

# The zref-check package

## Code documentation

gusbrs

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

<https://www.ctan.org/pkg/zref-check>

Version v0.3.7 – 2024-11-28

## Contents

<b>1</b>	<b>Initial setup</b>	<b>2</b>
<b>2</b>	<b>Dependencies</b>	<b>2</b>
<b>3</b>	<b>zref setup</b>	<b>2</b>
<b>4</b>	<b>Plumbing</b>	<b>3</b>
	4.1 Auxiliary . . . . .	3
	4.2 Messages . . . . .	4
	4.3 Integer testing . . . . .	5
	4.4 Options . . . . .	6
	4.5 Position on page . . . . .	9
	4.6 Counter . . . . .	11
	4.7 Label formats . . . . .	12
	4.8 Property values . . . . .	12
<b>5</b>	<b>User interface</b>	<b>14</b>
	5.1 \zcheck . . . . .	14
	5.2 Targets . . . . .	16
<b>6</b>	<b>Checks</b>	<b>17</b>
	6.1 Single label checks . . . . .	17
	6.2 Setup . . . . .	18
	6.3 Running . . . . .	19
	6.4 Conditionals . . . . .	22
	6.4.1 This page . . . . .	22
	6.4.2 On page . . . . .	23
	6.4.3 Before / After . . . . .	24
	6.4.4 Pages . . . . .	24
	6.4.5 Close / Far . . . . .	26

6.4.6	Chapter	27
6.4.7	Section	29
6.4.8	Manual	31
<b>7</b>	<b>zref-clever integration</b>	<b>31</b>
<b>8</b>	<b>zref-vario integration</b>	<b>32</b>
	<b>Index</b>	<b>32</b>

## 1 Initial setup

Start the DocStrip guards.

```
1 \*package)
```

Identify the internal prefix (L<sup>A</sup>T<sub>E</sub>X3 DocStrip convention).

```
2 @@=zrefcheck)
```

For the `chapter` and `section` checks, `zref-check` uses the new hook system in `ltxcmds`, which was released with the 2021/06/01 L<sup>A</sup>T<sub>E</sub>X kernel. And, since we followed the move to `e`-type expansion, to play safe we require the 2023-11-01 kernel or newer.

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

Identify the package.

```
14 \ProvidesExplPackage {zref-check} {2024-11-28} {0.3.7}
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

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. Thanks Ulrike Fischer for suggestions at TeX.SX about the proper place to make the hooks for this purpose.

```
19 \newcounter { zc@abschap }
20 \newcounter { zc@abssec } [ zc@abschap ]
```

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   { \stepcounter { zc@abschap } }
23 \zref@newprop { zc@abschap } [0] { \int_use:N \c@zc@abschap }
24 \zref@addprop \ZREF@mainlist { zc@abschap }

25 \AddToHook { cmd / section / before }
26   { \stepcounter { zc@abssec } }
27 \zref@newprop { zc@abssec } [0] { \int_use:N \c@zc@abssec }
28 \zref@addprop \ZREF@mainlist { zc@abssec }

```

These are the lists 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. For user facing labels, we must use the `main` property list, so that `zref-clever` can also retrieve the properties it needs to refer to them.

```

29 \zref@newlist { zrefcheck-check }
30 \zref@addprops { zrefcheck-check }
31   {
32     page , % for messages
33     abspage ,
34     zc@abschap ,
35     zc@abssec
36   }
37 \zref@newlist { zrefcheck-end }
38 \zref@addprops { zrefcheck-end }
39   {
40     abspage ,
41     zc@abschap ,
42     zc@abssec
43   }

```

For `zref-vario` we only need page information, since we only perform above and below checks there.

```

44 \zref@newlist { zrefcheck-zrefvario }
45 \zref@addprops { zrefcheck-zrefvario }
46   {
47     page , % for messages
48     abspage ,
49   }

```

## 4 Plumbing

### 4.1 Auxiliary

```

\l__zrefcheck_tmpa_tl Temporary scratch variables.
\l__zrefcheck_tmpb_tl
\g__zrefcheck_tmpa_tl
\l__zrefcheck_tmpa_int
\l__zrefcheck_tmpa_bool
\g__zrefcheck_tmpa_ior
50 \tl_new:N \l__zrefcheck_tmpa_tl
51 \tl_new:N \l__zrefcheck_tmpb_tl
52 \tl_new:N \g__zrefcheck_tmpa_tl
53 \int_new:N \l__zrefcheck_tmpa_int
54 \bool_new:N \l__zrefcheck_tmpa_bool
55 \ior_new:N \g__zrefcheck_tmpa_ior

```

*(End of definition for `\l__zrefcheck_tmpa_tl` and others.)*

## 4.2 Messages

```

\__zrefcheck_message:nnnn
\__zrefcheck_message:nne
56 \cs_new_protected:Npn \__zrefcheck_message:nnnn #1#2#3#4
57 {
58   \use:c { msg_ \l__zrefcheck_msglevel_tl :nnnnn }
59   { zref-check } {#1} {#2} {#3} {#4}
60 }
61 \cs_generate_variant:Nn \__zrefcheck_message:nnnn { nne }

(End of definition for \__zrefcheck_message:nnnn.)

62 \msg_new:nnn { zref-check } { check-failed }
63 {
64   Check~failed~\msg_line_context:..~
65   Failed~check~'#1'~for~label~'#2'~on~page~#3.
66 }
67 \msg_new:nnn { zref-check } { double-check }
68 {
69   Same~page~check~\msg_line_context:..~
70   Double~check~'#1'~for~label~'#2'~on~page~#3.
71 }
72 \msg_new:nnn { zref-check } { empty-label }
73 {
74   Check~failed~\msg_line_context:..~
75   Failed~check~'#1'~for~empty~label.
76 }
77 \msg_new:nnn { zref-check } { no-checks }
78 { No~checks~for~'\iow_char:N\zcheck'~\msg_line_context:. }
79 \msg_new:nnn { zref-check } { check-missing }
80 { Check~'#1'~not~defined~\msg_line_context:. }
81 \msg_new:nnn { zref-check } { property-undefined }
82 { Property~'#1'~not~defined~\msg_line_context:. }
83 \msg_new:nnn { zref-check } { property-not-in-label }
84 { Label~'#1'~has~no~property~'#2'~\msg_line_context:. }
85 \msg_new:nnn { zref-check } { property-not-integer }
86 { Property~'#1'~for~label~'#2'~not~an~integer~\msg_line_context:. }
87 \msg_new:nnn { zref-check } { hyperref-preamble-only }
88 {
89   Option~'hyperref'~only~available~in~the~preamble. \iow_newline:
90   Use~the~starred~version~of~'\iow_char:N\zcheck'~instead.
91 }
92 \msg_new:nnn { zref-check } { missing-hyperref }
93 { Missing~'hyperref'~package. \iow_newline: Setting~'hyperref=false'. }
94 \msg_new:nnn { zref-check } { ignore-ok-document-only }
95 {
96   Option~'#1'~only~available~in~the~document. \iow_newline:
97   Use~option~'msglevel'~instead.
98 }
99 \msg_new:nnn { zref-check } { option-preamble-only }
100 { Option~'#1'~is~preamble~only~\msg_line_context:. }
101 \msg_new:nnn { zref-check } { closerange-not-positive-integer }
102 {
103   Option~'closerange'~not~a~positive~integer~\msg_line_context:..~

```

```

104     Using-default-value.
105   }
106   \msg_new:nnn { zref-check } { labelcmd-undefined }
107   {
108     Control-sequence-named-’#1’-used-in-option-’labelcmd’-is-not-defined.~
109     Using-default-value.
110   }
111   \msg_new:nnn { zref-check } { option-deprecated-with-alternative }
112   {
113     Option-’#1’-has-been-deprecated-\msg_line_context:\iow_newline:
114     Use-’#2’-instead.
115   }
116   \msg_new:nnn { zref-check } { option-deprecated }
117   { Option-’#1’-has-been-deprecated-\msg_line_context:. }
118   \msg_new:nnn { zref-check } { load-time-options }
119   {
120     ’zref-check’~does-not-accept-load-time-options.~
121     To-configure-package-options,~use-’\iow_char:N\zrefchecksetup’.
122   }

```

### 4.3 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.

```

123 \cs_new_eq:NN \_zrefcheck_int_to_roman:w \_int_to_roman:w
124 \prg_new_conditional:Npnn \_zrefcheck_is_integer:n #1 { p, T , F , TF }
125 {
126   \tl_if_empty:oTF {#1}
127   { \prg_return_false: }
128   {
129     \tl_if_empty:oTF { \_zrefcheck_int_to_roman:w -0#1 }
130     { \prg_return_true: }
131     { \prg_return_false: }
132   }
133 }

```

*(End of 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.

```

134 \prg_new_protected_conditional:Npnn \__zrefcheck_is_integer_rgx:n #1 { TF }
135 {
136   \regex_match:nnTF { \A\d+\Z } {#1}
137   { \prg_return_true: }
138   { \prg_return_false: }
139 }

```

(End of definition for \\_\_zrefcheck\_is\_integer\_rgx:n.)

