

The zref-check package implementation*

Gustavo Barros[†]

2021-08-04

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	Running	17
6.2	Conditionals	19
6.2.1	This page	19
6.2.2	On page	20
6.2.3	Before / After	20
6.2.4	Pages	21
6.2.5	Close / Far	23
6.2.6	Chapter	23
6.2.7	Section	25
	Index	28

*This file describes v0.1.0, last revised 2021-08-04.

[†]<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} {2021-08-04} {0.1.0}
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
`\g__zrefcheck_abssec_int` 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 { abschap } [0] { \int_use:N \g__zrefcheck_abschap_int }
27 \zref@addprop \ZREF@mainlist { abschap }
```

```

28 \AddToHook { cmd / section / before }
29 { \int_gincr:N \g__zrefcheck_abssec_int }
30 \zref@newprop { abssec } [0] { \int_use:N \g__zrefcheck_abssec_int }
31 \zref@addprop \ZREF@mainlist { 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     abspage ,
36     abschap ,
37     abssec ,
38     page
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 \AtBeginDocument
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 \AtBeginDocument
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 \AtBeginDocument
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. 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-abspace.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:nnnnn`.

```

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

```

```

368 \NewDocumentCommand \zcheck
369 { s O { } } > { \SplitList { , } } m > { \SplitList { , } } 0 { } m }
370 { \zref@wrapper@babel \__zrefcheck_zcheck:nnnnn {#3} {#1} {#2} {#4} {#5} }

```

(End definition for `\zcheck`.)

```

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

```

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

`__zrefcheck_zcheck:nnnnn` 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:nnnnn {\labels} {\(*)} {\options} {\checks} {\text}

```

```

377 \cs_new:Npn \__zrefcheck_zcheck:nnnnn #1#2#3#4#5
378 {
379     \group_begin:

```

Process local options.

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

Names of the labels for this zrefcheck call.

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

Set checkbeg label.

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

Typeset $\langle\textit{text}\rangle$, 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.

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

If the reference is undefined, just typeset.

```
390 {#5}
391 {
392   \bool_if:nTF
393   {
394     \l__zrefcheck_use_hyperref_bool &&
395     ! \l__zrefcheck_link_star_tl
396   }
397   {
398     \exp_args:Nx \zrefcheck_get_astl:nnn
399     { \l__zrefcheck_link_label_tl }
400     { anchor } { \l__zrefcheck_link_anchor_tl }
401     \hyperlink { \l__zrefcheck_link_anchor_tl } {#5}
402   }
403   {#5}
404 }
```

Set checkend label.

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

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

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

Run the checks.

```
407 \__zrefcheck_run_checks:nnV {#4} {#1} { \l__zrefcheck_checkbeg_tl }
408 \group_end:
409 }
```

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

5.2 Targets

```

\zctarget      \zctarget{<label>}{<text>}
410 \NewDocumentCommand \zctarget { m +m }
411 {
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.
412 \group_begin:
413 \refstepcounter { zrefcheck }
414 \zref@wrapper@babel \_zrefcheck_target_label:n {#1}
415 #2
416 \zref@wrapper@babel
417 \zref@labelbylist { \_zrefcheck_end_lblfmt:n {#1} } { zrefcheck }
418 \group_end:
419 }

(End definition for \zctarget.)

zcregion      \begin{zcregion}{<label>}
              ...
              \end{zcregion}
420 \NewDocumentEnvironment {zcregion} { m }
421 {
422 \refstepcounter { zrefcheck }
423 \zref@wrapper@babel \_zrefcheck_target_label:n {#1}
424 }
425 {
426 \zref@wrapper@babel
427 \zref@labelbylist { \_zrefcheck_end_lblfmt:n {#1} } { zrefcheck }
428 }

(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.

6.1 Running

```
\__zrefcheck_run_checks:nnn      \__zrefcheck_run_checks:nnn {\check} {\label} {\reference}
\__zrefcheck_run_checks:nnV
429 \cs_new:Npn \__zrefcheck_run_checks:nnn #1#2#3
430 {
431   \group_begin:
432     \tl_map_inline:nn {#2}
433     {
434       \tl_map_inline:nn {#1}
435       { \__zrefcheck_do_check:nnn {####1} {##1} {#3} }
436     }
437   \group_end:
438 }
439 \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
440 \bool_new:N \l__zrefcheck_passedcheck_bool
441 \bool_new:N \l__zrefcheck_onpage_bool
442 \seq_new:N \c__zrefcheck_onpage_checks_seq
443 \seq_set_from_clist:Nn \c__zrefcheck_onpage_checks_seq
444 { 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`.

