

# The `zref-check` package implementation\*

Gustavo Barros<sup>†</sup>

2022-02-11

## Contents

<b>1</b>	<b>Initial setup</b>	<b>2</b>
<b>2</b>	<b>Dependencies</b>	<b>2</b>
<b>3</b>	<b><code>zref</code> setup</b>	<b>2</b>
<b>4</b>	<b>Plumbing</b>	<b>3</b>
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 . . . . .	11
4.7	Property values . . . . .	12
<b>5</b>	<b>User interface</b>	<b>14</b>
5.1	<code>\zcheck</code> . . . . .	14
5.2	Targets . . . . .	16
<b>6</b>	<b>Checks</b>	<b>16</b>
6.1	Single label checks . . . . .	17
6.2	Setup . . . . .	17
6.3	Running . . . . .	18
6.4	Conditionals . . . . .	21
6.4.1	This page . . . . .	21
6.4.2	On page . . . . .	22
6.4.3	Before / After . . . . .	23
6.4.4	Pages . . . . .	23
6.4.5	Close / Far . . . . .	25
6.4.6	Chapter . . . . .	26
6.4.7	Section . . . . .	28
<b>7</b>	<b><code>zref-clever</code> integration</b>	<b>30</b>

---

\*This file describes v0.2.5, released 2022-02-11.

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

<b>8</b>	<b>zref-vario integration</b>	<b>31</b>
<b>Index</b>		<b>31</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.

```
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} {2022-02-11} {0.2.5}
15 {Flexible cross-references with contextual checks based on zref}
```

## 2 Dependencies

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

## 3 zref setup

`\g__zrefcheck_abschap_int` Provide absolute counters for section and chapter, and respective `zref` properties, so that we can make checks about relation of chapters/sections regardless of internal counters, since we don't get those for the unnumbered (starred) ones. Thanks Ulrike Fischer for suggestions at TeX.SX about the proper place to make the hooks for this purpose.

```
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_gzero:N \g__zrefcheck_abssec_int
```

```

25 }
26 \zref@newprop { zc@abschap } [0] { \int_use:N \g__zrefcheck_abschap_int }
27 \zref@addprop \ZREF@mainlist { zc@abschap }
28 \AddToHook { cmd / section / before }
29 { \int_gincr:N \g__zrefcheck_abssec_int }
30 \zref@newprop { zc@abssec } [0] { \int_use:N \g__zrefcheck_abssec_int }
31 \zref@addprop \ZREF@mainlist { zc@abssec }

```

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.

```

32 \zref@newlist { zrefcheck-check }
33 \zref@addprops { zrefcheck-check }
34 {
35   page , % for messages
36   abspage ,
37   zc@abschap ,
38   zc@abssec
39 }
40 \zref@newlist { zrefcheck-end }
41 \zref@addprops { zrefcheck-end }
42 {
43   abspage ,
44   zc@abschap ,
45   zc@abssec
46 }

```

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

```

47 \zref@newlist { zrefcheck-zrefvario }
48 \zref@addprops { zrefcheck-zrefvario }
49 {
50   page , % for messages
51   abspage ,
52 }

```

## 4 Plumbing

### 4.1 Messages

```

\__zrefcheck_message:nnnn
\__zrefcheck_message:nnnx
53 \cs_new_protected:Npn \__zrefcheck_message:nnnn #1#2#3#4
54 {
55   \use:c { msg_ \l__zrefcheck_msglevel_tl :nnnnn }
56   { zref-check } {#1} {#2} {#3} {#4}
57 }
58 \cs_generate_variant:Nn \__zrefcheck_message:nnnn { nnxx }

```

(End definition for `\__zrefcheck_message:nnnn`.)

```

59 \msg_new:nnn { zref-check } { check-failed }
60 { Failed~check~'~#1'~for~label~'~#2'~on~page~#3~\msg_line_context:. }
61 \msg_new:nnn { zref-check } { double-check }

```

```

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

```

## 4.2 Integer testing

`__zrefcheck_is_integer:n` 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.

```

102 \cs_new_eq:NN \__zrefcheck_int_to_roman:w \__int_to_roman:w
103 \prg_new_conditional:Npnn \__zrefcheck_is_integer:n #1 { p, T , F , TF }
104 {
105   \tl_if_empty:oTF {#1}
106   { \prg_return_false: }
107   {
108     \tl_if_empty:oTF { \__zrefcheck_int_to_roman:w -0#1 }
109     { \prg_return_true: }
110     { \prg_return_false: }
111   }
112 }