## 4.4 Options

### hyperref option

---

```

\l__zrefcheck_use_hyperref_bool
\l__zrefcheck_warn_hyperref_bool

```

---

```

140 \bool_new:N \l__zrefcheck_use_hyperref_bool
141 \bool_new:N \l__zrefcheck_warn_hyperref_bool
142 \keys_define:nn { zref-check/setup }
143 {
144   hyperref .choice: ,
145   hyperref / auto .code:n =
146   {
147     \bool_set_true:N \l__zrefcheck_use_hyperref_bool
148     \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
149   } ,
150   hyperref / true .code:n =
151   {
152     \bool_set_true:N \l__zrefcheck_use_hyperref_bool
153     \bool_set_true:N \l__zrefcheck_warn_hyperref_bool
154   } ,
155   hyperref / false .code:n =
156   {
157     \bool_set_false:N \l__zrefcheck_use_hyperref_bool
158     \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
159   } ,
160   hyperref .initial:n = auto ,
161   hyperref .default:n = auto
162 }

163 \AddToHook { begindocument }
164 {
165   \@ifpackageloaded { hyperref }
166   {
167     \bool_if:NT \l__zrefcheck_use_hyperref_bool
168     { \RequirePackage { zref-hyperref } }
169   }
170   {
171     \bool_if:NT \l__zrefcheck_warn_hyperref_bool
172     { \msg_warning:nn { zref-check } { missing-hyperref } }
173     \bool_set_false:N \l__zrefcheck_use_hyperref_bool
174   }
175   \keys_define:nn { zref-check/setup }
176   {

```

```

177         hyperref .code:n =
178         { \msg_warning:nn { zref-check } { hyperref-preamble-only } }
179     }
180 }

```

## msglevel option

---

### \l\_\_zrefcheck\_msglevel\_tl

```

181 \tl_new:N \l__zrefcheck_msglevel_tl
182 \keys_define:nn { zref-check/setup }
183 {
184     msglevel .choice: ,
185     msglevel / warn .code:n =
186     { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } } ,
187     msglevel / info .code:n =
188     { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } } ,
189     msglevel / none .code:n =
190     { \tl_set:Nn \l__zrefcheck_msglevel_tl { none } } ,
191     msglevel / infoifdraft .code:n =
192     {
193         \ifdraft
194         { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
195         { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
196     } ,
197     msglevel / warniffinal .code:n =
198     {
199         \ifoptionfinal
200         { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
201         { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
202     } ,
203     msglevel .value_required:n = true ,
204     msglevel .initial:n = warn ,

```

ignore and ok are convenience aliases for msglevel=none, but only for use in the document body.

```

205     ignore .code:n =
206     { \msg_warning:nnn { zref-check } { ignore-ok-document-only } { ignore } } ,
207     ignore .value_forbidden:n = true ,
208     ok .code:n =
209     { \msg_warning:nnn { zref-check } { ignore-ok-document-only } { ok } } ,
210     ok .value_forbidden:n = true ,
211 }

212 \AddToHook { begindocument }
213 {
214     \keys_define:nn { zref-check/setup }
215     {
216         ignore .meta:n = { msglevel = none } ,
217         ok .meta:n = { msglevel = none } ,
218     }
219 }

```

## onpage option

### \l\_\_zrefcheck\_msgonpage\_bool

```
220 \bool_new:N \l__zrefcheck_msgonpage_bool
221 \keys_define:nn { zref-check/setup }
222 {
223   onpage .choice: ,
224   onpage / labelseq .code:n =
225     {
226       \bool_set_false:N \l__zrefcheck_msgonpage_bool
227     } ,
228   onpage / msg .code:n =
229     {
230       \bool_set_true:N \l__zrefcheck_msgonpage_bool
231     } ,
232   onpage / labelseqifdraft .code:n =
233     {
234       \ifdraft
235         { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
236         { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
237     } ,
238   onpage / msgiffinal .code:n =
239     {
240       \ifoptionfinal
241         { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
242         { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
243     } ,
244   onpage .value_required:n = true ,
245   onpage .initial:n = labelseq
246 }
```

## closerange option

### \l\_\_zrefcheck\_close\_range\_int

```
247 \int_new:N \l__zrefcheck_close_range_int
248 \keys_define:nn { zref-check/setup }
249 {
250   closerange .code:n =
251     {
252       \__zrefcheck_is_integer_rgx:nTF {#1}
253       { \int_set:Nn \l__zrefcheck_close_range_int { \int_eval:n {#1} } }
254       {
255         \msg_warning:nn { zref-check } { closerange-not-positive-integer }
256         \int_set:Nn \l__zrefcheck_close_range_int { 5 }
257       }
258     } ,
259   closerange .value_required:n = true ,
260   closerange .initial:n = 5
261 }
```



## Package options

zref-check does not accept load-time options. Despite the tradition of so doing, Joseph Wright has a point in recommending otherwise at <https://chat.stackexchange.com/transcript/message/60360822#60360822>: separating “loading the package” from “configuring the package” grants less trouble with “option clashes” and with expansion of options at load-time.

```
262 \bool_lazy_and:nnT
263   { \tl_if_exist_p:c { opt@ zref-check.sty } }
264   { ! \tl_if_empty_p:c { opt@ zref-check.sty } }
265   { \msg_warning:nn { zref-check } { load-time-options } }
```

`\zrefchecksetup` Provide `\zrefchecksetup`.

```
266 \NewDocumentCommand \zrefchecksetup { m }
267   { \keys_set:nn { zref-check/setup } {#1} }
```

*(End of definition for \zrefchecksetup.)*

## 4.5 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@filesw \immediate\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](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`

```

268 \prop_new:N \g__zrefcheck_auxfile_lblseq_prop
269 \tl_gset:Nn \g__zrefcheck_tmpa_tl { \c_sys_jobname_str .aux }
270 \file_if_exist:nT { \g__zrefcheck_tmpa_tl }
271 {

```

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.

```

272   \ior_open:Nn \g__zrefcheck_tmpa_ior { \g__zrefcheck_tmpa_tl }
273   \group_begin:
274   \int_zero:N \l__zrefcheck_tmpa_int
275   \tl_clear:N \l__zrefcheck_tmpa_tl
276   \tl_clear:N \l__zrefcheck_tmpb_tl
277   \bool_set_false:N \l__zrefcheck_tmpa_bool
278   \ior_map_variable:NNn \g__zrefcheck_tmpa_ior \l__zrefcheck_tmpa_tl
279   {

```

```

280     \tl_map_variable:NNn \l__zrefcheck_tmpa_tl \l__zrefcheck_tmpb_tl
281     {
282         \tl_if_eq:NnTF \l__zrefcheck_tmpb_tl { \zref@newlabel }
283         {

```

Found a \zref@label, signal it.

```

284         \bool_set_true:N \l__zrefcheck_tmpa_bool
285     }
286     {
287         \bool_if:NTF \l__zrefcheck_tmpa_bool
288         {
289             \bool_set_false:N \l__zrefcheck_tmpa_bool
290             \int_incr:N \l__zrefcheck_tmpa_int
291             \prop_gput:Nee \g__zrefcheck_auxfile_lblseq_prop
292             { \l__zrefcheck_tmpb_tl }
293             { \int_use:N \l__zrefcheck_tmpa_int }
294         }
295     }

```

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\_\_zrefcheck\_tmpa\_bool in the T branch.

```

296         \tl_map_break:
297     }
298 }
299 }
300 }
301 \group_end:
302 \ior_close:N \g__zrefcheck_tmpa_ior
303 }

```

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.

There's an interesting answer by David Carlisle at <https://tex.stackexchange.com/a/419189> to decide whether to typeset “above” or “below” using a method which essentially boils down to “position in the .aux file”.

## 4.6 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.

```

304 \begingroup
305   \let \@addtoreset \ltx@gobbletwo
306   \newcounter { zrefcheck }
307 \endgroup
308 \setcounter { zrefcheck } { 0 }

```

## 4.7 Label formats

```

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

```

(End of definition for `\__zrefcheck_check_lblfmt:n`)

```

\__zrefcheck_end_lblfmt:n       \__zrefcheck_end_lblfmt:n {<label>}
310 \cs_new:Npn \__zrefcheck_end_lblfmt:n #1 { #1 @zrefcheck }

```

(End of definition for `\__zrefcheck_end_lblfmt:n`)

## 4.8 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, and also make these functions ‘protected’ (see egreg’s <https://tex.stackexchange.com/a/572903>: “a function that performs assignments should be protected”). 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>}

311 \cs_new_protected:Npn \zrefcheck_get_astl:nnn #1#2#3
312 {
313   \tl_clear:N #3
314   \tl_if_eq:nnTF {#2} { lblseq }
315   {
316     \prop_get:NnNF \g__zrefcheck_auxfile_lblseq_prop {#1} #3
317     {
318       \msg_warning:nnnn { zref-check }
319       { property-not-in-label } {#1} {#2}
320     }
321   }
322   {

```

There are three things we need to check to ensure the information we are trying to retrieve here exists: the existence of  $\langle label \rangle$ , the existence of  $\langle prop \rangle$ , 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:nnnn` (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.

```

323     \zref@ifrefundefined {#1}
324     {}
325     {
326         \zref@ifpropundefined {#2}
327         { \msg_warning:nnnn { zref-check } { property-undefined } {#2} }
328         {
329             \zref@ifrefcontainsprop {#1} {#2}
330             {
331                 \exp_args:NNNo \exp_args:NNNo \tl_set:Nn #3
332                 { \zref@extractdefault {#1} {#2} { \c_empty_tl } }
333             }
334             {
335                 \msg_warning:nnnn
336                 { zref-check } { property-not-in-label } {#1} {#2}
337             }
338         }
339     }
340 }
341 }

```

(End of 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).

```

342 \bool_new:N \l__zrefcheck_integer_bool

```

---

`\l__zrefcheck_propval_tl`

```

343 \tl_new:N \l__zrefcheck_propval_tl

```

```

\zrefcheck_get_asint:nnn      \zrefcheck_get_asint:nnn {<label>} {<prop>} {<int var>}
344 \cs_new_protected:Npn \zrefcheck_get_asint:nnn #1#2#3
345 {
346   \zrefcheck_get_astl:nnn {#1} {#2} { \l__zrefcheck_propval_tl }
347   \__zrefcheck_is_integer:nTF { \l__zrefcheck_propval_tl }
348   {

```

Make it an integer data type.