```
445 \cs_generate_variant:Nn \exp_args:Nno { Nnoo }
```

```
\__zrefcheck_do_check:nnn      \__zrefcheck_do_check:nnn {\check} {\label beg} {\reference beg}
```

```
446 \cs_new:Npn \__zrefcheck_do_check:nnn #1#2#3
447 {
448   \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.

```
449   \zref@ifrefundefined {#2}
450   {}
451   {
452     \tl_if_empty:nF {#1}
453     {
454       \bool_set_true:N \l__zrefcheck_passedcheck_bool
455       \bool_set_false:N \l__zrefcheck_onpage_bool
456       \cs_if_exist:CTF { __zrefcheck_check_ #1 :nnF }
457       {
```

“label beg” vs “reference beg”.

```

458         \use:c { __zrefcheck_check_ #1 :nnF }
459         {#2} {#3}
460         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }

```

“label beg” vs “reference end”.

```

461         \exp_args:Nnno \use:c { __zrefcheck_check_ #1 :nnF }
462         {#2} { \__zrefcheck_end_lblfmt:n {#3} }
463         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }

```

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

```

464         \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
465         {}
466         {

```

“label end” vs “reference beg”.

```

467         \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
468         { \__zrefcheck_end_lblfmt:n {#2} } {#3}
469         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }

```

“label end” vs “reference end”.

```

470         \exp_args:Nnoo \use:c { __zrefcheck_check_ #1 :nnF }
471         { \__zrefcheck_end_lblfmt:n {#2} }
472         { \__zrefcheck_end_lblfmt:n {#3} }
473         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
474     }

```

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.

```

475         \seq_if_in:NnT \c__zrefcheck_onpage_checks_seq {#1}
476         {
477             \__zrefcheck_check_thispage:nnT
478             {#2} {#3}
479             { \bool_set_true:N \l__zrefcheck_onpage_bool }
480             \__zrefcheck_check_thispage:nnT
481             {#2} { \__zrefcheck_end_lblfmt:n {#3} }
482             { \bool_set_true:N \l__zrefcheck_onpage_bool }
483             \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
484             {}
485             {
486                 \__zrefcheck_check_thispage:nnT
487                 { \__zrefcheck_end_lblfmt:n {#2} } {#3}
488                 { \bool_set_true:N \l__zrefcheck_onpage_bool }
489                 \__zrefcheck_check_thispage:nnT
490                 { \__zrefcheck_end_lblfmt:n {#2} }
491                 { \__zrefcheck_end_lblfmt:n {#3} }
492                 { \bool_set_true:N \l__zrefcheck_onpage_bool }
493             }
494         }
495         \bool_if:NTF \l__zrefcheck_passedcheck_bool
496         {
497             \bool_if:nT
498             {
499                 \l__zrefcheck_msgonpage_bool &&
500                 \l__zrefcheck_onpage_bool

```

```

501         }
502         {
503             \__zrefcheck_message:nnnx { double-check } {#1} {#2}
504             { \zref@extractdefault {#3} {page} {'unknown'} }
505         }
506     }
507     {
508         \__zrefcheck_message:nnnx { check-failed } {#1} {#2}
509         { \zref@extractdefault {#3} {page} {'unknown'} }
510     }
511 }
512 { \msg_warning:nnn { zref-check } { check-missing } {#1} }
513 }
514 }
515 \group_end:
516 }

```

(End definition for __zrefcheck_do_check:nnn.)

6.2 Conditionals

\l__zrefcheck_lbl_int More readable scratch variables for the tests.
\l__zrefcheck_ref_int 517 \int_new:N \l__zrefcheck_lbl_int
\l__zrefcheck_lbl_b_int 518 \int_new:N \l__zrefcheck_ref_int
\l__zrefcheck_ref_b_int 519 \int_new:N \l__zrefcheck_lbl_b_int
520 \int_new:N \l__zrefcheck_ref_b_int

(End definition for \l__zrefcheck_lbl_int and others.)

6.2.1 This page

```

\__zrefcheck_check_thispage:nn
521 \prg_new_conditional:Npnn \__zrefcheck_check_thispage:nn #1#2 { T , F , TF }
522 {
523     \group_begin:
524     \bool_set_true:N \l__zrefcheck_integer_bool
525     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
526     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
527     \bool_lazy_and:nnTF
528     { \l__zrefcheck_integer_bool }
529     {
530         \int_compare_p:nNn
531         { \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.
532         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
533         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
534     }
535     { \group_insert_after:N \prg_return_true: }
536     { \group_insert_after:N \prg_return_false: }
537 \group_end:
538 }

```

(End definition for `_zrefcheck_check_thispage:nn`.)