```

(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.

```

113 \prg_new_protected_conditional:Npnn \__zrefcheck_is_integer:rgx:n #1 { TF }
114 {
115   \regex_match:nnTF { \A\d+\Z } {#1}
116   { \prg_return_true: }
117   { \prg_return_false: }
118 }

```

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

## 4.3 Options

### hyperref option

```

\l__zrefcheck_use_hyperref_bool
\l__zrefcheck_warn_hyperref_bool
119 \bool_new:N \l__zrefcheck_use_hyperref_bool
120 \bool_new:N \l__zrefcheck_warn_hyperref_bool
121 \keys_define:nn { zref-check }
122 {
123   hyperref .choice: ,
124   hyperref / auto .code:n =
125   {
126     \bool_set_true:N \l__zrefcheck_use_hyperref_bool
127     \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
128   } ,
129   hyperref / true .code:n =
130   {
131     \bool_set_true:N \l__zrefcheck_use_hyperref_bool
132     \bool_set_true:N \l__zrefcheck_warn_hyperref_bool
133   } ,
134   hyperref / false .code:n =
135   {
136     \bool_set_false:N \l__zrefcheck_use_hyperref_bool

```

```

137         \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
138     } ,
139     hyperref .initial:n = auto ,
140     hyperref .default:n = auto
141 }

(End definition for \l__zrefcheck_use_hyperref_bool and \l__zrefcheck_warn_hyperref_bool.)

142 \AddToHook { begindocument }
143 {
144     \@ifpackageloaded { hyperref }
145     {
146         \bool_if:NT \l__zrefcheck_use_hyperref_bool
147         { \RequirePackage { zref-hyperref } }
148     }
149     {
150         \bool_if:NT \l__zrefcheck_warn_hyperref_bool
151         { \msg_warning:nn { zref-check } { missing-hyperref } }
152         \bool_set_false:N \l__zrefcheck_use_hyperref_bool
153     }
154     \keys_define:nn { zref-check }
155     {
156         hyperref .code:n =
157         { \msg_warning:nn { zref-check } { hyperref-preamble-only } }
158     }
159 }

```

### msglevel option

\l\_\_zrefcheck\_msglevel\_tl

```

160 \tl_new:N \l__zrefcheck_msglevel_tl
161 \keys_define:nn { zref-check }
162 {
163     msglevel .choice: ,
164     msglevel / warn .code:n =
165     { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } } ,
166     msglevel / info .code:n =
167     { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } } ,
168     msglevel / none .code:n =
169     { \tl_set:Nn \l__zrefcheck_msglevel_tl { none } } ,
170     msglevel / infoifdraft .code:n =
171     {
172         \ifdraft
173         { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
174         { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
175     } ,
176     msglevel / warniffinal .code:n =
177     {
178         \ifoptionfinal
179         { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
180         { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
181     } ,
182     msglevel / obeydraft .code:n =
183     {
184         % NOTE Option value deprecated in 2021-12-07 for v0.2.2.

```

```

185         \msg_warning:nnnn { zref-check }{ option-deprecated-with-alternative }
186         { msglevel=obeydraft } { msglevel=infoifdraft }
187     } ,
188     msglevel / obeyfinal .code:n =
189     {
190         % NOTE Option value deprecated in 2021-12-07 for v0.2.2.
191         \msg_warning:nnnn { zref-check }{ option-deprecated-with-alternative }
192         { msglevel=obeyfinal } { msglevel=warniffinal }
193     } ,
194     msglevel .value_required:n = true ,
195     msglevel .initial:n = warn ,

```

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

```

196     ignore .code:n =
197     { \msg_warning:nn { zref-check } { ignore-document-only } } ,
198     ignore .value_forbidden:n = true
199 }

```

(End definition for `\l__zrefcheck_msglevel_tl`.)

```

200 \AddToHook { begindocument }
201 {
202     \keys_define:nn { zref-check }
203     { ignore .meta:n = { msglevel = none } }
204 }