```

349     \int_set:Nn #3 { \int_eval:n { \l__zrefcheck_propval_tl } }
350   }
351 {
352   \bool_set_false:N \l__zrefcheck_integer_bool
353   \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`.

```

354   { }
355   {
356     \msg_warning:nnnn { zref-check }
357     { property-not-integer } {#2} {#1}
358   }
359 }
360 }

```

(End of 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>}

361 \NewDocumentCommand \zcheck { s O { } m m }
362 { \zref@wrapper@babel \__zrefcheck_zcheck:nnnn {#3} {#1} {#2} {#4} }

```

(End of definition for `\zcheck`.)

---

```

\l__zrefcheck_zcheck_labels_seq
\g__zrefcheck_id_int
\l__zrefcheck_checkbeg_tl
\l__zrefcheck_link_label_tl
\l__zrefcheck_link_anchor_tl
\l__zrefcheck_link_star_bool

```

---

```

363 \seq_new:N \l__zrefcheck_zcheck_labels_seq
364 \int_new:N \g__zrefcheck_id_int
365 \tl_new:N \l__zrefcheck_checkbeg_tl
366 \tl_new:N \l__zrefcheck_link_label_tl
367 \tl_new:N \l__zrefcheck_link_anchor_tl
368 \bool_new:N \l__zrefcheck_link_star_bool

```

`\__zrefcheck_zcheck:nmmm` 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:nmmm {<labels>} {(*)} {<checks/options>} {<text>}

```

```

369 \cs_new_protected:Npn \__zrefcheck_zcheck:nmmm #1#2#3#4
370 {
371   \group_begin:

```

Process local options and checks. We use `\seq_set_split:Nnn` to set `\l__zrefcheck_zcheck_labels_seq` – instead of `\seq_set_from_clist:Nn` – to support empty labels.

```

372   \keys_set:nn { zref-check/zcheck } {#3}
373   \seq_set_split:Nnn \l__zrefcheck_zcheck_labels_seq { , } {#1}

```

Names of the labels for this `zcheck` call.

```

374   \int_gincr:N \g__zrefcheck_id_int
375   \tl_set:Ne \l__zrefcheck_checkbeg_tl
376   { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }

```

Set `checkbeg` label.

```

377   \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrefcheck-check }

```

Typeset `{<text>}`, 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.

```

378   \seq_get:NN \l__zrefcheck_zcheck_labels_seq \l__zrefcheck_link_label_tl
379   \bool_set:Nn \l__zrefcheck_link_star_bool {#2}
380   \zref@ifrefundefined { \l__zrefcheck_link_label_tl }

```

If the reference is undefined, just typeset.

```

381   {#4}
382   {
383     \bool_if:nTF
384     {
385       \l__zrefcheck_use_hyperref_bool &&
386       ! \l__zrefcheck_link_star_bool
387     }
388     {
389       \exp_args:Ne \zrefcheck_get_astl:nmm

```

```

390         { \l__zrefcheck_link_label_tl }
391         { anchor } { \l__zrefcheck_link_anchor_tl }
392         \hyperlink { \l__zrefcheck_link_anchor_tl } {#4}
393     }
394     {#4}
395 }

```

Set checkend label.

```

396 \bool_if:NT \l__zrefcheck_zcheck_end_label_bool
397 {
398     \zref@labelbylist
399     { \l__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
400     { zrefcheck-end }
401 }

```

Check if  $\langle labels \rangle$  are defined.

```

402 \seq_map_inline:Nn \l__zrefcheck_zcheck_labels_seq
403 { \tl_if_empty:nF {##1} { \zref@refused {##1} } }

```

Run the checks.

```

404 \__zrefcheck_run_checks:nne { \l__zrefcheck_zcheck_checks_seq }
405 { \l__zrefcheck_zcheck_labels_seq } { \l__zrefcheck_checkbeg_tl }
406 \group_end:
407 }

```

*(End of definition for \\_\_zrefcheck\_zcheck:nnnn.)*

## 5.2 Targets

```

\zctarget \zctarget{\label}{\text}

```

```

408 \NewDocumentCommand \zctarget { m +m }
409 {

```

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.

```

410 \group_begin:
411 \refstepcounter { zrefcheck }
412 \zref@wrapper@babel \zref@label {#1}
413 #2
414 \tl_if_empty:nF {#2}
415 {
416     \zref@wrapper@babel
417     \zref@labelbylist { \__zrefcheck_end_lblfmt:n {#1} } { zrefcheck-end }
418 }
419 \group_end:
420 }

```

*(End of definition for \zctarget.)*

```

zcregion \begin{zcregion}{\label}
...
\end{zcregion}

```

```

421 \NewDocumentEnvironment {zcregion} { m }
422 {

```



```

423   \refstepcounter { zrefcheck }
424   \zref@wrapper@babel \zref@label {#1}
425 }
426 {
427   \zref@wrapper@babel
428   \zref@labelbylist { \__zrefcheck_end_lblfmt:n {#1} } { zrefcheck-end }
429 }

```

(End of definition for *zcregion*.)

## 6 Checks

What is needed define a *zref-check* check?

First, a conditional function defined with:

```
\prg_new_protected_conditional:Npnn \__zrefcheck_check_⟨check⟩:nn #1#2 { F }
```

where *⟨check⟩* is the name of the check, the first argument is the *{⟨label⟩}* and the second the *{⟨reference⟩}*. 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 *⟨reference⟩*. That is, the “before” check should return true if the *⟨label⟩* 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 *⟨reference⟩* 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 the *zref-check* / *zcheck* key set.

Fourth, if the check requires only a single label to work, it should be registered in `\c__zrefcheck_single_label_checks_seq`.

### 6.1 Single label checks

Some checks do not require an “end label” in `\zcheck`, notably the sectioning ones, which don’t rely on page boundaries. Hence, in case `\zcheck` only calls checks in this set, we can spare the setting of the end label.

---

`\c__zrefcheck_single_label_checks_seq`

---

```
430 \seq_const_from_clist:Nn \c__zrefcheck_single_label_checks_seq
431 {
432   thischap ,
433   prevchap ,
434   nextchap ,
435   chapsbefore ,
436   chapsafter ,
437   thissec ,
438   prevsec ,
439   nextsec ,
440   secsbefore ,
441   secsafter ,
442   manual ,
443 }
```

## 6.2 Setup

---

`\l__zrefcheck_zcheck_checks_seq`  
`\l__zrefcheck_end_label_required_bool`

---

```
444 \seq_new:N \l__zrefcheck_zcheck_checks_seq
445 \bool_new:N \l__zrefcheck_zcheck_end_label_bool
```

First, we inherit all the main options into the keys of `zref-check` / `zcheck`. See <https://github.com/latex3/latex3/issues/1254>.

```
446 \keys_define:nn { zref-check } { zcheck .inherit:n = zref-check/setup }
```

Then we add the checks to it.

```
447 \clist_map_inline:nn
448 {
449   thispage ,
450   prevpage ,
451   nextpage ,
452   facing ,
453   otherpage ,
454   pagegap ,
455   above ,
456   below ,
457   pagesbefore ,
458   ppbefore ,
459   pagesafter ,
460   ppafter ,
461   before ,
462   after ,
463   thischap ,
464   prevchap ,
465   nextchap ,
466   chapsbefore ,
467   chapsafter ,
468   thissec ,
469   prevsec ,
```

```

470     nextsec ,
471     secsbefore ,
472     secsafter ,
473     close ,
474     far ,
475     manual ,
476 }
477 {
478   \keys_define:nm { zref-check/zcheck }
479   {
480     #1 .code:n =
481     {
482       \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq {#1}
483       \seq_if_in:NnF \c__zrefcheck_single_label_checks_seq {#1}
484       { \bool_set_true:N \l__zrefcheck_zcheck_end_label_bool }
485     } ,
486     #1 .value_forbidden:n = true ,
487   }
488 }

```

### 6.3 Running

```

\__zrefcheck_run_checks:nnn   \__zrefcheck_run_checks:nnn {<checks>} {<labels>} {<reference>}
                              <checks> are expected to be received as a sequence variable.
489 \cs_new_protected:Npn \__zrefcheck_run_checks:nnn #1#2#3
490   {
491     \group_begin:
492     \seq_map_inline:Nn #2
493     {
494       \seq_if_empty:NTF #1
495       { \__zrefcheck_message:nnnn { no-checks } { } { } { } }
496       {
497         \seq_map_inline:Nn #1
498         { \__zrefcheck_do_check:nnn {###1} {#1} {#3} }
499       }
500     }
501     \group_end:
502   }
503 \cs_generate_variant:Nn \__zrefcheck_run_checks:nnn { nne }

```

(End of definition for \\_\_zrefcheck\_run\_checks:nnn.)