6.2.2 On page

```

\_zrefcheck\_check\_above:nn
\_zrefcheck\_check\_below:nn
539 \prg\_new\_conditional:Npnn \_zrefcheck\_check\_above:nn #1#2 { F , TF }
540 {
541   \group\_begin:
542     \_zrefcheck\_check\_thispage:nnTF {#1} {#2}
543     {
544       \bool\_set\_true:N \l\_zrefcheck\_integer\_bool
545       \zrefcheck\_get\_asint:nnn {#1} { lblseq } { \l\_zrefcheck\_lbl\_int }
546       \zrefcheck\_get\_asint:nnn {#2} { lblseq } { \l\_zrefcheck\_ref\_int }
547       \bool\_lazy\_and:nnTF
548       { \l\_zrefcheck\_integer\_bool }
549       {
550         \int\_compare\_p:nNn
551         { \l\_zrefcheck\_lbl\_int } < { \l\_zrefcheck\_ref\_int } &&
552         ! \int\_compare\_p:nNn { \l\_zrefcheck\_lbl\_int } = { 0 } &&
553         ! \int\_compare\_p:nNn { \l\_zrefcheck\_ref\_int } = { 0 }
554       }
555       { \group\_insert\_after:N \prg\_return\_true: }
556       { \group\_insert\_after:N \prg\_return\_false: }
557     }
558     { \group\_insert\_after:N \prg\_return\_false: }
559   \group\_end:
560 }
561 \prg\_new\_conditional:Npnn \_zrefcheck\_check\_below:nn #1#2 { F , TF }
562 {
563   \_zrefcheck\_check\_thispage:nnTF {#1} {#2}
564   {
565     \_zrefcheck\_check\_above:nnTF {#1} {#2}
566     { \prg\_return\_false: }
567     { \prg\_return\_true: }
568   }
569   { \prg\_return\_false: }
570 }

```

(End definition for `_zrefcheck_check_above:nn` and `_zrefcheck_check_below:nn`.)

6.2.3 Before / After

```

\_zrefcheck\_check\_before:nn
\_zrefcheck\_check\_after:nn
571 \prg\_new\_conditional:Npnn \_zrefcheck\_check\_before:nn #1#2 { F }
572 {
573   \_zrefcheck\_check\_pagesbefore:nnTF {#1} {#2}
574   { \prg\_return\_true: }
575   {
576     \_zrefcheck\_check\_above:nnTF {#1} {#2}
577     { \prg\_return\_true: }
578     { \prg\_return\_false: }
579   }
580 }
581 \prg\_new\_conditional:Npnn \_zrefcheck\_check\_after:nn #1#2 { F }

```

```

582 {
583   \_zrefcheck\_check\_pagesafter:nnTF {#1} {#2}
584   { \prg\_return\_true: }
585   {
586     \_zrefcheck\_check\_below:nnTF {#1} {#2}
587     { \prg\_return\_true: }
588     { \prg\_return\_false: }
589   }
590 }

```

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

6.2.4 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
591 \prg\_new\_conditional:Npnn \_zrefcheck\_check\_nextpage:nn #1#2 { F }
592 {
593   \group\_begin:
594   \bool\_set\_true:N \l\_zrefcheck\_integer\_bool
595   \zrefcheck\_get\_asint:nnn {#1} { abspage } { \l\_zrefcheck\_lbl\_int }
596   \zrefcheck\_get\_asint:nnn {#2} { abspage } { \l\_zrefcheck\_ref\_int }
597   \bool\_lazy\_and:nnTF
598   { \l\_zrefcheck\_integer\_bool }
599   {
600     \int\_compare\_p:nNn
601     { \l\_zrefcheck\_lbl\_int } = { \l\_zrefcheck\_ref\_int + 1 } &&
602     ! \int\_compare\_p:nNn { \l\_zrefcheck\_lbl\_int } = { 0 } &&
603     ! \int\_compare\_p:nNn { \l\_zrefcheck\_ref\_int } = { 0 }
604   }
605   { \group\_insert\_after:N \prg\_return\_true: }
606   { \group\_insert\_after:N \prg\_return\_false: }
607   \group\_end:
608 }
609 \prg\_new\_conditional:Npnn \_zrefcheck\_check\_prevpage:nn #1#2 { F }
610 {
611   \group\_begin:
612   \bool\_set\_true:N \l\_zrefcheck\_integer\_bool
613   \zrefcheck\_get\_asint:nnn {#1} { abspage } { \l\_zrefcheck\_lbl\_int }
614   \zrefcheck\_get\_asint:nnn {#2} { abspage } { \l\_zrefcheck\_ref\_int }
615   \bool\_lazy\_and:nnTF
616   { \l\_zrefcheck\_integer\_bool }
617   {
618     \int\_compare\_p:nNn
619     { \l\_zrefcheck\_lbl\_int } = { \l\_zrefcheck\_ref\_int - 1 } &&
620     ! \int\_compare\_p:nNn { \l\_zrefcheck\_lbl\_int } = { 0 } &&
621     ! \int\_compare\_p:nNn { \l\_zrefcheck\_ref\_int } = { 0 }
622   }
623   { \group\_insert\_after:N \prg\_return\_true: }
624   { \group\_insert\_after:N \prg\_return\_false: }
625   \group\_end:
626 }
627 \prg\_new\_conditional:Npnn \_zrefcheck\_check\_pagesbefore:nn #1#2 { F , TF }
628 {
629   \group\_begin:

```