```

## onpage option

`\l__zrefcheck_msgonpage_bool`

```

205 \bool_new:N \l__zrefcheck_msgonpage_bool
206 \keys_define:nn { zref-check }
207 {
208     onpage .choice: ,
209     onpage / labelseq .code:n =
210     {
211         \bool_set_false:N \l__zrefcheck_msgonpage_bool
212     } ,
213     onpage / msg .code:n =
214     {
215         \bool_set_true:N \l__zrefcheck_msgonpage_bool
216     } ,
217     onpage / labelseqifdraft .code:n =
218     {
219         \ifdraft
220             { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
221             { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
222         } ,
223     onpage / msgiffinal .code:n =
224     {
225         \ifoptionfinal
226             { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
227             { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
228         } ,
229     onpage / obeydraft .code:n =
230     {

```

```

231      % NOTE Option value deprecated in 2021-12-07 for v0.2.2.
232      \msg_warning:nnnn { zref-check }{ option-deprecated-with-alternative }
233      { onpage=obeydraft } { onpage=labelseqifdraft }
234    } ,
235    onpage / obeyfinal .code:n =
236    {
237      % NOTE Option value deprecated in 2021-12-07 for v0.2.2.
238      \msg_warning:nnnn { zref-check }{ option-deprecated-with-alternative }
239      { onpage=obeyfinal } { onpage=msgiffinal }
240    } ,
241    onpage .value_required:n = true ,
242    onpage .initial:n = labelseq
243  }

```

(End definition for \l\_\_zrefcheck\_msgonpage\_bool.)

### closerange option

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

```

244 \int_new:N \l__zrefcheck_close_range_int
245 \keys_define:nn { zref-check }
246 {
247   closerange .code:n =
248   {
249     \__zrefcheck_is_integer_rgx:nTF {#1}
250     { \int_set:Nn \l__zrefcheck_close_range_int { \int_eval:n {#1} } }
251     {
252       \msg_warning:nn { zref-check } { closerange-not-positive-integer }
253       \int_set:Nn \l__zrefcheck_close_range_int { 5 }
254     }
255   } ,
256   closerange .value_required:n = true ,
257   closerange .initial:n = 5
258 }

```

(End definition for \l\_\_zrefcheck\_close\_range\_int.)

### labelcmd option

```

259 \keys_define:nn { zref-check }
260 {
261   labelcmd .code:n =
262   {
263     % NOTE Option value deprecated in 2022-02-08 for v0.2.4.
264     \msg_warning:nnn { zref-check }{ option-deprecated }
265     { labelcmd }
266   } ,
267 }

```

### Package options

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

```

268 \RequirePackage { l3keys2e }
269 \ProcessKeysOptions { zref-check }

```



```
\zrefchecksetup Provide \zrefchecksetup.
270 \NewDocumentCommand \zrefchecksetup { m }
271 { \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 `ltxcmds` documentation. Indeed, `\@newl@bel` receives 3 arguments, and `\zref@newlabel` just passes the first, the following two will be scanned ahead. Second, the `ltxcmds` 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`

```
272 \prop_new:N \g__zrefcheck_auxfile_lblseq_prop
```

(End definition for `\g__zrefcheck_auxfile_lblseq_prop`.)

```
273 \tl_gset:Nn \g_tmpa_tl { \c_sys_jobname_str .aux }
```

```
274 \file_if_exist:nT { \g_tmpa_tl }
```

```
275 {
```

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.

```
276 \ior_open:Nn \g_tmpa_ior { \g_tmpa_tl }
```

```
277 \group_begin:
```

```
278 \int_zero:N \l_tmpa_int
```

```
279 \tl_clear:N \l_tmpa_tl
```

```
280 \tl_clear:N \l_tmpb_tl
```

```
281 \bool_set_false:N \l_tmpa_bool
```

```
282 \ior_map_variable:NNn \g_tmpa_ior \l_tmpa_tl
```

```
283 {
```

```
284 \tl_map_variable:NNn \l_tmpa_tl \l_tmpb_tl
```

```
285 {
```

```
286 \tl_if_eq:NnTF \l_tmpb_tl { \zref@newlabel }
```

```
287 {
```

Found a `\zref@label`, signal it.

```
288 \bool_set_true:N \l_tmpa_bool
```

```
289 }
```

```
290 {
```

```
291 \bool_if:NTF \l_tmpa_bool
```

```
292 {
```

```
293 \bool_set_false:N \l_tmpa_bool
```

```
294 \int_incr:N \l_tmpa_int
```

```
295 \prop_gput:Nxx \g__zrefcheck_auxfile_lblseq_prop
```

```

296             { \l_tmpb_tl } { \int_use:N \l_tmpa_int }
297         }
298     {

```

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.

