #2, we do nothing at all here.

```

553     \zref@ifrefundefined {#2}
554     {}

```

```

555     {
556       \tl_if_empty:nF {#1}
557       {
558         \bool_set_true:N \l__zrefcheck_passedcheck_bool
559         \bool_set_false:N \l__zrefcheck_onpage_bool
560         \cs_if_exist:cTF { __zrefcheck_check_ #1 :nnF }
561         {

```

“label beg” vs “reference beg”.

```

562         \use:c { __zrefcheck_check_ #1 :nnF }
563         {#2} {#3}
564         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }

```

“label beg” vs “reference end”.

```

565         \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
566         {#2} { \__zrefcheck_end_lblfmt:n {#3} }
567         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }

```

“label end” *may* have been created by the target commands.

```

568         \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
569         {}
570         {

```

“label end” vs “reference beg”.

```

571         \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
572         { \__zrefcheck_end_lblfmt:n {#2} } {#3}
573         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }

```

“label end” vs “reference end”.

```

574         \exp_args:Nnoo \use:c { __zrefcheck_check_ #1 :nnF }
575         { \__zrefcheck_end_lblfmt:n {#2} }
576         { \__zrefcheck_end_lblfmt:n {#3} }
577         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
578     }

```

Handle option `onpage=msg`. This is only granted for tests which perform “within this page” checks (above, below, before, after) *and* if any of the two by two checks uses a “within this page” comparison. If both conditions are met, signal.

```

579     \seq_if_in:NnT \c__zrefcheck_onpage_checks_seq {#1}
580     {
581       \__zrefcheck_check_thispage:nnT
582       {#2} {#3}
583       { \bool_set_true:N \l__zrefcheck_onpage_bool }
584       \__zrefcheck_check_thispage:nnT
585       {#2} { \__zrefcheck_end_lblfmt:n {#3} }
586       { \bool_set_true:N \l__zrefcheck_onpage_bool }
587       \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
588       {}
589       {
590         \__zrefcheck_check_thispage:nnT
591         { \__zrefcheck_end_lblfmt:n {#2} } {#3}
592         { \bool_set_true:N \l__zrefcheck_onpage_bool }
593         \__zrefcheck_check_thispage:nnT
594         { \__zrefcheck_end_lblfmt:n {#2} }
595         { \__zrefcheck_end_lblfmt:n {#3} }
596         { \bool_set_true:N \l__zrefcheck_onpage_bool }

```

```

597     }
598   }
599   \bool_if:NTF \l__zrefcheck_passedcheck_bool
600   {
601     \bool_if:N
602     {
603       \l__zrefcheck_msgonpage_bool &&
604       \l__zrefcheck_onpage_bool
605     }
606     {
607       \__zrefcheck_message:nnnx { double-check } {#1} {#2}
608       { \zref@extractdefault {#3} {page} {'unknown'} }
609     }
610   }
611   {
612     \__zrefcheck_message:nnnx { check-failed } {#1} {#2}
613     { \zref@extractdefault {#3} {page} {'unknown'} }
614   }
615 }
616 { \msg_warning:nnn { zref-check } { check-missing } {#1} }
617 }
618 }
619 \group_end:
620 }

```

(End definition for __zrefcheck_do_check:nnn.)

6.3 Conditionals

\l__zrefcheck_lbl_int More readable scratch variables for the tests.
\l__zrefcheck_ref_int 621 \int_new:N \l__zrefcheck_lbl_int
\l__zrefcheck_lbl_b_int 622 \int_new:N \l__zrefcheck_ref_int
\l__zrefcheck_ref_b_int 623 \int_new:N \l__zrefcheck_lbl_b_int
624 \int_new:N \l__zrefcheck_ref_b_int

(End definition for \l__zrefcheck_lbl_int and others.)