```

630 \bool_set_true:N \l__zrefcheck_integer_bool
631 \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
632 \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
633 \bool_lazy_and:nnTF
634 { \l__zrefcheck_integer_bool }
635 {
636   \int_compare_p:nNn
637     { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
638     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
639     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
640 }
641 { \group_insert_after:N \prg_return_true: }
642 { \group_insert_after:N \prg_return_false: }
643 \group_end:
644 }
645 \cs_new_eq:NN \__zrefcheck_check_ppbefore:nnF \__zrefcheck_check_pagesbefore:nnF
646 \prg_new_conditional:Npnn \__zrefcheck_check_pagesafter:nn #1#2 { F , TF }
647 {
648   \group_begin:
649     \bool_set_true:N \l__zrefcheck_integer_bool
650     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
651     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
652     \bool_lazy_and:nnTF
653     { \l__zrefcheck_integer_bool }
654     {
655       \int_compare_p:nNn
656         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
657         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
658         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
659     }
660     { \group_insert_after:N \prg_return_true: }
661     { \group_insert_after:N \prg_return_false: }
662   \group_end:
663 }
664 \cs_new_eq:NN \__zrefcheck_check_ppafter:nnF \__zrefcheck_check_pagesafter:nnF
665 \prg_new_conditional:Npnn \__zrefcheck_check_facing:nn #1#2 { F }
666 {
667   \group_begin:
668     \bool_set_true:N \l__zrefcheck_integer_bool
669     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
670     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
671     \bool_lazy_and:nnTF
672     { \l__zrefcheck_integer_bool }
673     {

```

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

```

674 \legacy_if_p:n { @twoside } &&

```

Now we test “facing”.

```

675 (
676   (
677     \int_if_odd_p:n { \l__zrefcheck_ref_int } &&
678     \int_compare_p:nNn
679       { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 }
680   ) ||

```

```

681      (
682        \int_if_even_p:n { \l__zrefcheck_ref_int } &&
683        \int_compare_p:nNn
684          { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 }
685      )
686    ) &&
687    ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
688    ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
689  }
690  { \group_insert_after:N \prg_return_true: }
691  { \group_insert_after:N \prg_return_false: }
692 \group_end:
693 }

```

(End definition for __zrefcheck_check_nextpage:nn and others.)

6.2.5 Close / Far

```

\__zrefcheck_check_close:nn
\__zrefcheck_check_far:nn
694 \prg_new_conditional:Npnn \__zrefcheck_check_close:nn #1#2 { F , TF }
695 {
696   \group_begin:
697     \bool_set_true:N \l__zrefcheck_integer_bool
698     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
699     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
700     \bool_lazy_and:nnTF
701       { \l__zrefcheck_integer_bool }
702     {
703       \int_compare_p:nNn
704         { \int_abs:n { \l__zrefcheck_lbl_int - \l__zrefcheck_ref_int } }
705       <
706       { \l__zrefcheck_close_range_int + 1 } &&
707       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
708       ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
709     }
710     { \group_insert_after:N \prg_return_true: }
711     { \group_insert_after:N \prg_return_false: }
712   \group_end:
713 }
714 \prg_new_conditional:Npnn \__zrefcheck_check_far:nn #1#2 { F }
715 {
716   \__zrefcheck_check_close:nnTF {#1} {#2}
717   { \prg_return_false: }
718   { \prg_return_true: }
719 }

```

(End definition for __zrefcheck_check_close:nn and __zrefcheck_check_far:nn.)

6.2.6 Chapter

```

\__zrefcheck_check_thischap:nn
\__zrefcheck_check_nextchap:nn
\__zrefcheck_check_prevchap:nn
\__zrefcheck_check_chapsafter:nn
\__zrefcheck_check_chapsbefore:nn
720 \prg_new_conditional:Npnn \__zrefcheck_check_thischap:nn #1#2 { F }
721 {
722   \group_begin:

```

```

723 \bool_set_true:N \l__zrefcheck_integer_bool
724 \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
725 \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
726 \bool_lazy_and:nnTF
727 { \l__zrefcheck_integer_bool }
728 {
729   \int_compare_p:nNn
730   { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int } &&

```

‘0’ is the default value of `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.

```

731   ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
732   ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
733 }
734 { \group_insert_after:N \prg_return_true: }
735 { \group_insert_after:N \prg_return_false: }
736 \group_end:
737 }
738 \prg_new_conditional:Npnn \__zrefcheck_check_nextchap:nn #1#2 { F }
739 {
740   \group_begin:
741   \bool_set_true:N \l__zrefcheck_integer_bool
742   \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
743   \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
744   \bool_lazy_and:nnTF
745   { \l__zrefcheck_integer_bool }
746   {
747     \int_compare_p:nNn
748     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
749     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
750   }
751   { \group_insert_after:N \prg_return_true: }
752   { \group_insert_after:N \prg_return_false: }
753   \group_end:
754 }
755 \prg_new_conditional:Npnn \__zrefcheck_check_prevchap:nn #1#2 { F }
756 {
757   \group_begin:
758   \bool_set_true:N \l__zrefcheck_integer_bool
759   \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
760   \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
761   \bool_lazy_and:nnTF
762   { \l__zrefcheck_integer_bool }
763   {
764     \int_compare_p:nNn
765     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
766     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
767     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
768   }
769   { \group_insert_after:N \prg_return_true: }
770   { \group_insert_after:N \prg_return_false: }

```