```

299         \tl_map_break:
300     }
301 }
302 }
303 }
304 \group_end:
305 \ior_close:N \g_tmpa_ior
306 }

```

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”.

## 4.5 Counter

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

```

307 \begingroup
308   \let \@addtoreset \ltx@gobbletwo
309   \newcounter { zrefcheck }
310 \endgroup
311 \setcounter { zrefcheck } { 0 }

```

## 4.6 Label formats

```

\__zrefcheck_check_lblfmt:n      \__zrefcheck_check_lblfmt:n {\check id int}}

```

```
312 \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}}
```

```
313 \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, 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}}
```

```
314 \cs_new_protected:Npn \zrefcheck_get_astl:nnn #1#2#3
```

```
315 {
316   \tl_clear:N #3
317   \tl_if_eq:nnTF {#2} { lblseq }
318   {
319     \prop_get:NnNF \g__zrefcheck_auxfile_lblseq_prop {#1} #3
320     {
321       \msg_warning:nnnn { zref-check }
322       { property-not-in-label } {#1} {#2}
323     }
324   }
325 }
```

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.

```
326 \zref@ifrefundefined {#1}
```

```

327     {}
328     {
329         \zref@ifpropundefined {#2}
330         { \msg_warning:nnnn { zref-check } { property-undefined } {#2} }
331         {
332             \zref@ifrefcontainsprop {#1} {#2}
333             {
334                 \tl_set:Nx #3
335                 { \zref@extractdefault {#1} {#2} { \c_empty_tl } }
336             }
337             {
338                 \msg_warning:nnnn
339                 { zref-check } { property-not-in-label } {#1} {#2}
340             }
341         }
342     }
343 }
344 }

```

(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).

```

345 \bool_new:N \l__zrefcheck_integer_bool

```

(End definition for \l\_\_zrefcheck\_integer\_bool.)

\l\_\_zrefcheck\_propval\_tl

```

346 \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>}
347 \cs_new_protected:Npn \zrefcheck_get_asint:nnn #1#2#3
348 {
349     \zrefcheck_get_astl:nnn {#1} {#2} { \l__zrefcheck_propval_tl }
350     \__zrefcheck_is_integer:nTF { \l__zrefcheck_propval_tl }
351     {

```

Make it an integer data type.

```

352     \int_set:Nn #3 { \int_eval:n { \l__zrefcheck_propval_tl } }
353     }
354     {
355     \bool_set_false:N \l__zrefcheck_integer_bool
356     \zref@ifrefundefined {#1}

```

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

```

357         { }
358         {
359             \msg_warning:nnnn { zref-check }
360             { property-not-integer } {#2} {#1}
361         }
362     }
363 }

```

(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{*}[\langle checks/options \rangle]{\langle labels \rangle}{\langle text \rangle}`

```

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

```

(End 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
366 \seq_new:N \l__zrefcheck_zcheck_labels_seq
367 \int_new:N \g__zrefcheck_id_int
368 \tl_new:N \l__zrefcheck_checkbeg_tl
369 \tl_new:N \l__zrefcheck_link_label_tl
370 \tl_new:N \l__zrefcheck_link_anchor_tl
371 \bool_new:N \l__zrefcheck_link_star_bool

```

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

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

`\__zrefcheck_zcheck:nnnn {\langle labels \rangle} {\langle * \rangle} {\langle checks/options \rangle} {\langle text \rangle}`

```

372 \cs_new_protected:Npn \__zrefcheck_zcheck:nnnn #1#2#3#4
373 {
374     \group_begin:

```

Process local options and checks.

```

375         \keys_set:nn { zref-check / zcheck } {#3}
376         \seq_set_from_clist:Nn \l__zrefcheck_zcheck_labels_seq {#1}

```

Names of the labels for this zcheck call.

```

377      \int_gincr:N \g__zrefcheck_id_int
378      \tl_set:Nx \l__zrefcheck_checkbeg_tl
379      { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }

```

Set checkbeg label.

```

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

```

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.

```

381      \seq_get:NN \l__zrefcheck_zcheck_labels_seq \l__zrefcheck_link_label_tl
382      \bool_set:Nn \l__zrefcheck_link_star_bool {#2}
383      \zref@ifrefundefined { \l__zrefcheck_link_label_tl }