6.3.1 This page

```

\__zrefcheck_check_thispage:nn
625 \prg_new_conditional:Npnn \__zrefcheck_check_thispage:nn #1#2 { T , F , TF }
626 {
627   \group_begin:
628   \bool_set_true:N \l__zrefcheck_integer_bool
629   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
630   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
631   \bool_lazy_and:nnTF
632   { \l__zrefcheck_integer_bool }
633   {
634     \int_compare_p:nNn
635     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int } &&

```

‘0’ is the default value of `abspage`, but this value should not happen normally for this property, since even the first page, after it gets shipped out, will receive value ‘1’. So, if

we do find ‘0’ here, better signal something is wrong. This comment extends to all page number checks.

```

636         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
637         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
638     }
639     { \group_insert_after:N \prg_return_true: }
640     { \group_insert_after:N \prg_return_false: }
641 \group_end:
642 }

```

(End definition for `__zrefcheck_check_thispage:nn`.)

6.3.2 On page

`__zrefcheck_check_above:nn`
`__zrefcheck_check_below:nn`

```

643 \prg_new_conditional:Npnn \__zrefcheck_check_above:nn #1#2 { F , TF }
644 {
645     \group_begin:
646     \__zrefcheck_check_thispage:nnTF {#1} {#2}
647     {
648         \bool_set_true:N \l__zrefcheck_integer_bool
649         \zrefcheck_get_asint:nnn {#1} { lblseq } { \l__zrefcheck_lbl_int }
650         \zrefcheck_get_asint:nnn {#2} { lblseq } { \l__zrefcheck_ref_int }
651         \bool_lazy_and:nnTF
652         { \l__zrefcheck_integer_bool }
653         {
654             \int_compare_p:nNn
655             { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
656             ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
657             ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
658         }
659         { \group_insert_after:N \prg_return_true: }
660         { \group_insert_after:N \prg_return_false: }
661     }
662     { \group_insert_after:N \prg_return_false: }
663 \group_end:
664 }
665 \prg_new_conditional:Npnn \__zrefcheck_check_below:nn #1#2 { F , TF }
666 {
667     \__zrefcheck_check_thispage:nnTF {#1} {#2}
668     {
669         \__zrefcheck_check_above:nnTF {#1} {#2}
670         { \prg_return_false: }
671         { \prg_return_true: }
672     }
673     { \prg_return_false: }
674 }

```

(End definition for `__zrefcheck_check_above:nn` and `__zrefcheck_check_below:nn`.)

6.3.3 Before / After

`__zrefcheck_check_before:nn`
`__zrefcheck_check_after:nn`

```

675 \prg_new_conditional:Npnn \__zrefcheck_check_before:nn #1#2 { F }

```

```

676 {
677   \_zrefcheck_check_pagesbefore:nnTF {#1} {#2}
678   { \prg_return_true: }
679   {
680     \_zrefcheck_check_above:nnTF {#1} {#2}
681     { \prg_return_true: }
682     { \prg_return_false: }
683   }
684 }
685 \prg_new_conditional:Npnn \_zrefcheck_check_after:nn #1#2 { F }
686 {
687   \_zrefcheck_check_pagesafter:nnTF {#1} {#2}
688   { \prg_return_true: }
689   {
690     \_zrefcheck_check_below:nnTF {#1} {#2}
691     { \prg_return_true: }
692     { \prg_return_false: }
693   }
694 }

```

(End definition for _zrefcheck_check_before:nn and _zrefcheck_check_after:nn.)

6.3.4 Pages

```

\_zrefcheck_check_nextpage:nn
\_zrefcheck_check_prevpage:nn
\_zrefcheck_check_pagesbefore:nn
\_zrefcheck_check_ppbefore:nn
\_zrefcheck_check_pagesafter:nn
\_zrefcheck_check_ppafter:nn
\_zrefcheck_check_facing:nn
695 \prg_new_conditional:Npnn \_zrefcheck_check_nextpage:nn #1#2 { F }
696 {
697   \group_begin:
698   \bool_set_true:N \l__zrefcheck_integer_bool
699   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
700   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
701   \bool_lazy_and:nnTF
702   { \l__zrefcheck_integer_bool }
703   {
704     \int_compare_p:nNn
705     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
706     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
707     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
708   }
709   { \group_insert_after:N \prg_return_true: }
710   { \group_insert_after:N \prg_return_false: }
711   \group_end:
712 }
713 \prg_new_conditional:Npnn \_zrefcheck_check_prevpage:nn #1#2 { F }
714 {
715   \group_begin:
716   \bool_set_true:N \l__zrefcheck_integer_bool
717   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
718   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
719   \bool_lazy_and:nnTF
720   { \l__zrefcheck_integer_bool }
721   {
722     \int_compare_p:nNn
723     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&