```

771 \group_end:
772 }
773 \prg_new_conditional:Npnn \__zrefcheck_check_chapsafter:nn #1#2 { F }
774 {
775   \group_begin:
776     \bool_set_true:N \l__zrefcheck_integer_bool
777     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
778     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
779     \bool_lazy_and:nnTF
780       { \l__zrefcheck_integer_bool }
781       {
782         \int_compare_p:nNn
783           { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
784           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
785       }
786     { \group_insert_after:N \prg_return_true: }
787     { \group_insert_after:N \prg_return_false: }
788   \group_end:
789 }
790 \prg_new_conditional:Npnn \__zrefcheck_check_chapsbefore:nn #1#2 { F }
791 {
792   \group_begin:
793     \bool_set_true:N \l__zrefcheck_integer_bool
794     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
795     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
796     \bool_lazy_and:nnTF
797       { \l__zrefcheck_integer_bool }
798       {
799         \int_compare_p:nNn
800           { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
801           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
802           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
803       }
804     { \group_insert_after:N \prg_return_true: }
805     { \group_insert_after:N \prg_return_false: }
806   \group_end:
807 }

```

(End definition for __zrefcheck_check_thischap:nn and others.)

6.2.7 Section

```

\__zrefcheck_check_thissec:nn
\__zrefcheck_check_nextsec:nn
\__zrefcheck_check_prevsec:nn
\__zrefcheck_check_secsafter:nn
\__zrefcheck_check_secsbefore:nn
808 \prg_new_conditional:Npnn \__zrefcheck_check_thissec:nn #1#2 { F }
809 {
810   \group_begin:
811     \bool_set_true:N \l__zrefcheck_integer_bool
812     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
813     \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
814     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
815     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
816     \bool_lazy_and:nnTF
817       { \l__zrefcheck_integer_bool }
818       {

```

```

819         \int_compare_p:nNn
820         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
821         \int_compare_p:nNn
822         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int } &&

```

‘0’ is the default value of `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.

```

823         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
824         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
825     }
826     { \group_insert_after:N \prg_return_true: }
827     { \group_insert_after:N \prg_return_false: }
828 \group_end:
829 }
830 \prg_new_conditional:Npnn \__zrefcheck_check_nextsec:nn #1#2 { F }
831 {
832     \group_begin:
833     \bool_set_true:N \l__zrefcheck_integer_bool
834     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
835     \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
836     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
837     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
838     \bool_lazy_and:nnTF
839     { \l__zrefcheck_integer_bool }
840     {
841         \int_compare_p:nNn
842         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
843         \int_compare_p:nNn
844         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
845         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
846     }
847     { \group_insert_after:N \prg_return_true: }
848     { \group_insert_after:N \prg_return_false: }
849 \group_end:
850 }
851 \prg_new_conditional:Npnn \__zrefcheck_check_prevsec:nn #1#2 { F }
852 {
853     \group_begin:
854     \bool_set_true:N \l__zrefcheck_integer_bool
855     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
856     \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
857     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
858     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
859     \bool_lazy_and:nnTF
860     { \l__zrefcheck_integer_bool }
861     {
862         \int_compare_p:nNn
863         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
864         \int_compare_p:nNn
865         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
866         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&

```

```

867         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
868     }
869     { \group_insert_after:N \prg_return_true: }
870     { \group_insert_after:N \prg_return_false: }
871 \group_end:
872 }
873 \prg_new_conditional:Npnn \__zrefcheck_check_secsafter:nn #1#2 { F }
874 {
875     \group_begin:
876     \bool_set_true:N \l__zrefcheck_integer_bool
877     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
878     \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
879     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
880     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
881     \bool_lazy_and:nnTF
882     { \l__zrefcheck_integer_bool }
883     {
884         \int_compare_p:nNn
885         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
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_secsbefore:nn #1#2 { F }
895 {
896     \group_begin:
897     \bool_set_true:N \l__zrefcheck_integer_bool
898     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
899     \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
900     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
901     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
902     \bool_lazy_and:nnTF
903     { \l__zrefcheck_integer_bool }
904     {
905         \int_compare_p:nNn
906         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
907         \int_compare_p:nNn
908         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
909         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
910         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
911     }
912     { \group_insert_after:N \prg_return_true: }
913     { \group_insert_after:N \prg_return_false: }
914 \group_end:
915 }

```

(End definition for __zrefcheck_check_thissec:nn and others.)