---

```

\l__zrefcheck_passedcheck_bool
\l__zrefcheck_onpage_bool
\l__zrefcheck_empty_label_bool
\c__zrefcheck_onpage_checks_seq

```

```

504 \bool_new:N \l__zrefcheck_passedcheck_bool
505 \bool_new:N \l__zrefcheck_onpage_bool
506 \bool_new:N \l__zrefcheck_empty_label_bool
507 \seq_const_from_clist:Nn \c__zrefcheck_onpage_checks_seq
508   { above , below , before , after }

```

Variant not provided by expl3.

```

509 \cs_generate_variant:Nn \exp_args:Nnno { Nnoo }
\__zrefcheck_do_check:nnn
\__zrefcheck_do_check:nnn {<check>} {<label beg>} {<reference beg>}
510 \cs_new_protected:Npn \__zrefcheck_do_check:nnn #1#2#3
511 {
512   \group_begin:
513   \bool_set_true:N \l__zrefcheck_passedcheck_bool
514   \bool_set_false:N \l__zrefcheck_onpage_bool
515   \bool_set_false:N \l__zrefcheck_empty_label_bool
516   \cs_if_exist:cTF { __zrefcheck_check_ #1 :nnF }
517   {

```

<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. So there's no need to do it again here. The only exception is the case of empty labels, for which we want to issue a failed check warning.

```

518   \zref@ifrefundefined {#2}
519   {
520     \tl_if_empty:nT {#2}
521     {
522       \bool_set_false:N \l__zrefcheck_passedcheck_bool
523       \bool_set_true:N \l__zrefcheck_empty_label_bool
524     }
525   }
526   {
527     % ‘‘label beg’’ vs ‘‘reference beg’’.
528     \use:c { __zrefcheck_check_ #1 :nnF }
529     {#2} {#3}
530     { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
531     % ‘‘reference end’’ \emph{may} exist or not depending on the
532     % checks.
533     \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#3} }
534     {
535       % ‘‘label end’’ \emph{may} have been created by the
536       % target commands.
537       \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
538       {}
539       {
540         % ‘‘label end’’ vs ‘‘reference beg’’.
541         \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
542         { \__zrefcheck_end_lblfmt:n {#2} } {#3}
543         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
544       }
545     }
546   }
547   {
548     % ‘‘label beg’’ vs ‘‘reference end’’.
549     \exp_args:Nnno \use:c { __zrefcheck_check_ #1 :nnF }
550     {#2} { \__zrefcheck_end_lblfmt:n {#3} }
551     { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
552     % ‘‘label end’’ \emph{may} have been created by the
553     % target commands.
554     \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
555     {}
556     {

```

```

556 % ‘‘label end’’ vs ‘‘reference beg’’.
557 \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
558 { \__zrefcheck_end_lblfmt:n {#2} } {#3}
559 { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
560 % ‘‘label end’’ vs ‘‘reference end’’.
561 \exp_args:Nnoo \use:c { __zrefcheck_check_ #1 :nnF }
562 { \__zrefcheck_end_lblfmt:n {#2} }
563 { \__zrefcheck_end_lblfmt:n {#3} }
564 { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
565 }
566 }

```

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.

```

567 \seq_if_in:NnT \c__zrefcheck_onpage_checks_seq {#1}
568 {
569   \__zrefcheck_check_thispage:nnT
570   {#2} {#3}
571   { \bool_set_true:N \l__zrefcheck_onpage_bool }
572   \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#3} }
573   {
574     \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
575     {}
576     {
577       \__zrefcheck_check_thispage:nnT
578       { \__zrefcheck_end_lblfmt:n {#2} } {#3}
579       { \bool_set_true:N \l__zrefcheck_onpage_bool }
580     }
581   }
582   {
583     \__zrefcheck_check_thispage:nnT
584     {#2} { \__zrefcheck_end_lblfmt:n {#3} }
585     { \bool_set_true:N \l__zrefcheck_onpage_bool }
586     \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
587     {}
588     {
589       \__zrefcheck_check_thispage:nnT
590       { \__zrefcheck_end_lblfmt:n {#2} } {#3}
591       { \bool_set_true:N \l__zrefcheck_onpage_bool }
592       \__zrefcheck_check_thispage:nnT
593       { \__zrefcheck_end_lblfmt:n {#2} }
594       { \__zrefcheck_end_lblfmt:n {#3} }
595       { \bool_set_true:N \l__zrefcheck_onpage_bool }
596     }
597   }
598 }
599 }
600 }
601 { \msg_warning:nnn { zref-check } { check-missing } {#1} }
602 \bool_if:NTF \l__zrefcheck_passedcheck_bool
603 {
604   \bool_if:nT
605   {

```

```

606         \l__zrefcheck_msgonpage_bool &&
607         \l__zrefcheck_onpage_bool
608     }
609     {
610         \__zrefcheck_message:nne { double-check } {#1} {#2}
611         { \zref@extractdefault {#3} {page} {'unknown'} }
612     }
613 }
614 {
615     \bool_if:NTF \l__zrefcheck_empty_label_bool
616     { \__zrefcheck_message:nnnn { empty-label } {#1} { } { } }
617     {
618         \__zrefcheck_message:nne { check-failed } {#1} {#2}
619         { \zref@extractdefault {#3} {page} {'unknown'} }
620     }
621 }
622 \group_end:
623 }
624 \cs_generate_variant:Nn \__zrefcheck_do_check:nnn { nnV }

```

(End of definition for `\__zrefcheck_do_check:nnn`.)

## 6.4 Conditionals

---

```

\l__zrefcheck_lbl_int
\l__zrefcheck_ref_int
\l__zrefcheck_lbl_b_int
\l__zrefcheck_ref_b_int

```

More readable scratch variables for the tests.

```

625 \int_new:N \l__zrefcheck_lbl_int
626 \int_new:N \l__zrefcheck_ref_int
627 \int_new:N \l__zrefcheck_lbl_b_int
628 \int_new:N \l__zrefcheck_ref_b_int

```

### 6.4.1 This page

```

\__zrefcheck_check_thispage:nn
\__zrefcheck_check_otherpage:nn
629 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thispage:nn #1#2 { T , F , TF }
630 {
631     \group_begin:
632     \bool_set_true:N \l__zrefcheck_integer_bool
633     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
634     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
635     \bool_lazy_and:nnTF
636     { \l__zrefcheck_integer_bool }
637     {
638         \int_compare_p:nNn
639         { \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.

```

640         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
641         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }

```

```

642     }
643     { \group_insert_after:N \prg_return_true: }
644     { \group_insert_after:N \prg_return_false: }
645   \group_end:
646 }
647 \prg_new_protected_conditional:Npnn \__zrefcheck_check_otherpage:nn #1#2 { T , F , TF }
648 {
649   \__zrefcheck_check_thispage:nnTF {#1} {#2}
650   { \prg_return_false: }
651   { \prg_return_true: }
652 }

```

(End of definition for \\_\_zrefcheck\_check\_thispage:nn and \\_\_zrefcheck\_check\_otherpage:nn.)

## 6.4.2 On page

```

\__zrefcheck_check_above:nn
\__zrefcheck_check_below:nn
653 \prg_new_protected_conditional:Npnn \__zrefcheck_check_above:nn #1#2 { F , TF }
654 {
655   \group_begin:
656   \__zrefcheck_check_thispage:nnTF {#1} {#2}
657   {
658     \bool_set_true:N \l__zrefcheck_integer_bool
659     \zrefcheck_get_asint:nmn {#1} { lblseq } { \l__zrefcheck_lbl_int }
660     \zrefcheck_get_asint:nmn {#2} { lblseq } { \l__zrefcheck_ref_int }
661     \bool_lazy_and:nnTF
662       { \l__zrefcheck_integer_bool }
663       {
664         \int_compare_p:nNn
665           { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
666           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
667           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
668       }
669     { \group_insert_after:N \prg_return_true: }
670     { \group_insert_after:N \prg_return_false: }
671   }
672   { \group_insert_after:N \prg_return_false: }
673   \group_end:
674 }
675 \prg_new_protected_conditional:Npnn \__zrefcheck_check_below:nn #1#2 { F , TF }
676 {
677   \__zrefcheck_check_thispage:nnTF {#1} {#2}
678   {
679     \__zrefcheck_check_above:nnTF {#1} {#2}
680     { \prg_return_false: }
681     { \prg_return_true: }
682   }
683   { \prg_return_false: }
684 }

```

(End of definition for \\_\_zrefcheck\_check\_above:nn and \\_\_zrefcheck\_check\_below:nn.)

### 6.4.3 Before / After

```

\__zrefcheck_check_before:nn
\__zrefcheck_check_after:nn
685 \prg_new_protected_conditional:Npnm \__zrefcheck_check_before:nn #1#2 { F }
686 {
687   \__zrefcheck_check_pagesbefore:nnTF {#1} {#2}
688   { \prg_return_true: }
689   {
690     \__zrefcheck_check_above:nnTF {#1} {#2}
691     { \prg_return_true: }
692     { \prg_return_false: }
693   }
694 }
695 \prg_new_protected_conditional:Npnm \__zrefcheck_check_after:nn #1#2 { F }
696 {
697   \__zrefcheck_check_pagesafter:nnTF {#1} {#2}
698   { \prg_return_true: }
699   {
700     \__zrefcheck_check_below:nnTF {#1} {#2}
701     { \prg_return_true: }
702     { \prg_return_false: }
703   }
704 }

```

(End of definition for \\_\_zrefcheck\_check\_before:nn and \\_\_zrefcheck\_check\_after:nn.)

### 6.4.4 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_pagegap:nn
\__zrefcheck_check_facing:nn
705 \prg_new_protected_conditional:Npnm \__zrefcheck_check_nextpage:nn #1#2 { F }
706 {
707   \group_begin:
708   \bool_set_true:N \l__zrefcheck_integer_bool
709   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
710   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
711   \bool_lazy_and:nnTF
712   { \l__zrefcheck_integer_bool }
713   {
714     \int_compare_p:nNn
715     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
716     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
717     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
718   }
719   { \group_insert_after:N \prg_return_true: }
720   { \group_insert_after:N \prg_return_false: }
721   \group_end:
722 }
723 \prg_new_protected_conditional:Npnm \__zrefcheck_check_prevpage:nn #1#2 { F }
724 {
725   \group_begin:
726   \bool_set_true:N \l__zrefcheck_integer_bool
727   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
728   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
729   \bool_lazy_and:nnTF