```

```

724         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
725         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
726     }
727     { \group_insert_after:N \prg_return_true: }
728     { \group_insert_after:N \prg_return_false: }
729 \group_end:
730 }
731 \prg_new_conditional:Npnn \__zrefcheck_check_pagesbefore:nn #1#2 { F , TF }
732 {
733     \group_begin:
734     \bool_set_true:N \l__zrefcheck_integer_bool
735     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
736     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
737     \bool_lazy_and:nnTF
738     { \l__zrefcheck_integer_bool }
739     {
740         \int_compare_p:nNn
741         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
742         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
743         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
744     }
745     { \group_insert_after:N \prg_return_true: }
746     { \group_insert_after:N \prg_return_false: }
747 \group_end:
748 }
749 \cs_new_eq:NN \__zrefcheck_check_ppbefore:nnF \__zrefcheck_check_pagesbefore:nnF
750 \prg_new_conditional:Npnn \__zrefcheck_check_pagesafter:nn #1#2 { F , TF }
751 {
752     \group_begin:
753     \bool_set_true:N \l__zrefcheck_integer_bool
754     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
755     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
756     \bool_lazy_and:nnTF
757     { \l__zrefcheck_integer_bool }
758     {
759         \int_compare_p:nNn
760         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
761         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
762         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
763     }
764     { \group_insert_after:N \prg_return_true: }
765     { \group_insert_after:N \prg_return_false: }
766 \group_end:
767 }
768 \cs_new_eq:NN \__zrefcheck_check_ppafter:nnF \__zrefcheck_check_pagesafter:nnF
769 \prg_new_conditional:Npnn \__zrefcheck_check_facing:nn #1#2 { F }
770 {
771     \group_begin:
772     \bool_set_true:N \l__zrefcheck_integer_bool
773     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
774     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
775     \bool_lazy_and:nnTF
776     { \l__zrefcheck_integer_bool }
777     {

```


There exists no “facing” page if the document is not twoside.

```

778         \legacy_if_p:n { @twoside } &&
Now we test “facing”.
779     (
780         (
781             \int_if_odd_p:n { \l__zrefcheck_ref_int } &&
782             \int_compare_p:nNn
783                 { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 }
784         ) ||
785         (
786             \int_if_even_p:n { \l__zrefcheck_ref_int } &&
787             \int_compare_p:nNn
788                 { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 }
789         )
790     ) &&
791     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
792     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
793 }
794 { \group_insert_after:N \prg_return_true: }
795 { \group_insert_after:N \prg_return_false: }
796 \group_end:
797 }

```

(End definition for `__zrefcheck_check_nextpage:nn` and others.)

6.3.5 Close / Far

```

\__zrefcheck_check_close:nn
\__zrefcheck_check_far:nn
798 \prg_new_conditional:Npnn \__zrefcheck_check_close:nn #1#2 { F , TF }
799 {
800     \group_begin:
801     \bool_set_true:N \l__zrefcheck_integer_bool
802     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
803     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
804     \bool_lazy_and:nnTF
805         { \l__zrefcheck_integer_bool }
806     {
807         \int_compare_p:nNn
808             { \int_abs:n { \l__zrefcheck_lbl_int - \l__zrefcheck_ref_int } }
809         <
810             { \l__zrefcheck_close_range_int + 1 } &&
811         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
812         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
813     }
814     { \group_insert_after:N \prg_return_true: }
815     { \group_insert_after:N \prg_return_false: }
816 \group_end:
817 }
818 \prg_new_conditional:Npnn \__zrefcheck_check_far:nn #1#2 { F }
819 {
820     \__zrefcheck_check_close:nnTF {#1} {#2}
821     { \prg_return_false: }
822     { \prg_return_true: }
823 }

```

(End definition for `_zrefcheck_check_close:nn` and `_zrefcheck_check_far:nn`.)