```

916 \endpackage

```

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		
\A	107	\exp_args:Nno 467
\AddToHook	21, 28	\exp_args:Nnoo 470
\AtBeginDocument	134, 183, 243	\exp_args:Nx 398
		\exp_not:n 253
B		F
\beginpgroup	311	file commands:
bool commands:		\file_if_exist:nTF 278
\bool_if:NTF	138, 145, 245, 295, 495	\fmtversion 3
\bool_if:nTF	392, 497	
\bool_lazy_and:nnTF	13, 527, 547, 597,	G
	615, 633, 652, 671, 700, 726, 744,	group commands:
	761, 779, 796, 816, 838, 859, 881, 902	\group_begin: 281,
\bool_new:N	111, 112, 188, 231, 349, 376, 440, 441	379, 412, 431, 448, 523, 541, 593,
\bool_set:Nn	388	611, 629, 648, 667, 696, 722, 740,
\bool_set_false:N	119, 128, 129, 147, 194, 203, 210,	757, 775, 792, 810, 832, 853, 875, 896
	285, 297, 359, 455, 460, 463, 469, 473	\group_end: 308,
\bool_set_true:N	118, 123,	408, 418, 437, 515, 537, 559, 607,
	124, 198, 204, 209, 237, 292, 454,	625, 643, 662, 692, 712, 736, 753,
	479, 482, 488, 492, 524, 544, 594,	771, 788, 806, 828, 849, 871, 892, 914
	612, 630, 649, 668, 697, 723, 741,	\group_insert_after:N 535,
	758, 776, 793, 811, 833, 854, 876, 897	536, 555, 556, 558, 605, 606, 623,
\l_tmpa_bool	11, 285, 292, 295, 297	624, 641, 642, 660, 661, 690, 691,
		710, 711, 734, 735, 751, 752, 769,
		770, 786, 787, 804, 805, 826, 827,
		847, 848, 869, 870, 890, 891, 912, 913
C		H
\catcode	11	\hyperlink 14, 401
\chapter	2, 24	
cs commands:		I
\cs:w	254	\ifdraft 167, 202
\cs_end:	254	\IfFormatAtLeastTF 3, 4
\cs_generate_variant:Nn	45, 439, 445	\ifoptionfinal 173, 208
\cs_if_exist:NTF	249, 456	int commands:
\cs_new:Npn	40,	\int_abs:n 704
	241, 316, 317, 318, 351, 377, 429, 446	\int_compare_p:nNn
\cs_new_eq:NN	94, 645, 664	530, 532, 533, 550, 552, 553,
\cs_set:Npx	251	600, 602, 603, 618, 620, 621, 636,
		638, 639, 655, 657, 658, 678, 683,
D		687, 688, 703, 707, 708, 729, 731,
\d	107	732, 747, 749, 764, 766, 767, 782,
		784, 799, 801, 802, 819, 821, 823,
E		824, 841, 843, 845, 862, 864, 866,
\endgroup	314	867, 884, 886, 888, 905, 907, 909, 910
\endinput	12	\int_eval:n 221, 356
exp commands:		\int_gincr:N 23, 29, 381
\exp_args:Nnno	445, 461	\int_if_even_p:n 682
\exp_args:NnnV	259	\int_if_odd_p:n 677

<code>\int_incr:N</code>	298	<code>\prg_return_false:</code>	
<code>\int_new:N</code>	16, 98, 102, 109, 536, 556,
.	19, 20, 215, 371, 517, 518, 519, 520	558, 566, 569, 578, 588, 606, 624,
<code>\int_set:Nn</code>	221, 224, 356	642, 661, 691, 711, 717, 735, 752,
<code>\int_use:N</code>	26, 30, 300, 316	770, 787, 805, 827, 848, 870, 891, 913
<code>\int_zero:N</code>	24, 282	<code>\prg_return_true:</code>	
<code>\l_tmpa_int</code>	282, 298, 300	16, 101, 108, 535, 555,
int internal commands:		567, 574, 577, 584, 587, 605, 623,
<code>__int_to_roman:w</code>	4, 5, 94	641, 660, 690, 710, 718, 734, 751,
ior commands:		769, 786, 804, 826, 847, 869, 890, 912
<code>\ior_close:N</code>	309	<code>\ProcessKeysOptions</code>	273
<code>\ior_map_variable:Nnn</code>	286	prop commands:	
<code>\ior_open:Nn</code>	280	<code>\prop_get:NnNTF</code>	323
<code>\g_tmpa_ior</code>	280, 286, 309	<code>\prop_gput:Nnn</code>	299
iow commands:		<code>\prop_new:N</code>	276
<code>\iow_newline:</code>		<code>\providecommand</code>	3
.	48, 53, 64, 69, 73, 76, 81, 86	<code>\ProvidesExplPackage</code>	14
K		R	
keys commands:		<code>\refstepcounter</code>	11, 16, 413, 422
<code>\keys_define:nn</code>		regex commands:	
.	113, 149, 156, 185, 189, 216, 232, 263	<code>\regex_match:nnTF</code>	107