```



```

730     { \l__zrefcheck_integer_bool }
731     {
732         \int_compare_p:nNn
733         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
734         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
735         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
736     }
737     { \group_insert_after:N \prg_return_true: }
738     { \group_insert_after:N \prg_return_false: }
739 \group_end:
740 }
741 \prg_new_protected_conditional:Npnn \__zrefcheck_check_pagesbefore:nn #1#2 { F , TF }
742 {
743     \group_begin:
744     \bool_set_true:N \l__zrefcheck_integer_bool
745     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
746     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
747     \bool_lazy_and:nnTF
748     { \l__zrefcheck_integer_bool }
749     {
750         \int_compare_p:nNn
751         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
752         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
753         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
754     }
755     { \group_insert_after:N \prg_return_true: }
756     { \group_insert_after:N \prg_return_false: }
757 \group_end:
758 }
759 \cs_new_eq:NN \__zrefcheck_check_ppbefore:nnF \__zrefcheck_check_pagesbefore:nnF
760 \prg_new_protected_conditional:Npnn \__zrefcheck_check_pagesafter:nn #1#2 { F , TF }
761 {
762     \group_begin:
763     \bool_set_true:N \l__zrefcheck_integer_bool
764     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
765     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
766     \bool_lazy_and:nnTF
767     { \l__zrefcheck_integer_bool }
768     {
769         \int_compare_p:nNn
770         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
771         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
772         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
773     }
774     { \group_insert_after:N \prg_return_true: }
775     { \group_insert_after:N \prg_return_false: }
776 \group_end:
777 }
778 \cs_new_eq:NN \__zrefcheck_check_ppafter:nnF \__zrefcheck_check_pagesafter:nnF
779 \prg_new_protected_conditional:Npnn \__zrefcheck_check_pagegap:nn #1#2 { F }
780 {
781     \group_begin:
782     \bool_set_true:N \l__zrefcheck_integer_bool
783     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }

```

```

784 \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
785 \bool_lazy_and:nnTF
786   { \l__zrefcheck_integer_bool }
787   {
788     \int_compare_p:nNn
789     { \int_abs:n { \l__zrefcheck_lbl_int - \l__zrefcheck_ref_int } } > { 1 } &&
790     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
791     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
792   }
793   { \group_insert_after:N \prg_return_true: }
794   { \group_insert_after:N \prg_return_false: }
795 \group_end:
796 }
797 \prg_new_protected_conditional:Npnn \__zrefcheck_check_facing:nn #1#2 { F }
798 {
799   \group_begin:
800   \bool_set_true:N \l__zrefcheck_integer_bool
801   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
802   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
803   \bool_lazy_and:nnTF
804     { \l__zrefcheck_integer_bool }
805     {

```

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

```

806   \legacy_if_p:n { @twoside } &&

```

Now we test “facing”.

```

807   (
808     (
809       \int_if_odd_p:n { \l__zrefcheck_ref_int } &&
810       \int_compare_p:nNn
811         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 }
812     ) ||
813     (
814       \int_if_even_p:n { \l__zrefcheck_ref_int } &&
815       \int_compare_p:nNn
816         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 }
817     )
818   ) &&
819   ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
820   ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
821 }
822 { \group_insert_after:N \prg_return_true: }
823 { \group_insert_after:N \prg_return_false: }
824 \group_end:
825 }

```

*(End of definition for \\_\_zrefcheck\_check\_nextpage:nn and others.)*

#### 6.4.5 Close / Far

```

\__zrefcheck_check_close:nn
\__zrefcheck_check_far:nn
826 \prg_new_protected_conditional:Npnn \__zrefcheck_check_close:nn #1#2 { F , TF }
827 {
828   \group_begin:

```

```

829 \bool_set_true:N \l__zrefcheck_integer_bool
830 \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
831 \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
832 \bool_lazy_and:nnTF
833   { \l__zrefcheck_integer_bool }
834   {
835     \int_compare_p:nNn
836       { \int_abs:n { \l__zrefcheck_lbl_int - \l__zrefcheck_ref_int } }
837       <
838       { \l__zrefcheck_close_range_int + 1 } &&
839       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
840       ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
841   }
842   { \group_insert_after:N \prg_return_true: }
843   { \group_insert_after:N \prg_return_false: }
844 \group_end:
845 }
846 \prg_new_protected_conditional:Npnn \__zrefcheck_check_far:nn #1#2 { F }
847 {
848   \__zrefcheck_check_close:nnTF {#1} {#2}
849   { \prg_return_false: }
850   { \prg_return_true: }
851 }

```

(End of definition for `\__zrefcheck_check_close:nn` and `\__zrefcheck_check_far:nn`.)

#### 6.4.6 Chapter

```

\__zrefcheck_check_thischap:nn
\__zrefcheck_check_nextchap:nn
\__zrefcheck_check_prevchap:nn
\__zrefcheck_check_chapsafter:nn
\__zrefcheck_check_chapsbefore:nn
852 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thischap:nn #1#2 { F }
853 {
854   \group_begin:
855   \bool_set_true:N \l__zrefcheck_integer_bool
856   \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
857   \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
858   \bool_lazy_and:nnTF
859     { \l__zrefcheck_integer_bool }
860     {
861       \int_compare_p:nNn
862         { \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.

```

863     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
864     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
865   }
866   { \group_insert_after:N \prg_return_true: }
867   { \group_insert_after:N \prg_return_false: }
868 \group_end:
869 }
870 \prg_new_protected_conditional:Npnn \__zrefcheck_check_nextchap:nn #1#2 { F }

```

```

871 {
872   \group_begin:
873   \bool_set_true:N \l__zrefcheck_integer_bool
874   \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
875   \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
876   \bool_lazy_and:nnTF
877   { \l__zrefcheck_integer_bool }
878   {
879     \int_compare_p:nNn
880     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
881     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
882   }
883   { \group_insert_after:N \prg_return_true: }
884   { \group_insert_after:N \prg_return_false: }
885   \group_end:
886 }
887 \prg_new_protected_conditional:Npnn \__zrefcheck_check_prevchap:nn #1#2 { F }
888 {
889   \group_begin:
890   \bool_set_true:N \l__zrefcheck_integer_bool
891   \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
892   \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
893   \bool_lazy_and:nnTF
894   { \l__zrefcheck_integer_bool }
895   {
896     \int_compare_p:nNn
897     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
898     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
899     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
900   }
901   { \group_insert_after:N \prg_return_true: }
902   { \group_insert_after:N \prg_return_false: }
903   \group_end:
904 }
905 \prg_new_protected_conditional:Npnn \__zrefcheck_check_chapsafter:nn #1#2 { F }
906 {
907   \group_begin:
908   \bool_set_true:N \l__zrefcheck_integer_bool
909   \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
910   \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
911   \bool_lazy_and:nnTF
912   { \l__zrefcheck_integer_bool }
913   {
914     \int_compare_p:nNn
915     { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
916     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
917   }
918   { \group_insert_after:N \prg_return_true: }
919   { \group_insert_after:N \prg_return_false: }
920   \group_end:
921 }
922 \prg_new_protected_conditional:Npnn \__zrefcheck_check_chapsbefore:nn #1#2 { F }
923 {
924   \group_begin:

```

```

925 \bool_set_true:N \l__zrefcheck_integer_bool
926 \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
927 \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
928 \bool_lazy_and:nnTF
929   { \l__zrefcheck_integer_bool }
930   {
931     \int_compare_p:nNn
932       { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
933       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
934       ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
935   }
936   { \group_insert_after:N \prg_return_true: }
937   { \group_insert_after:N \prg_return_false: }
938 \group_end:
939 }

```

(End of definition for `\__zrefcheck_check_thischap:nn` and others.)

#### 6.4.7 Section

```

\__zrefcheck_check_thissec:nn
\__zrefcheck_check_nextsec:nn
\__zrefcheck_check_prevsec:nn
\__zrefcheck_check_secsafter:nn
\__zrefcheck_check_secsbefore:nn
940 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thissec:nn #1#2 { F }
941 {
942   \group_begin:
943   \bool_set_true:N \l__zrefcheck_integer_bool
944   \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
945   \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
946   \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
947   \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
948   \bool_lazy_and:nnTF
949     { \l__zrefcheck_integer_bool }
950     {
951       \int_compare_p:nNn
952         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
953       \int_compare_p:nNn
954         { \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.

```

955       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
956       ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
957     }
958     { \group_insert_after:N \prg_return_true: }
959     { \group_insert_after:N \prg_return_false: }
960   \group_end:
961 }
962 \prg_new_protected_conditional:Npnn \__zrefcheck_check_nextsec:nn #1#2 { F }
963 {
964   \group_begin:
965   \bool_set_true:N \l__zrefcheck_integer_bool
966   \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }

```

```

967 \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
968 \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
969 \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
970 \bool_lazy_and:nnTF
971   { \l__zrefcheck_integer_bool }
972   {
973     \int_compare_p:nNn
974     { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
975     \int_compare_p:nNn
976     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
977     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
978   }
979   { \group_insert_after:N \prg_return_true: }
980   { \group_insert_after:N \prg_return_false: }
981 \group_end:
982 }
983 \prg_new_protected_conditional:Npnn \__zrefcheck_check_prevsec:nn #1#2 { F }
984 {
985   \group_begin:
986   \bool_set_true:N \l__zrefcheck_integer_bool
987   \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
988   \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
989   \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
990   \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
991   \bool_lazy_and:nnTF
992     { \l__zrefcheck_integer_bool }
993     {
994       \int_compare_p:nNn
995       { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
996       \int_compare_p:nNn
997       { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
998       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
999       ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
1000     }
1001     { \group_insert_after:N \prg_return_true: }
1002     { \group_insert_after:N \prg_return_false: }
1003   \group_end:
1004 }
1005 \prg_new_protected_conditional:Npnn \__zrefcheck_check_secsafter:nn #1#2 { F }
1006 {
1007   \group_begin:
1008   \bool_set_true:N \l__zrefcheck_integer_bool
1009   \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
1010   \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
1011   \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
1012   \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
1013   \bool_lazy_and:nnTF
1014     { \l__zrefcheck_integer_bool }
1015     {
1016       \int_compare_p:nNn
1017       { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
1018       \int_compare_p:nNn
1019       { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
1020       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }

```

```

1021     }
1022     { \group_insert_after:N \prg_return_true: }
1023     { \group_insert_after:N \prg_return_false: }
1024   \group_end:
1025 }
1026 \prg_new_protected_conditional:Npnn \__zrefcheck_check_secsbefore:nn #1#2 { F }
1027 {
1028   \group_begin:
1029   \bool_set_true:N \l__zrefcheck_integer_bool
1030   \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
1031   \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
1032   \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
1033   \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
1034   \bool_lazy_and:nnTF
1035     { \l__zrefcheck_integer_bool }
1036     {
1037       \int_compare_p:nNn
1038         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
1039       \int_compare_p:nNn
1040         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
1041         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
1042         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
1043     }
1044     { \group_insert_after:N \prg_return_true: }
1045     { \group_insert_after:N \prg_return_false: }
1046   \group_end:
1047 }

```

(End of definition for `\__zrefcheck_check_thissec:nn` and others.)

### 6.4.8 Manual

`\__zrefcheck_check_manual:nn`

```

1048 \prg_new_protected_conditional:Npnn \__zrefcheck_check_manual:nn #1#2 { F }
1049 { \prg_return_false: }

```

(End of definition for `\__zrefcheck_check_manual:nn`.)

## 7 zref-clever integration

There are four tasks `zref-clever` needs to do, in order to offer integration with `zref-check` from the options of `\zcref`: i) set the “beg label”; ii) set the checks options; iii) run the checks; iv) (possibly) set the “end label”. Since ‘ii)’ can be done directly by running `\keys_set:nn { zref-check/zcheck }` on the options received, we provide convenience functions for the other three tasks.

`\zrefcheck_zcref_beg_label:`

```

\zrefcheck_zcref_end_label_maybe: 1050 \cs_new_protected:Npn \zrefcheck_zcref_beg_label:
\zrefcheck_zcref_run_checks_on_labels:n 1051 {
1052   \int_gincr:N \g__zrefcheck_id_int
1053   \tl_set:Ne \l__zrefcheck_checkbeg_tl
1054     { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
1055   \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrefcheck-check }

```

```

1056 }
1057 \cs_new_protected:Npn \zrefcheck_zcref_end_label_maybe:
1058 {
1059   \bool_if:NT \l__zrefcheck_zcheck_end_label_bool
1060   {
1061     \zref@labelbylist
1062     { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
1063     { zrefcheck-end }
1064   }
1065 }
1066 \cs_new_protected:Npn \zrefcheck_zcref_run_checks_on_labels:n #1
1067 {
1068   \__zrefcheck_run_checks:nne
1069   { \l__zrefcheck_zcheck_checks_seq } {#1} { \l__zrefcheck_checkbeg_tl }
1070 }

```

(End of definition for `\zrefcheck_zcref_beg_label:`, `\zrefcheck_zcref_end_label_maybe:`, and `\zrefcheck_zcref_run_checks_on_labels:n`. These functions are documented on page ??.)

## 8 zref-vario integration

```

\zrefcheck_zrefvario_label:
\zrefcheck_zrefvario_run_check_on_label:n
1071 \cs_new_protected:Npn \zrefcheck_zrefvario_label:
1072 {
1073   \int_gincr:N \g__zrefcheck_id_int
1074   \tl_set:Ne \l__zrefcheck_checkbeg_tl
1075   { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
1076   \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrefcheck-zrefvario }
1077 }
1078 \cs_new_protected:Npn \zrefcheck_zrefvario_run_check_on_label:nn #1#2
1079 { \__zrefcheck_do_check:nnV {#1} {#2} \l__zrefcheck_checkbeg_tl }
1080 \cs_generate_variant:Nn \zrefcheck_zrefvario_run_check_on_label:nn { Vn }

```

(End of definition for `\zrefcheck_zrefvario_label:` and `\zrefcheck_zrefvario_run_check_on_label:n`. These functions are documented on page ??.)

1081 `\</package>`

## Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols	bool commands:
<code>\</code> ..... <a href="#">78</a> , <a href="#">90</a> , <a href="#">121</a>	<code>\bool_if:NTF</code> ..... ... <a href="#">167</a> , <a href="#">171</a> , <a href="#">287</a> , <a href="#">396</a> , <a href="#">602</a> , <a href="#">615</a> , <a href="#">1059</a>
A	<code>\bool_if:nTF</code> ..... <a href="#">383</a> , <a href="#">604</a>
<code>\A</code> ..... <a href="#">136</a>	<code>\bool_lazy_and:nnTF</code> ..... ... <a href="#">13</a> , <a href="#">262</a> , <a href="#">635</a> , <a href="#">661</a> , <a href="#">711</a> , <a href="#">729</a> , <a href="#">747</a> , <a href="#">766</a> , <a href="#">785</a> , <a href="#">803</a> , <a href="#">832</a> , <a href="#">858</a> , <a href="#">876</a> , <a href="#">893</a> , <a href="#">911</a> , <a href="#">928</a> , <a href="#">948</a> , <a href="#">970</a> , <a href="#">991</a> , <a href="#">1013</a> , <a href="#">1034</a>
<code>\AddToHook</code> ..... <a href="#">21</a> , <a href="#">25</a> , <a href="#">163</a> , <a href="#">212</a>	<code>\bool_new:N</code> ..... <a href="#">54</a> , <a href="#">140</a> ,
B	
<code>\begingroup</code> ..... <a href="#">304</a>	



	141, 220, 342, 368, 445, 504, 505, 506		776, 795, 824, 844, 868, 885, 903, 920, 938, 960, 981, 1003, 1024, 1046
<code>\bool_set:Nn</code>	.....	379	
<code>\bool_set_false:N</code>	..	148, 157, 158, 173, 226, 235, 242, 277, 289, 352, 514, 515, 522, 530, 543, 550, 559, 564	
<code>\bool_set_true:N</code>	.....		
	.....	147, 152, 153, 230, 236, 241, 284, 484, 513, 523, 571, 579, 585, 591, 595, 632, 658, 708, 726, 744, 763, 782, 800, 829, 855, 873, 890, 908, 925, 943, 965, 986, 1008, 1029	
<b>C</b>			
<code>\catcode</code>	.....	11	
<code>\chapter</code>	.....	3, 27	
clist commands:			
<code>\clist_map_inline:nn</code>	.....	447	
cs commands:			
<code>\cs_generate_variant:Nn</code>	.....		
	.....	61, 503, 509, 624, 1080	
<code>\cs_if_exist:NTF</code>	.....	516	
<code>\cs_new:Npn</code>	.....	309, 310	
<code>\cs_new_eq:NN</code>	.....	123, 759, 778	
<code>\cs_new_protected:Npn</code>	.....		
	.....	56, 311, 344, 369, 489, 510, 1050, 1057, 1066, 1071, 1078	
<b>D</b>			
<code>\d</code>	.....	136	
<code>\def</code>	.....	3	
<b>E</b>			
<code>\emph</code>	.....	531, 535, 551	
<code>\endgroup</code>	.....	307	
exp commands:			
<code>\exp_args:Ne</code>	.....	389	
<code>\exp_args:NNno</code>	.....	331	
<code>\exp_args:Nno</code>	.....	509, 548	
<code>\exp_args:NNo</code>	.....	331	
<code>\exp_args:Nno</code>	.....	541, 557	
<code>\exp_args:Nnoo</code>	.....	561	
<b>F</b>			
file commands:			
<code>\file_if_exist:nTF</code>	.....	270	
<code>\fmtversion</code>	.....	5	
<b>G</b>			
group commands:			
<code>\group_begin:</code>	.....	273, 371, 410, 491, 512, 631, 655, 707, 725, 743, 762, 781, 799, 828, 854, 872, 889, 907, 924, 942, 964, 985, 1007, 1028	
<code>\group_end:</code>	.....	301, 406, 419, 501, 622, 645, 673, 721, 739, 757,	
			776, 795, 824, 844, 868, 885, 903, 920, 938, 960, 981, 1003, 1024, 1046
<code>\group_insert_after:N</code>	.....		
	.....	643, 644, 669, 670, 672, 719, 720, 737, 738, 755, 756, 774, 775, 793, 794, 822, 823, 842, 843, 866, 867, 883, 884, 901, 902, 918, 919, 936, 937, 958, 959, 979, 980, 1001, 1002, 1022, 1023, 1044, 1045	
<b>H</b>			
<code>\hyperlink</code>	.....	14, 392	
<b>I</b>			
<code>\ifdraft</code>	.....	193, 234	
<code>\IfFormatAtLeastTF</code>	.....	5, 6	
<code>\ifoptionfinal</code>	.....	199, 240	
int commands:			
<code>\int_abs:n</code>	.....	789, 836	
<code>\int_compare_p:nNn</code>	.....		
	.....	638, 640, 641, 664, 666, 667, 714, 716, 717, 732, 734, 735, 750, 752, 753, 769, 771, 772, 788, 790, 791, 810, 815, 819, 820, 835, 839, 840, 861, 863, 864, 879, 881, 896, 898, 899, 914, 916, 931, 933, 934, 951, 953, 955, 956, 973, 975, 977, 994, 996, 998, 999, 1016, 1018, 1020, 1037, 1039, 1041, 1042	
<code>\int_eval:n</code>	.....	253, 349	
<code>\int_gincr:N</code>	.....	374, 1052, 1073	
<code>\int_if_even_p:n</code>	.....	814	
<code>\int_if_odd_p:n</code>	.....	809	
<code>\int_incr:N</code>	.....	290	
<code>\int_new:N</code>	.....		
	.....	53, 247, 364, 625, 626, 627, 628	
<code>\int_set:Nn</code>	.....	253, 256, 349	
<code>\int_use:N</code>	.....	23, 27, 293, 309	
<code>\int_zero:N</code>	.....	274	
int internal commands:			
<code>\__int_to_roman:w</code>	.....	5, 123	
ior commands:			
<code>\ior_close:N</code>	.....	302	
<code>\ior_map_variable:NNn</code>	.....	278	
<code>\ior_new:N</code>	.....	55	
<code>\ior_open:Nn</code>	.....	272	
iow commands:			
<code>\iow_char:N</code>	.....	78, 90, 121	
<code>\iow_newline:</code>	.....	89, 93, 96, 113	
<b>K</b>			
keys commands:			
<code>\keys_define:mn</code>	.....		
	.....	142, 175, 182, 214, 221, 248, 446, 478	
<code>\keys_set:mn</code>	.....	267, 372	