6.3.6 Chapter

```

\_zrefcheck\_check\_thischap:nn
\_zrefcheck\_check\_nextchap:nn
\_zrefcheck\_check\_prevchap:nn
\_zrefcheck\_check\_chapsafter:nn
\_zrefcheck\_check\_chapsbefore:nn
824 \prg_new_conditional:Npnn \_zrefcheck\_check\_thischap:nn #1#2 { F }
825 {
826   \group_begin:
827     \bool_set_true:N \l\_zrefcheck\_integer\_bool
828     \zrefcheck\_get\_asint:nnn {#1} { zc@abschap } { \l\_zrefcheck\_lbl\_int }
829     \zrefcheck\_get\_asint:nnn {#2} { zc@abschap } { \l\_zrefcheck\_ref\_int }
830     \bool_lazy_and:nnTF
831       { \l\_zrefcheck\_integer\_bool }
832       {
833         \int_compare_p:nNn
834           { \l\_zrefcheck\_lbl\_int } = { \l\_zrefcheck\_ref\_int } &&
‘0’ is the default value of zc@abschap property, and means here no \chapter has yet been
issued, therefore it cannot be “this chapter”, nor “the next chapter”, nor “the previous
chapter”, it is just “no chapter”. Note, however, that a statement about a “future”
chapter does not require the “current” one to exist. This comment extends to all chapter
checks.
835           ! \int_compare_p:nNn { \l\_zrefcheck\_lbl\_int } = { 0 } &&
836           ! \int_compare_p:nNn { \l\_zrefcheck\_ref\_int } = { 0 }
837       }
838       { \group_insert_after:N \prg_return_true: }
839       { \group_insert_after:N \prg_return_false: }
840   \group_end:
841 }
842 \prg_new_conditional:Npnn \_zrefcheck\_check\_nextchap:nn #1#2 { F }
843 {
844   \group_begin:
845     \bool_set_true:N \l\_zrefcheck\_integer\_bool
846     \zrefcheck\_get\_asint:nnn {#1} { zc@abschap } { \l\_zrefcheck\_lbl\_int }
847     \zrefcheck\_get\_asint:nnn {#2} { zc@abschap } { \l\_zrefcheck\_ref\_int }
848     \bool_lazy_and:nnTF
849       { \l\_zrefcheck\_integer\_bool }
850       {
851         \int_compare_p:nNn
852           { \l\_zrefcheck\_lbl\_int } = { \l\_zrefcheck\_ref\_int + 1 } &&
853           ! \int_compare_p:nNn { \l\_zrefcheck\_lbl\_int } = { 0 }
854       }
855       { \group_insert_after:N \prg_return_true: }
856       { \group_insert_after:N \prg_return_false: }
857   \group_end:
858 }
859 \prg_new_conditional:Npnn \_zrefcheck\_check\_prevchap:nn #1#2 { F }
860 {
861   \group_begin:
862     \bool_set_true:N \l\_zrefcheck\_integer\_bool
863     \zrefcheck\_get\_asint:nnn {#1} { zc@abschap } { \l\_zrefcheck\_lbl\_int }
864     \zrefcheck\_get\_asint:nnn {#2} { zc@abschap } { \l\_zrefcheck\_ref\_int }
865     \bool_lazy_and:nnTF
866       { \l\_zrefcheck\_integer\_bool }

```

```

867     {
868         \int_compare_p:nNn
869         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
870         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
871         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
872     }
873     { \group_insert_after:N \prg_return_true: }
874     { \group_insert_after:N \prg_return_false: }
875 \group_end:
876 }
877 \prg_new_conditional:Npnn \__zrefcheck_check_chapsafter:nn #1#2 { F }
878 {
879     \group_begin:
880     \bool_set_true:N \l__zrefcheck_integer_bool
881     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
882     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
883     \bool_lazy_and:nnTF
884     { \l__zrefcheck_integer_bool }
885     {
886         \int_compare_p:nNn
887         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
888         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
889     }
890     { \group_insert_after:N \prg_return_true: }
891     { \group_insert_after:N \prg_return_false: }
892 \group_end:
893 }
894 \prg_new_conditional:Npnn \__zrefcheck_check_chapsbefore:nn #1#2 { F }
895 {
896     \group_begin:
897     \bool_set_true:N \l__zrefcheck_integer_bool
898     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
899     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
900     \bool_lazy_and:nnTF
901     { \l__zrefcheck_integer_bool }
902     {
903         \int_compare_p:nNn
904         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
905         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
906         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
907     }
908     { \group_insert_after:N \prg_return_true: }
909     { \group_insert_after:N \prg_return_false: }
910 \group_end:
911 }

```

(End definition for __zrefcheck_check_thischap:nn and others.)