```

If the reference is undefined, just typeset.

```

384      {#4}
385      {
386        \bool_if:nTF
387        {
388          \l__zrefcheck_use_hyperref_bool &&
389          ! \l__zrefcheck_link_star_bool
390        }
391        {
392          \exp_args:Nx \zrefcheck_get_astl:nnn
393          { \l__zrefcheck_link_label_tl }
394          { anchor } { \l__zrefcheck_link_anchor_tl }
395          \hyperlink { \l__zrefcheck_link_anchor_tl } {#4}
396        }
397        {#4}
398      }

```

Set checkend label.

```

399      \bool_if:NT \l__zrefcheck_zcheck_end_label_bool
400      {
401        \zref@labelbylist
402        { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
403        { zrefcheck-end }
404      }

```

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

```

405      \seq_map_function:NN \l__zrefcheck_zcheck_labels_seq \zref@refused

```

Run the checks.

```

406      \__zrefcheck_run_checks:nnx { \l__zrefcheck_zcheck_checks_seq }
407      { \l__zrefcheck_zcheck_labels_seq } { \l__zrefcheck_checkbeg_tl }
408      \group_end:
409      }

```

(End definition for  $\backslash\_zrefcheck\_zcheck:nnnn$ .)

## 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 \zref@label {#1}
415 #2
416 \tl_if_empty:nF {#2}
417 {
418 \zref@wrapper@babel
419 \zref@labelbylist { \__zrefcheck_end_lblfmt:n {#1} } { zrefcheck-end }
420 }
421 \group_end:
422 }

```

(End definition for \zctarget.)

```

zcregion      \begin{zcregion}{<label>}
...
\end{zcregion}
423 \NewDocumentEnvironment {zcregion} { m }
424 {
425 \refstepcounter { zrefcheck }
426 \zref@wrapper@babel \zref@label {#1}
427 }
428 {
429 \zref@wrapper@babel
430 \zref@labelbylist { \__zrefcheck_end_lblfmt:n {#1} } { zrefcheck-end }
431 }

```

(End 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  $\langle check \rangle$  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 *<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`

```

432 \seq_const_from_clist:Nn \c__zrefcheck_single_label_checks_seq
433 {
434   thischap ,
435   prevchap ,
436   nextchap ,
437   chapsbefore ,
438   chapsafter ,
439   thissec ,
440   prevsec ,
441   nextsec ,
442   secsbefore ,
443   secsafter ,
444 }
```

(End definition for `\c__zrefcheck_single_label_checks_seq`.)

## 6.2 Setup

`\l__zrefcheck_zcheck_checks_seq`  
`\l__zrefcheck_end_label_required_bool`

```

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

(End definition for `\l__zrefcheck_zcheck_checks_seq` and `\l__zrefcheck_end_label_required_bool`.)

First, we inherit all the main options into the keys of `zref-check / zcheck`.

```

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

Then we add the checks to it.

```

448 \clist_map_inline:nn
449 {
450   thispage ,
451   prevpage ,
452   nextpage ,
453   facing ,
```

```

454     otherpage ,
455     pagegap ,
456     above ,
457     below ,
458     pagesbefore ,
459     ppbefore ,
460     pagesafter ,
461     ppafter ,
462     before ,
463     after ,
464     thischap ,
465     prevchap ,
466     nextchap ,
467     chapsbefore ,
468     chapsafter ,
469     thissec ,
470     prevsec ,
471     nextsec ,
472     secsbefore ,
473     secsafter ,
474     close ,
475     far ,
476 }
477 {
478   \keys_define:nn { 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>}}
\__zrefcheck_run_checks:nnn {\<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_map_inline:Nn #1
495     { \__zrefcheck_do_check:nnn {####1} {##1} {#3} }
496   }
497   \group_end:
498 }
499 \cs_generate_variant:Nn \__zrefcheck_run_checks:nnn { nnx }

```

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

```

\l__zrefcheck_passedcheck_bool
\l__zrefcheck_onpage_bool
\c__zrefcheck_onpage_checks_seq
500 \bool_new:N \l__zrefcheck_passedcheck_bool
501 \bool_new:N \l__zrefcheck_onpage_bool
502 \seq_const_from_clist:Nn \c__zrefcheck_onpage_checks_seq
503 { 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.
504 \cs_generate_variant:Nn \exp_args:Nnno { Nnoo }