<code>\keys_set:nn</code>	275, 380	<code>\RequirePackage</code>	16, 17, 18, 140, 272
L		<code>\romannumeral</code>	4
legacy commands:		S	
<code>\legacy_if_p:n</code>	674	<code>\section</code>	26
<code>\let</code>	312	seq commands:	
M		<code>\seq_if_in:NnTF</code>	475
<code>\MessageBreak</code>	10	<code>\seq_new:N</code>	442
msg commands:		<code>\seq_set_from_clist:Nn</code>	443
<code>\msg_line_number:</code>		<code>\setcounter</code>	315
.	49, 54, 57, 59, 61, 65, 82, 87	<code>\SplitList</code>	369
<code>\msg_new:nnn</code>	46,	sys commands:	
.	51, 56, 58, 60, 62, 67, 72, 74, 79, 84, 89	<code>\c_sys_jobname_str</code>	277
<code>\msg_warning:nn</code>	146, 152, 180, 223	T	
<code>\msg_warning:nnn</code>	259, 267, 512	TeX and L ^A T _E X 2 _ε commands:	
<code>\msg_warning:nnnn</code>	325, 334, 342, 363	<code>\@addtoreset</code>	312
N		<code>\@currentlabel</code>	16
<code>\newcounter</code>	313	<code>\@ifl@t@r</code>	3
<code>\NewDocumentCommand</code>	274, 368, 410	<code>\@ifpackageloaded</code>	136
<code>\NewDocumentEnvironment</code>	420	<code>\@newl@bel</code>	9
<code>\noexpand</code>	70	<code>\ltx@gobbletwo</code>	312
P		<code>\zref@addprop</code>	17, 27, 31, 141
<code>\PackageError</code>	7	<code>\zref@addprops</code>	33
prg commands:		<code>\zref@extractdefault</code>	12, 339, 504, 509
<code>\prg_new_conditional:Npnn</code>	16, 95,	<code>\zref@ifpropundefined</code>	333
.	521, 539, 561, 571, 581, 591, 609,	<code>\zref@ifrefcontainsprop</code>	336
.	627, 646, 665, 694, 714, 720, 738,	<code>\zref@ifrefundefined</code>	
.	755, 773, 790, 808, 830, 851, 873, 894	12, 330, 360, 389, 449, 464, 483
<code>\prg_new_protected_conditional:Npnn</code>	105	<code>\ZREF@label</code>	11
		<code>\zref@label</code>	10, 11
		<code>\zref@labelbylist</code>	
		242, 386, 405, 417, 427

\ZREF@mainlist	27, 31	__zrefcheck_check_⟨check⟩:nn	16
\zref@newlabel	9–11, 290	__zrefcheck_check_above:nn	539
\zref@newlist	32	__zrefcheck_check_above:nnTF	565, 576
\zref@newprop	17, 26, 30	__zrefcheck_check_after:nn	571
\zref@refused	12, 406	__zrefcheck_check_before:nn	571
\zref@require@unique	11	__zrefcheck_check_below:nn	539
\zref@wrapper@babel	9, 14, 370, 414, 416, 423, 426	__zrefcheck_check_below:nnTF	586
tl commands:		__zrefcheck_check_chapsafter:nn	720
\c_empty_tl	12, 339	__zrefcheck_check_chapsbefore:nn	720
\tl_clear:N	12, 248, 283, 284, 320	__zrefcheck_check_close:nn	694
\tl_head:n	387	__zrefcheck_check_close:nnTF	716
\tl_if_blank:nnTF	4, 247	__zrefcheck_check_facing:nn	591
\tl_if_empty:nnTF	4, 97, 100, 452	__zrefcheck_check_far:nn	694
\tl_if_eq:NnTF	290	__zrefcheck_check_lblfmt:n	12, 316, 383
\tl_if_eq:nnTF	321	__zrefcheck_check_nextchap:nn	720
\tl_map_break:	303	__zrefcheck_check_nextpage:nn	591
\tl_map_function:nN	406	__zrefcheck_check_nextsec:nn	808
\tl_map_inline:nn	432, 434	__zrefcheck_check_pagesafter:nn	591
\tl_map_variable:NNn	288	__zrefcheck_check_pagesafter:nnTF	583, 664
\tl_new:N	155, 230, 350, 372, 373, 374, 375	__zrefcheck_check_pagesbefore:nn	591
\tl_set:Nn	160, 162, 164, 168, 169, 174, 175, 236, 277, 338, 382, 384, 387	__zrefcheck_check_pagesbefore:nnTF	573, 645
\g_tmpa_tl	277, 278, 280	__zrefcheck_check_ppafter:nn	591
\l_tmpa_tl	283, 286, 288	__zrefcheck_check_ppafter:nnTF	664
\l_tmpb_tl	284, 288, 290, 300	__zrefcheck_check_ppbefore:nn	591
		__zrefcheck_check_ppbefore:nnTF	645
U		__zrefcheck_check_prevchap:nn	720
use commands:		__zrefcheck_check_prevpage:nn	591