	<b>L</b>		<code>\romannumeral</code> . . . . .	5
<code>\label</code> . . . . .				11
legacy commands:			<b>S</b>	
<code>\legacy_if_p:n</code> . . . . .			<code>\section</code> . . . . .	29
<code>\let</code> . . . . .			seq commands:	
	<b>M</b>		<code>\seq_const_from_clist:Nn</code> . . . . .	430, 507
msg commands:			<code>\seq_get:NN</code> . . . . .	378
<code>\msg_line_context:</code> . . . . .			<code>\seq_if_empty:NTF</code> . . . . .	494
74, 78, 80, 82, 84, 86, 100, 103, 113, 117			<code>\seq_if_in:NnTF</code> . . . . .	483, 567
<code>\msg_new:nnn</code> . . . . .			<code>\seq_map_inline:Nn</code> . . . . .	402, 492, 497
. . . . . 62, 67, 72, 77, 79, 81, 83, 85,			<code>\seq_new:N</code> . . . . .	363, 444
87, 92, 94, 99, 101, 106, 111, 116, 118			<code>\seq_put_right:Nn</code> . . . . .	482
<code>\msg_warning:nn</code> . . . . .			<code>\seq_set_from_clist:Nn</code> . . . . .	15
172, 178, 255, 265			<code>\seq_set_split:Nnn</code> . . . . .	15, 373
<code>\msg_warning:nnn</code> . . . . .			<code>\setcounter</code> . . . . .	308
206, 209, 601			<code>\space</code> . . . . .	11
<code>\msg_warning:nnnn</code> . . . . .			<code>\stepcounter</code> . . . . .	22, 26
318, 327, 335, 356			sys commands:	
	<b>N</b>		<code>\c_sys_jobname_str</code> . . . . .	269
<code>\NeedsTeXFormat</code> . . . . .				
<code>\newcounter</code> . . . . .			<b>T</b>	
<code>\NewDocumentCommand</code> . . . . .			TeX and L <sup>A</sup> T <sub>E</sub> X 2 <sub>ε</sub> commands:	
<code>\NewDocumentEnvironment</code> . . . . .			<code>\@addtoreset</code> . . . . .	305
			<code>\@currentlabel</code> . . . . .	16
	<b>P</b>		<code>\@ifl@t@r</code> . . . . .	5
<code>\PackageError</code> . . . . .			<code>\@ifpackageloaded</code> . . . . .	165
<code>\pageref</code> . . . . .			<code>\@newl@bel</code> . . . . .	9
prg commands:			<code>\c@z@abschap</code> . . . . .	23
<code>\prg_new_conditional:Npnn</code> . . . . .			<code>\c@z@abssec</code> . . . . .	27
<code>\prg_new_protected_conditional:Npnn</code>			<code>\ltx@gobbletwo</code> . . . . .	305
. . . . . 17, 134, 629, 647, 653, 675,			<code>\zref@addprop</code> . . . . .	17, 24, 28
685, 695, 705, 723, 741, 760, 779,			<code>\zref@addprops</code> . . . . .	30, 38, 45
797, 826, 846, 852, 870, 887, 905,			<code>\zref@extractdefault</code> . . . . .	12, 332, 611, 619
922, 940, 962, 983, 1005, 1026, 1048			<code>\zref@ifpropundefined</code> . . . . .	326
<code>\prg_return_false:</code> . . . . .			<code>\zref@ifrefcontainsprop</code> . . . . .	329
17, 127, 131,			<code>\zref@ifrefundefined</code> . . . . .	13, 323, 353,
138, 644, 650, 670, 672, 680, 683,			380, 518, 533, 537, 553, 572, 574, 586	
692, 702, 720, 738, 756, 775, 794,			<code>\ZREF@label</code> . . . . .	11
823, 843, 849, 867, 884, 902, 919,			<code>\zref@label</code> . . . . .	11, 412, 424
937, 959, 980, 1002, 1023, 1045, 1049			<code>\zref@labelbylist</code> . . . . .	
<code>\prg_return_true:</code> . . . . .			. . . . . 377, 398, 417, 428, 1055, 1061, 1076	
17,			<code>\ZREF@mainlist</code> . . . . .	24, 28
130, 137, 643, 651, 669, 681, 688,			<code>\zref@newlabel</code> . . . . .	9–11, 282
691, 698, 701, 719, 737, 755, 774,			<code>\zref@newlist</code> . . . . .	29, 37, 44
793, 822, 842, 850, 866, 883, 901,			<code>\zref@newprop</code> . . . . .	17, 23, 27
918, 936, 958, 979, 1001, 1022, 1044			<code>\zref@refused</code> . . . . .	13, 403
prop commands:			<code>\zref@require@unique</code> . . . . .	12
<code>\prop_get:NnNTF</code> . . . . .			<code>\zref@wrapper@babel</code> . . . . .	
<code>\prop_gput:Nnn</code> . . . . .			. . . . . 10, 15, 362, 412, 416, 424, 427	
<code>\prop_new:N</code> . . . . .			<code>\zrefcheck@required@kernel</code> . . . . .	3, 4, 6, 11
<code>\providecommand</code> . . . . .			tl commands:	
<code>\ProvidesExplPackage</code> . . . . .			<code>\c_empty_tl</code> . . . . .	12, 332
	<b>R</b>		<code>\tl_clear:N</code> . . . . .	12, 275, 276, 313
<code>\refstepcounter</code> . . . . .			<code>\tl_gset:Nn</code> . . . . .	269
regex commands:			<code>\tl_if_blank:nTF</code> . . . . .	5
<code>\regex_match:nnTF</code> . . . . .				
<code>\RequirePackage</code> . . . . .				

<code>\tl_if_empty:nTF</code> .....	5, 126, 129, 403, 414, 520	<code>\__zrefcheck_check_above:nnTF</code> ...	679, 690
<code>\tl_if_empty_p:N</code> .....	264	<code>\__zrefcheck_check_after:nn</code>	685, 695
<code>\tl_if_eq:NnTF</code> .....	282	<code>\__zrefcheck_check_before:nn</code>	685, 685
<code>\tl_if_eq:nnTF</code> .....	314	<code>\__zrefcheck_check_below:nn</code>	653, 675
<code>\tl_if_exist_p:N</code> .....	263	<code>\__zrefcheck_check_below:nnTF</code> ..	700
<code>\tl_map_break:</code> .....	296	<code>\__zrefcheck_check_chapsafter:nn</code>	
<code>\tl_map_variable:NNn</code> .....	280	.....	852, 905
<code>\tl_new:N</code> .....		<code>\__zrefcheck_check_chapsbefore:nn</code>	
... 50, 51, 52, 181, 343, 365, 366, 367		.....	852, 922
<code>\tl_set:Nn</code> .....	186, 188, 190,	<code>\__zrefcheck_check_close:nn</code>	826, 826
194, 195, 200, 201, 331, 375, 1053, 1074		<code>\__zrefcheck_check_close:nnTF</code> ..	848
		<code>\__zrefcheck_check_facing:nn</code>	705, 797
		<code>\__zrefcheck_check_far:nn</code> ..	826, 846
		<code>\__zrefcheck_check_lblfmt:n</code> .....	
		.....	12, 309, 309, 376, 1054, 1075
		<code>\__zrefcheck_check_manual:nn</code> ..	
		.....	1048, 1048
		<code>\__zrefcheck_check_nextchap:nn</code> ..	
		.....	852, 870
		<code>\__zrefcheck_check_nextpage:nn</code> ..	
		.....	705, 705
		<code>\__zrefcheck_check_nextsec:nn</code> ..	
		.....	940, 962
		<code>\__zrefcheck_check_otherpage:nn</code> ..	
		.....	629, 647
		<code>\__zrefcheck_check_pagegap:nn</code> ..	
		.....	705, 779
		<code>\__zrefcheck_check_pagesafter:nn</code>	
		.....	705, 760
		<code>\__zrefcheck_check_pagesafter:nnTF</code>	
		.....	697, 778
		<code>\__zrefcheck_check_pagesbefore:nn</code>	
		.....	705, 741
		<code>\__zrefcheck_check_pagesbefore:nnTF</code>	
		.....	687, 759
		<code>\__zrefcheck_check_ppafter:nn</code> ..	705
		<code>\__zrefcheck_check_ppafter:nnTF</code>	778
		<code>\__zrefcheck_check_ppbefore:nn</code> ..	705
		<code>\__zrefcheck_check_ppbefore:nnTF</code>	759
		<code>\__zrefcheck_check_prevchap:nn</code> ..	
		.....	852, 887
		<code>\__zrefcheck_check_prevpage:nn</code> ..	
		.....	705, 723
		<code>\__zrefcheck_check_prevsec:nn</code> ..	
		.....	940, 983
		<code>\__zrefcheck_check_secsafter:nn</code> ..	
		.....	940, 1005
		<code>\__zrefcheck_check_secsbefore:nn</code>	
		.....	940, 1026
		<code>\__zrefcheck_check_thischap:nn</code> ..	
		.....	852, 852
		<code>\__zrefcheck_check_thispage:nn</code> ..	
		.....	629, 629