\__zrefcheck_do_check:nnn
\__zrefcheck_do_check:nnn {<check>} {<label beg>} {<reference beg>}
505 \cs_new_protected:Npn \__zrefcheck_do_check:nnn #1#2#3
506 {
507 \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.
508 \zref@ifrefundefined {#2}
509 {}
510 {
511 \tl_if_empty:nF {#1}
512 {
513 \bool_set_true:N \l__zrefcheck_passedcheck_bool
514 \bool_set_false:N \l__zrefcheck_onpage_bool
515 \cs_if_exist:cTF { __zrefcheck_check_ #1 :nnF }
516 {
517 % ‘‘label beg’’ vs ‘‘reference beg’’.
518 \use:c { __zrefcheck_check_ #1 :nnF }
519 {#2} {#3}
520 { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
521 % ‘‘reference end’’ \emph{may} exist or not depending on the
522 % checks.
523 \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#3} }
524 {
525 % ‘‘label end’’ \emph{may} have been created by the
526 % target commands.
527 \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
528 {}
529 {
530 % ‘‘label end’’ vs ‘‘reference beg’’.
531 \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
532 { \__zrefcheck_end_lblfmt:n {#2} } {#3}
533 { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
534 }
535 }
536 {
537 % ‘‘label beg’’ vs ‘‘reference end’’.
538 \exp_args:Nnno \use:c { __zrefcheck_check_ #1 :nnF }
539 {#2} { \__zrefcheck_end_lblfmt:n {#3} }
540 { \bool_set_false:N \l__zrefcheck_passedcheck_bool }

```

```

541 % ‘‘label end’’ \emph{may} have been created by the
542 % target commands.
543 \zref@ifrefundefined { \_zrefcheck\_end\_lblfmt:n {#2} }
544 {}
545 {
546 % ‘‘label end’’ vs ‘‘reference beg’’.
547 \exp\_args:Nno \use:c { \_zrefcheck\_check\_ \_1 :nnF }
548 { \_zrefcheck\_end\_lblfmt:n {#2} } {#3}
549 { \bool\_set\_false:N \l\_zrefcheck\_passedcheck\_bool }
550 % ‘‘label end’’ vs ‘‘reference end’’.
551 \exp\_args:Nnoo \use:c { \_zrefcheck\_check\_ \_1 :nnF }
552 { \_zrefcheck\_end\_lblfmt:n {#2} }
553 { \_zrefcheck\_end\_lblfmt:n {#3} }
554 { \bool\_set\_false:N \l\_zrefcheck\_passedcheck\_bool }
555 }
556 }

```

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.

```

557 \seq\_if\_in:NnT \c\_zrefcheck\_onpage\_checks\_seq {#1}
558 {
559 \_zrefcheck\_check\_thispage:nnT
560 {#2} {#3}
561 { \bool\_set\_true:N \l\_zrefcheck\_onpage\_bool }
562 \zref@ifrefundefined { \_zrefcheck\_end\_lblfmt:n {#3} }
563 {
564 \zref@ifrefundefined { \_zrefcheck\_end\_lblfmt:n {#2} }
565 {}
566 {
567 \_zrefcheck\_check\_thispage:nnT
568 { \_zrefcheck\_end\_lblfmt:n {#2} } {#3}
569 { \bool\_set\_true:N \l\_zrefcheck\_onpage\_bool }
570 }
571 }
572 {
573 \_zrefcheck\_check\_thispage:nnT
574 {#2} { \_zrefcheck\_end\_lblfmt:n {#3} }
575 { \bool\_set\_true:N \l\_zrefcheck\_onpage\_bool }
576 \zref@ifrefundefined { \_zrefcheck\_end\_lblfmt:n {#2} }
577 {}
578 {
579 \_zrefcheck\_check\_thispage:nnT
580 { \_zrefcheck\_end\_lblfmt:n {#2} } {#3}
581 { \bool\_set\_true:N \l\_zrefcheck\_onpage\_bool }
582 \_zrefcheck\_check\_thispage:nnT
583 { \_zrefcheck\_end\_lblfmt:n {#2} }
584 { \_zrefcheck\_end\_lblfmt:n {#3} }
585 { \bool\_set\_true:N \l\_zrefcheck\_onpage\_bool }
586 }
587 }
588 }
589 \bool\_if:NtF \l\_zrefcheck\_passedcheck\_bool
590 {

```

```

591         \bool_if:nT
592         {
593             \l__zrefcheck_msgonpage_bool &&
594             \l__zrefcheck_onpage_bool
595         }
596         {
597             \__zrefcheck_message:nnnx { double-check } {#1} {#2}
598             { \zref@extractdefault {#3} {page} {'unknown'} }
599         }
600     }
601     {
602         \__zrefcheck_message:nnnx { check-failed } {#1} {#2}
603         { \zref@extractdefault {#3} {page} {'unknown'} }
604     }
605 }
606 { \msg_warning:nnn { zref-check } { check-missing } {#1} }
607 }
608 }
609 \group_end:
610 }
611 \cs_generate_variant:Nn \__zrefcheck_do_check:nnn { nnV }

```

(End 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.