6.3.7 Section

```

\__zrefcheck_check_thissec:nn
\__zrefcheck_check_nextsec:nn 912 \prg_new_conditional:Npnn \__zrefcheck_check_thissec:nn #1#2 { F }
\__zrefcheck_check_prevsec:nn 913 {
\__zrefcheck_check_secsafter:nn 914     \group_begin:
\__zrefcheck_check_secsbefore:nn

```

```

915 \bool_set_true:N \l__zrefcheck_integer_bool
916 \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
917 \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
918 \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
919 \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
920 \bool_lazy_and:nnTF
921 { \l__zrefcheck_integer_bool }
922 {
923   \int_compare_p:nNn
924     { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
925   \int_compare_p:nNn
926     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int } &&

```

‘0’ is the default value of `zc@abssec` property, and means here no `\section` has yet been issued since its counter has been reset, which occurs at the beginning of the document and at every chapter. Hence, as is the case for chapters, ‘0’ is just “not a section”. The same observation about the need of the “current” section to exist to be able to refer to a “future” one also holds. This comment extends to all section checks.

```

927   ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
928   ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
929 }
930 { \group_insert_after:N \prg_return_true: }
931 { \group_insert_after:N \prg_return_false: }
932 \group_end:
933 }
934 \prg_new_conditional:Npnn \__zrefcheck_check_nextsec:nn #1#2 { F }
935 {
936   \group_begin:
937   \bool_set_true:N \l__zrefcheck_integer_bool
938   \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
939   \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
940   \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
941   \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
942   \bool_lazy_and:nnTF
943   { \l__zrefcheck_integer_bool }
944   {
945     \int_compare_p:nNn
946       { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
947     \int_compare_p:nNn
948       { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
949     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
950   }
951   { \group_insert_after:N \prg_return_true: }
952   { \group_insert_after:N \prg_return_false: }
953   \group_end:
954 }
955 \prg_new_conditional:Npnn \__zrefcheck_check_prevsec:nn #1#2 { F }
956 {
957   \group_begin:
958   \bool_set_true:N \l__zrefcheck_integer_bool
959   \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
960   \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
961   \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
962   \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }

```

```

963     \bool_lazy_and:nnTF
964     { \l__zrefcheck_integer_bool }
965     {
966         \int_compare_p:nNn
967         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
968         \int_compare_p:nNn
969         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
970         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
971         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
972     }
973     { \group_insert_after:N \prg_return_true: }
974     { \group_insert_after:N \prg_return_false: }
975 \group_end:
976 }
977 \prg_new_conditional:Npnn \__zrefcheck_check_secsafter:nn #1#2 { F }
978 {
979     \group_begin:
980     \bool_set_true:N \l__zrefcheck_integer_bool
981     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
982     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
983     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
984     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
985     \bool_lazy_and:nnTF
986     { \l__zrefcheck_integer_bool }
987     {
988         \int_compare_p:nNn
989         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
990         \int_compare_p:nNn
991         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
992         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
993     }
994     { \group_insert_after:N \prg_return_true: }
995     { \group_insert_after:N \prg_return_false: }
996 \group_end:
997 }
998 \prg_new_conditional:Npnn \__zrefcheck_check_secsbefore:nn #1#2 { F }
999 {
1000     \group_begin:
1001     \bool_set_true:N \l__zrefcheck_integer_bool
1002     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
1003     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
1004     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
1005     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
1006     \bool_lazy_and:nnTF
1007     { \l__zrefcheck_integer_bool }
1008     {
1009         \int_compare_p:nNn
1010         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
1011         \int_compare_p:nNn
1012         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
1013         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
1014         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
1015     }
1016     { \group_insert_after:N \prg_return_true: }