**U**

use commands:

`\use:N` .... 58, 528, 541, 548, 557, 561

**Z**

<code>\Z</code> .....	136
<code>\zcheck</code> .....	14, 15, 17, 361
<code>\zcref</code> .....	31
<code>zcregion</code> .....	421
<code>\zctarget</code> .....	16, 408
<code>\zref</code> .....	15

zrefcheck commands:

<code>\zrefcheck_get_asint:nnn</code> .....	
.....	13, 14, 17, 344, 344,
	633, 634, 659, 660, 709, 710, 727,
	728, 745, 746, 764, 765, 783, 784,
	801, 802, 830, 831, 856, 857, 874,
	875, 891, 892, 909, 910, 926, 927,
	944, 945, 946, 947, 966, 967, 968,
	969, 987, 988, 989, 990, 1009, 1010,
	1011, 1012, 1030, 1031, 1032, 1033
<code>\zrefcheck_get_astl:nnn</code> .....	
.....	12, 13, 17, 311, 311, 346, 389
<code>\zrefcheck_zcref_beg_label:</code> .....	
.....	1050, 1050
<code>\zrefcheck_zcref_end_label_-</code>	
maybe: .....	1050, 1057
<code>\zrefcheck_zcref_run_checks_on_-</code>	
labels:n .....	1050, 1066
<code>\zrefcheck_zrefvario_label:</code> .....	
.....	1071, 1071
<code>\zrefcheck_zrefvario_run_check_-</code>	
on_label:n .....	1071
<code>\zrefcheck_zrefvario_run_check_-</code>	
on_label:nn .....	1078, 1080

zrefcheck internal commands:

<code>\g_zrefcheck_auxfile_lblseq_-</code>	
prop .....	10, 12, 17, 268, 291, 316
<code>\__zrefcheck_check_(check):nn</code> ..	17
<code>\__zrefcheck_check_above:nn</code>	653, 653

<code>\__zrefcheck_check_thispage:nnTF</code>	<code>\l__zrefcheck_link_star_bool</code> . . .
569, 577, 583, 589, 592, 649, 656, 677	..... 15, 368, 379, 386
<code>\__zrefcheck_check_thissec:nn</code> . . .	<code>\__zrefcheck_message:nnnn</code> . . . . .
..... 940, 940	..... 56, 56, 61, 495, 610, 616, 618
<code>\l__zrefcheck_checkbeg_tl</code> . . . . .	<code>\l__zrefcheck_msglevel_tl</code> . . 7, 58,
. 15, 365, 375, 377, 399, 405, 1053,	181, 186, 188, 190, 194, 195, 200, 201
1055, 1062, 1069, 1074, 1076, 1079	<code>\l__zrefcheck_msgonpage_bool</code> . 8,
<code>\l__zrefcheck_close_range_int</code> . . .	220, 226, 230, 235, 236, 241, 242, 606
..... 8, 247, 253, 256, 838	<code>\l__zrefcheck_onpage_bool</code> . . . 19,
<code>\__zrefcheck_do_check:nnn</code> . . . . .	505, 514, 571, 579, 585, 591, 595, 607
..... 20, 498, 510, 510, 624, 1079	<code>\c__zrefcheck_onpage_checks_seq</code> .
<code>\l__zrefcheck_empty_label_bool</code> . .	..... 19, 507, 567
..... 19, 506, 515, 523, 615	<code>\l__zrefcheck_passedcheck_bool</code> . .
<code>\l__zrefcheck_end_label_required_</code>	..... 19, 504,
<code>bool</code> . . . . . 18	513, 522, 530, 543, 550, 559, 564, 602
<code>\__zrefcheck_end_lblfmt:n</code> . . . 12,	<code>\l__zrefcheck_propval_tl</code> . . . . .
310, 310, 399, 417, 428, 533, 537,	..... 13, 343, 346, 347, 349
542, 549, 553, 558, 562, 563, 572,	<code>\l__zrefcheck_ref_b_int</code> . . . . .
574, 578, 584, 586, 590, 593, 594, 1062	..... 22, 628, 947, 952, 969,
<code>\__zrefcheck_get_asint:nnn</code> . . . . . 12	974, 990, 995, 1012, 1017, 1033, 1038
<code>\__zrefcheck_get_astl:nnn</code> . . . . . 12	<code>\l__zrefcheck_ref_int</code> . . 22, 626,
<code>\g__zrefcheck_id_int</code> . . . . . 15,	634, 639, 641, 660, 665, 667, 710,
364, 374, 376, 1052, 1054, 1073, 1075	715, 717, 728, 733, 735, 746, 751,
<code>\__zrefcheck_int_to_roman:w</code> . . . .	753, 765, 770, 772, 784, 789, 791,
..... 123, 123, 129	802, 809, 811, 814, 816, 820, 831,
<code>\l__zrefcheck_integer_bool</code> . . . . .	836, 840, 857, 862, 864, 875, 880,
..... 13, 342, 352, 632, 636, 658,	892, 897, 899, 910, 915, 927, 932,
662, 708, 712, 726, 730, 744, 748,	934, 945, 954, 956, 967, 976, 988,
763, 767, 782, 786, 800, 804, 829,	997, 999, 1010, 1019, 1031, 1040, 1042
833, 855, 859, 873, 877, 890, 894,	<code>\__zrefcheck_run_checks:nnn</code> . . . .
908, 912, 925, 929, 943, 949, 965,	..... 19, 404, 489, 489, 503, 1068
971, 986, 992, 1008, 1014, 1029, 1035	<code>\c__zrefcheck_single_label_</code>
<code>\__zrefcheck_is_integer:n</code> 5, 123, 124	<code>checks_seq</code> . . . . . 17, 18, 430, 483
<code>\__zrefcheck_is_integer:nTF</code> . . . 347	<code>\l__zrefcheck_tmpa_bool</code> . . . . .
<code>\__zrefcheck_is_integer_rgx:n</code> . . .	..... 11, 50, 277, 284, 287, 289
..... 5, 134, 134	<code>\l__zrefcheck_tmpa_int</code> . . . . .
<code>\__zrefcheck_is_integer_rgx:nTF</code> 252	..... 50, 274, 290, 293
<code>\l__zrefcheck_lbl_b_int</code> . . . . .	<code>\g__zrefcheck_tmpa_ior</code> . . . . .
..... 22, 627, 946, 952, 968,	..... 50, 272, 278, 302
974, 989, 995, 1011, 1017, 1032, 1038	<code>\g__zrefcheck_tmpa_tl</code> 50, 269, 270, 272
<code>\l__zrefcheck_lbl_int</code> . . . . .	<code>\l__zrefcheck_tmpa_tl</code> 50, 275, 278, 280
..... 22, 625, 633, 639,	<code>\l__zrefcheck_tmpb_tl</code> . . . . .
640, 659, 665, 666, 709, 715, 716,	..... 50, 276, 280, 282, 292
727, 733, 734, 745, 751, 752, 764,	<code>\l__zrefcheck_use_hyperref_bool</code> .
770, 771, 783, 789, 790, 801, 811,	. . 6, 140, 147, 152, 157, 167, 173, 385
816, 819, 830, 836, 839, 856, 862,	<code>\l__zrefcheck_warn_hyperref_bool</code>
863, 874, 880, 881, 891, 897, 898,	..... 6, 141, 148, 153, 158, 171
909, 915, 916, 926, 932, 933, 944,	<code>\__zrefcheck_zcheck:nnnn</code> . . . . .
954, 955, 966, 976, 977, 987, 997,	..... 15, 362, 369, 369
998, 1009, 1019, 1020, 1030, 1040, 1041	<code>\__zrefcheck_zcheck:nnnnn</code> 13, 14, 20
<code>\l__zrefcheck_link_anchor_tl</code> . . .	<code>\l__zrefcheck_zcheck_checks_seq</code> .
..... 15, 367, 391, 392	..... 18, 404, 444, 482, 1069
<code>\l__zrefcheck_link_label_tl</code> . . . .	<code>\l__zrefcheck_zcheck_end_label_</code>
..... 15, 366, 378, 380, 390	<code>bool</code> . . . . . 396, 445, 484, 1059

`\l_zrefcheck_zcheck_labels_seq` . `\zrefchecksetup` ..... [9](#), [266](#)  
..... [15](#), [363](#), [373](#), [378](#), [402](#), [405](#)