```

612 \int_new:N \l__zrefcheck_lbl_int
613 \int_new:N \l__zrefcheck_ref_int
614 \int_new:N \l__zrefcheck_lbl_b_int
615 \int_new:N \l__zrefcheck_ref_b_int

```

(End definition for \l\_\_zrefcheck\_lbl\_int and others.)

### 6.4.1 This page

```

\__zrefcheck_check_thispage:nn
\__zrefcheck_check_otherpage:nn

```

```

616 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thispage:nn #1#2 { T , F , TF }
617 {
618     \group_begin:
619     \bool_set_true:N \l__zrefcheck_integer_bool
620     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
621     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
622     \bool_lazy_and:nnTF
623     { \l__zrefcheck_integer_bool }
624     {
625         \int_compare_p:nNn
626         { \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.

```

627         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&

```

```

628         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
629     }
630     { \group_insert_after:N \prg_return_true: }
631     { \group_insert_after:N \prg_return_false: }
632 \group_end:
633 }
634 \prg_new_protected_conditional:Npnn \__zrefcheck_check_otherpage:nn #1#2 { T , F , TF }
635 {
636     \__zrefcheck_check_thispage:nnTF {#1} {#2}
637     { \prg_return_false: }
638     { \prg_return_true: }
639 }

```

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

## 6.4.2 On page

```

\__zrefcheck_check_above:nn
\__zrefcheck_check_below:nn
640 \prg_new_protected_conditional:Npnn \__zrefcheck_check_above:nn #1#2 { F , TF }
641 {
642     \group_begin:
643     \__zrefcheck_check_thispage:nnTF {#1} {#2}
644     {
645         \bool_set_true:N \l__zrefcheck_integer_bool
646         \zrefcheck_get_asint:nnn {#1} { lblseq } { \l__zrefcheck_lbl_int }
647         \zrefcheck_get_asint:nnn {#2} { lblseq } { \l__zrefcheck_ref_int }
648         \bool_lazy_and:nnTF
649         { \l__zrefcheck_integer_bool }
650         {
651             \int_compare_p:nNn
652             { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
653             ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
654             ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
655         }
656         { \group_insert_after:N \prg_return_true: }
657         { \group_insert_after:N \prg_return_false: }
658     }
659     { \group_insert_after:N \prg_return_false: }
660 \group_end:
661 }
662 \prg_new_protected_conditional:Npnn \__zrefcheck_check_below:nn #1#2 { F , TF }
663 {
664     \__zrefcheck_check_thispage:nnTF {#1} {#2}
665     {
666         \__zrefcheck_check_above:nnTF {#1} {#2}
667         { \prg_return_false: }
668         { \prg_return_true: }
669     }
670     { \prg_return_false: }
671 }

```

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

### 6.4.3 Before / After

```

\__zrefcheck_check_before:nn
\__zrefcheck_check_after:nn
672 \prg_new_protected_conditional:Npnn \__zrefcheck_check_before:nn #1#2 { F }
673 {
674   \__zrefcheck_check_pagesbefore:nnTF {#1} {#2}
675   { \prg_return_true: }
676   {
677     \__zrefcheck_check_above:nnTF {#1} {#2}
678     { \prg_return_true: }
679     { \prg_return_false: }
680   }
681 }
682 \prg_new_protected_conditional:Npnn \__zrefcheck_check_after:nn #1#2 { F }
683 {
684   \__zrefcheck_check_pagesafter:nnTF {#1} {#2}
685   { \prg_return_true: }
686   {
687     \__zrefcheck_check_below:nnTF {#1} {#2}
688     { \prg_return_true: }
689     { \prg_return_false: }
690   }
691 }