```


\int_compare_p:nNn	634, 636, 637, 654, 656, 657, 704, 706, 707, 722, 724, 725, 740, 742, 743, 759, 761, 762, 782, 787, 791, 792, 807, 811, 812, 833, 835, 836, 851, 853, 868, 870, 871, 886, 888, 903, 905, 906, 923, 925, 927, 928, 945, 947, 949, 966, 968, 970, 971, 988, 990, 992, 1009, 1011, 1013, 1014
\int_eval:n	221, 356
\int_gincr:N	23, 29, 382
\int_if_even_p:n	786
\int_if_odd_p:n	781
\int_incr:N	298
\int_new:N	19, 20, 215, 371, 621, 622, 623, 624
\int_set:Nn	221, 224, 356
\int_use:N	26, 30, 300, 316
\int_zero:N	24, 282
\l_tmpa_int	282, 298, 300
int internal commands:	
_int_to_roman:w	4, 5, 94
ior commands:	
\ior_close:N	309
\ior_map_variable:NNn	286
\ior_open:Nn	280
\g_tmpa_ior	280, 286, 309
iow commands:	
\iow_newline:	48, 53, 64, 69, 73, 76, 81, 86
K	
keys commands:	
\keys_define:nn	113, 149, 156, 185, 189, 216, 232, 263, 434
\keys_set:nn	275, 381
L	
\label	11
legacy commands:	
\legacy_if_p:n	778
\let	312
M	
\MessageBreak	10
msg commands:	
\msg_line_number:	49, 54, 57, 59, 61, 65, 82, 87
\msg_new:nnn	46, 51, 56, 58, 60, 62, 67, 72, 74, 79, 84, 89
\msg_warning:nn	146, 152, 180, 223
\msg_warning:nnn	259, 267, 616
\msg_warning:nnnn	325, 334, 342, 363
N	
\newcounter	313
\NewDocumentCommand	274, 368, 412
\NewDocumentEnvironment	422
\noexpand	70
P	
\PackageError	7
\pageref	11
prg commands:	
\prg_new_conditional:Npnn	16, 95, 625, 643, 665, 675, 685, 695, 713, 731, 750, 769, 798, 818, 824, 842, 859, 877, 894, 912, 934, 955, 977, 998
\prg_new_protected_conditional:Npnn	105
\prg_return_false:	16, 98, 102, 109, 640, 660, 662, 670, 673, 682, 692, 710, 728, 746, 765, 795, 815, 821, 839, 856, 874, 891, 909, 931, 952, 974, 995, 1017
\prg_return_true:	16, 101, 108, 639, 659, 671, 678, 681, 688, 691, 709, 727, 745, 764, 794, 814, 822, 838, 855, 873, 890, 908, 930, 951, 973, 994, 1016
\ProcessKeysOptions	273
prop commands:	
\prop_get:NnNTF	323
\prop_gput:Nnn	299
\prop_new:N	276
\providecommand	3
\ProvidesExplPackage	14
R	
\refstepcounter	11, 16, 415, 424
regex commands:	
\regex_match:nnTF	107
\RequirePackage	16, 17, 18, 140, 272
\romannumeral	4
S	
\section	28
seq commands:	
\seq_if_in:NnTF	579
\seq_map_inline:Nn	538
\seq_new:N	431, 546
\seq_put_right:Nn	437, 441, 445, 449, 453, 457, 461, 465, 469, 473, 477, 481, 485, 489, 493, 497, 501, 505, 509, 513, 517, 521, 525, 529
\seq_set_from_clist:Nn	547
\setcounter	315
\SplitList	369

sys commands:

\c_sys_jobname_str 277

T

T_EX and L^AT_EX 2_ε commands:

\@addtoreset 312
 \@currentlabel 16
 \@ifl@t@r 3
 \@ifpackageloaded 136
 \@newl@bel 9
 \ltx@gobbletwo 312
 \zref@addprop 17, 27, 31, 141
 \zref@addprops 33
 \zref@extractdefault 12, 339, 608, 613
 \zref@ifpropundefined 333
 \zref@ifrefcontainsprop 336
 \zref@ifrefundefined
 12, 330, 360, 390, 553, 568, 587
 \ZREF@label 11
 \zref@label 10, 11
 \zref@labelbylist
 242, 387, 406, 419, 429
 \ZREF@mainlist 27, 31
 \zref@newlabel 9–11, 290
 \zref@newlist 32
 \zref@newprop 17, 26, 30
 \zref@refused 12, 407
 \zref@require@unique 11
 \zref@wrapper@babel
 9, 14, 370, 416, 418, 425, 428

tl commands:

\c_empty_tl 12, 339
 \tl_clear:N 12, 248, 283, 284, 320
 \tl_head:n 388
 \tl_if_blank:nTF 4, 247
 \tl_if_empty:nTF 4, 97, 100, 556
 \tl_if_eq:NnTF 290
 \tl_if_eq:nnTF 321
 \tl_map_break: 303
 \tl_map_function:nN 407
 \tl_map_inline:nn 536
 \tl_map_variable:NNn 288
 \tl_new:N
 155, 230, 350, 372, 373, 374, 375
 \tl_set:Nn 160, 162, 164, 168, 169,
 174, 175, 236, 277, 338, 383, 385, 388
 \g_tmpa_tl 277, 278, 280
 \l_tmpa_tl 283, 286, 288
 \l_tmpb_tl 284, 288, 290, 300

U

use commands:

\use:N 42, 562, 565, 571, 574

Z

\Z 107
 \zcheck 14, 16, 17, 70, 368
 zcregion 422
 \zctarget 8, 16, 412
 \zref 14

zrefcheck commands:

\zrefcheck_get_asint:nnn
 13, 16, 351, 629, 630, 649, 650,
 699, 700, 717, 718, 735, 736, 754,
 755, 773, 774, 802, 803, 828, 829,
 846, 847, 863, 864, 881, 882, 898,
 899, 916, 917, 918, 919, 938, 939,
 940, 941, 959, 960, 961, 962, 981,
 982, 983, 984, 1002, 1003, 1004, 1005
 \zrefcheck_get_astl:nnn
 12, 13, 16, 318, 353, 399

zrefcheck internal commands:

\g__zrefcheck_abschap_int . . . 19, 23, 26
 \g__zrefcheck_abssec_int 19, 24, 29, 30
 \g__zrefcheck_auxfile_lblseq_
 prop 12, 17, 276, 299, 323
 __zrefcheck_check_<check>:nn . . . 16
 __zrefcheck_check_above:nn . . . 643
 __zrefcheck_check_above:nnTF . . .
 669, 680
 __zrefcheck_check_after:nn . . . 675
 __zrefcheck_check_before:nn . . . 675
 __zrefcheck_check_below:nn . . . 643
 __zrefcheck_check_below:nnTF . . . 690
 __zrefcheck_check_chapsafter:nn 824
 __zrefcheck_check_chapsbefore:nn
 824
 __zrefcheck_check_close:nn . . . 798
 __zrefcheck_check_close:nnTF . . . 820
 __zrefcheck_check_facing:nn . . . 695
 __zrefcheck_check_far:nn 798
 __zrefcheck_check_lblfmt:n
 12, 316, 384
 __zrefcheck_check_nextchap:nn . . . 824
 __zrefcheck_check_nextpage:nn . . . 695
 __zrefcheck_check_nextsec:nn . . . 912
 __zrefcheck_check_pagesafter:nn 695
 __zrefcheck_check_pagesafter:nnTF
 687, 768
 __zrefcheck_check_pagesbefore:nn
 695
 __zrefcheck_check_pagesbefore:nnTF
 677, 749
 __zrefcheck_check_ppafter:nn . . . 695
 __zrefcheck_check_ppafter:nnTF 768
 __zrefcheck_check_ppbefore:nn . . . 695
 __zrefcheck_check_ppbefore:nnTF 749
 __zrefcheck_check_prevchap:nn . . . 824