\use:N	42, 458, 461, 467, 470	__zrefcheck_check_prevsec:nn	808
Z		__zrefcheck_check_secsafter:nn	808
\Z	107	__zrefcheck_check_secsbefore:nn	808
\zcheck	14, 16, 70, 368	__zrefcheck_check_thischap:nn	720
zcregion	420	__zrefcheck_check_thispage:nn	521
\zctarget	8, 16, 410	__zrefcheck_check_thispage:nnTF	477, 480, 486, 489, 542, 563
\zref	14	__zrefcheck_check_thissec:nn	808
zrefcheck commands:		\l__zrefcheck_checkbeg_tl	371, 382, 385, 386, 407
\zrefcheck_get_asint:nnn	13, 16, 351, 525, 526, 545, 546, 595, 596, 613, 614, 631, 632, 650, 651, 669, 670, 698, 699, 724, 725, 742, 743, 759, 760, 777, 778, 794, 795, 812, 813, 814, 815, 834, 835, 836, 837, 855, 856, 857, 858, 877, 878, 879, 880, 898, 899, 900, 901	\l__zrefcheck_checkend_tl	371, 384, 405
\zrefcheck_get_astl:nnn	12, 13, 16, 318, 353, 398	\l__zrefcheck_close_range_int	215, 706
zrefcheck internal commands:		__zrefcheck_do_check:nnn	17, 435, 446
\g__zrefcheck_abschap_int	19, 23, 26	__zrefcheck_end_lblfmt:n	12, 317, 385, 417, 427, 462, 464, 468, 471, 472, 481, 483, 487, 490, 491
\g__zrefcheck_abssec_int	19, 24, 29, 30	__zrefcheck_get_asint:nnn	12
\g__zrefcheck_auxfile_lblseq_prop	12, 17, 276, 299, 323	__zrefcheck_get_astl:nnn	12
		\g__zrefcheck_id_int	371, 381, 383

_zrefcheck_int_to_roman:w	94	\c_zrefcheck_onpage_checks_seq .	440 , 475
\l_zrefcheck_integer_bool	13 , 349 , 359 , 524 , 528 , 544 , 548 , 594 , 598 , 612 , 616 , 630 , 634 , 649 , 653 , 668 , 672 , 697 , 701 , 723 , 727 , 741 , 745 , 758 , 762 , 776 , 780 , 793 , 797 , 811 , 817 , 833 , 839 , 854 , 860 , 876 , 882 , 897 , 903	\l_zrefcheck_passedcheck_bool . .	440 , 454 , 460 , 463 , 469 , 473 , 495
_zrefcheck_is_integer:n	5 , 94	\l_zrefcheck_propval_tl	350 , 353 , 354 , 356
_zrefcheck_is_integer:nTF . . .	354	\l_zrefcheck_ref_b_int	517 , 815 , 820 , 837 , 842 , 858 , 863 , 880 , 885 , 901 , 906
_zrefcheck_is_integer_rgx:n . .	5 , 105	\l_zrefcheck_ref_int	517 , 526 , 531 , 533 , 546 , 551 , 553 , 596 , 601 , 603 , 614 , 619 , 621 , 632 , 637 , 639 , 651 , 656 , 658 , 670 , 677 , 679 , 682 , 684 , 688 , 699 , 704 , 708 , 725 , 730 , 732 , 743 , 748 , 760 , 765 , 767 , 778 , 783 , 795 , 800 , 802 , 813 , 822 , 824 , 835 , 844 , 856 , 865 , 867 , 878 , 887 , 899 , 908 , 910
_zrefcheck_is_integer_rgx:nTF .	220	_zrefcheck_run_checks:nnn	17 , 407 , 429
\l_zrefcheck_lbl_b_int	517 , 814 , 820 , 836 , 842 , 857 , 863 , 879 , 885 , 900 , 906	_zrefcheck_target_label:n	241 , 251 , 414 , 423
\l_zrefcheck_lbl_int	517 , 525 , 531 , 532 , 545 , 551 , 552 , 595 , 601 , 602 , 613 , 619 , 620 , 631 , 637 , 638 , 650 , 656 , 657 , 669 , 679 , 684 , 687 , 698 , 704 , 707 , 724 , 730 , 731 , 742 , 748 , 749 , 759 , 765 , 766 , 777 , 783 , 784 , 794 , 800 , 801 , 812 , 822 , 823 , 834 , 844 , 845 , 855 , 865 , 866 , 877 , 887 , 888 , 898 , 908 , 909	\l_zrefcheck_target_label_bool .	231 , 237 , 245
\l_zrefcheck_link_anchor_tl . . .	371 , 400 , 401	\l_zrefcheck_target_label_tl . . .	230 , 247 , 248 , 249 , 254 , 260
\l_zrefcheck_link_label_tl	371 , 387 , 389 , 399	\l_zrefcheck_use_hyperref_bool .	111 , 138 , 147 , 394
\l_zrefcheck_link_star_tl	371 , 388 , 395	\l_zrefcheck_warn_hyperref_bool	111 , 145
_zrefcheck_message:nnnn 40 , 503 , 508		_zrefcheck_zcheck:nnnnn	12 , 14 , 17 , 370 , 377
\l_zrefcheck_msglevel_tl . . .	42 , 155	\zrefchecksetup	9 , 274
\l_zrefcheck_msgonpage_bool 188 , 499			
\l_zrefcheck_onpage_bool	440 , 455 , 479 , 482 , 488 , 492 , 500		