_zrefcheck_check_prevpage:nn .	695	\l_zrefcheck_link_label_tl	
_zrefcheck_check_prevsec:nn . .	912	371, 388, 390, 400
_zrefcheck_check_secsafter:nn	912	\l_zrefcheck_link_star_bool . . .	
_zrefcheck_check_secsbefore:nn	912	371, 389, 396
_zrefcheck_check_thischap:nn .	824	_zrefcheck_message:nnnn	40, 607, 612
_zrefcheck_check_thispage:nn .	625	\l_zrefcheck_msglevel_tl . . .	42, 155
_zrefcheck_check_thispage:nnTF		\l_zrefcheck_msgonpage_bool	188, 603
.....	581, 584, 590, 593, 646, 667	\l_zrefcheck_onpage_bool	
_zrefcheck_check_thissec:nn . .	912	544, 559, 583, 586, 592, 596, 604
\l_zrefcheck_checkbeg_tl		\c_zrefcheck_onpage_checks_seq .	
.....	371, 383, 386, 387	544, 579
\l_zrefcheck_checkend_tl		\l_zrefcheck_passedcheck_bool . .	
.....	371, 385, 406	544, 558, 564, 567, 573, 577, 599
\l_zrefcheck_close_range_int . . .		\l_zrefcheck_propval_tl	
.....	215, 810	350, 353, 354, 356
_zrefcheck_do_check:nnn	19, 539, 550	\l_zrefcheck_ref_b_int	
_zrefcheck_end_lblfmt:n	621, 919, 924,
. 12, 317, 386, 419, 429, 566, 568,			941, 946, 962, 967, 984, 989, 1005, 1010
572, 575, 576, 585, 587, 591, 594, 595		\l_zrefcheck_ref_int	
_zrefcheck_get_asint:nnn	12	621, 630, 635, 637, 650,
_zrefcheck_get_astl:nnn	12		655, 657, 700, 705, 707, 718, 723,
\g_zrefcheck_id_int	371, 382, 384		725, 736, 741, 743, 755, 760, 762,
_zrefcheck_int_to_roman:w	94		774, 781, 783, 786, 788, 792, 803,
\l_zrefcheck_integer_bool			808, 812, 829, 834, 836, 847, 852,
.....	13, 349, 359,		864, 869, 871, 882, 887, 899, 904,
628, 632, 648, 652, 698, 702, 716,			906, 917, 926, 928, 939, 948, 960,
720, 734, 738, 753, 757, 772, 776,			969, 971, 982, 991, 1003, 1012, 1014
801, 805, 827, 831, 845, 849, 862,		_zrefcheck_run_checks:nnn	
866, 880, 884, 897, 901, 915, 921,		19, 408, 533
937, 943, 958, 964, 980, 986, 1001, 1007		_zrefcheck_set_checks_keys: . . .	
_zrefcheck_is_integer:n	5, 94	17, 380, 432
_zrefcheck_is_integer:nTF . . .	354	_zrefcheck_target_label:n	
_zrefcheck_is_integer_rgx:n	5, 105	241, 251, 416, 425
_zrefcheck_is_integer_rgx:nTF	220	\l_zrefcheck_target_label_bool .	
\l_zrefcheck_lbl_b_int	231, 237, 245
.....	621, 918, 924,	\l_zrefcheck_target_label_tl . . .	
940, 946, 961, 967, 983, 989, 1004, 1010		230, 247, 248, 249, 254, 260
\l_zrefcheck_lbl_int		\l_zrefcheck_use_hyperref_bool .	
. 621, 629, 635, 636, 649, 655, 656,		111, 138, 147, 395
699, 705, 706, 717, 723, 724, 735,		\l_zrefcheck_warn_hyperref_bool	
741, 742, 754, 760, 761, 773, 783,		111, 145
788, 791, 802, 808, 811, 828, 834,		_zrefcheck_zcheck:nnnn	14, 370, 377
835, 846, 852, 853, 863, 869, 870,		_zrefcheck_zcheck:nnnnn .	12, 14, 19
881, 887, 888, 898, 904, 905, 916,		\l_zrefcheck_zcheck_checks_seq .	
926, 927, 938, 948, 949, 959, 969,		409, 431, 437, 441,
970, 981, 991, 992, 1002, 1012, 1013			445, 449, 453, 457, 461, 465, 469,
\l_zrefcheck_link_anchor_tl . . .			473, 477, 481, 485, 489, 493, 497,
.....	371, 401, 402	\zrefchecksetup	9, 274