```

(End 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
692 \prg_new_protected_conditional:Npnn \__zrefcheck_check_nextpage:nn #1#2 { F }
693 {
694   \group_begin:
695   \bool_set_true:N \l__zrefcheck_integer_bool
696   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
697   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
698   \bool_lazy_and:nnTF
699   { \l__zrefcheck_integer_bool }
700   {
701     \int_compare_p:nNn
702     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
703     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
704     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
705   }
706   { \group_insert_after:N \prg_return_true: }
707   { \group_insert_after:N \prg_return_false: }
708   \group_end:
709 }
710 \prg_new_protected_conditional:Npnn \__zrefcheck_check_prevpage:nn #1#2 { F }
711 {
712   \group_begin:
713   \bool_set_true:N \l__zrefcheck_integer_bool
714   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
715   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
716   \bool_lazy_and:nnTF

```

```

717     { \l__zrefcheck_integer_bool }
718     {
719         \int_compare_p:nNn
720         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
721         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
722         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
723     }
724     { \group_insert_after:N \prg_return_true: }
725     { \group_insert_after:N \prg_return_false: }
726 \group_end:
727 }
728 \prg_new_protected_conditional:Npnn \__zrefcheck_check_pagesbefore:nn #1#2 { F , TF }
729 {
730     \group_begin:
731     \bool_set_true:N \l__zrefcheck_integer_bool
732     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
733     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
734     \bool_lazy_and:nnTF
735     { \l__zrefcheck_integer_bool }
736     {
737         \int_compare_p:nNn
738         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
739         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
740         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
741     }
742     { \group_insert_after:N \prg_return_true: }
743     { \group_insert_after:N \prg_return_false: }
744 \group_end:
745 }
746 \cs_new_eq:NN \__zrefcheck_check_ppbefore:nnF \__zrefcheck_check_pagesbefore:nnF
747 \prg_new_protected_conditional:Npnn \__zrefcheck_check_pagesafter:nn #1#2 { F , TF }
748 {
749     \group_begin:
750     \bool_set_true:N \l__zrefcheck_integer_bool
751     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
752     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
753     \bool_lazy_and:nnTF
754     { \l__zrefcheck_integer_bool }
755     {
756         \int_compare_p:nNn
757         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
758         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
759         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
760     }
761     { \group_insert_after:N \prg_return_true: }
762     { \group_insert_after:N \prg_return_false: }
763 \group_end:
764 }
765 \cs_new_eq:NN \__zrefcheck_check_ppafter:nnF \__zrefcheck_check_pagesafter:nnF
766 \prg_new_protected_conditional:Npnn \__zrefcheck_check_pagegap:nn #1#2 { F }
767 {
768     \group_begin:
769     \bool_set_true:N \l__zrefcheck_integer_bool
770     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }

```



```

771 \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
772 \bool_lazy_and:nnTF
773 { \l__zrefcheck_integer_bool }
774 {
775   \int_compare_p:nNn
776     { \int_abs:n { \l__zrefcheck_lbl_int - \l__zrefcheck_ref_int } } > { 1 } &&
777     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
778     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
779   }
780   { \group_insert_after:N \prg_return_true: }
781   { \group_insert_after:N \prg_return_false: }
782 \group_end:
783 }
784 \prg_new_protected_conditional:Npnn \__zrefcheck_check_facing:nn #1#2 { F }
785 {
786   \group_begin:
787   \bool_set_true:N \l__zrefcheck_integer_bool
788   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
789   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
790   \bool_lazy_and:nnTF
791     { \l__zrefcheck_integer_bool }
792     {

```

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

```

793   \legacy_if_p:n { @twoside } &&

```

Now we test “facing”.

```

794   (
795     (
796       \int_if_odd_p:n { \l__zrefcheck_ref_int } &&
797       \int_compare_p:nNn
798         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 }
799     ) ||
800     (
801       \int_if_even_p:n { \l__zrefcheck_ref_int } &&
802       \int_compare_p:nNn
803         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 }
804     )
805   ) &&
806   ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
807   ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
808 }
809 { \group_insert_after:N \prg_return_true: }
810 { \group_insert_after:N \prg_return_false: }
811 \group_end:
812 }

```

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

### 6.4.5 Close / Far

```

\__zrefcheck_check_close:nn
\__zrefcheck_check_far:nn
813 \prg_new_protected_conditional:Npnn \__zrefcheck_check_close:nn #1#2 { F , TF }
814 {
815   \group_begin:

```

```

816 \bool_set_true:N \l__zrefcheck_integer_bool
817 \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
818 \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
819 \bool_lazy_and:nnTF
820 { \l__zrefcheck_integer_bool }
821 {
822   \int_compare_p:nNn
823     { \int_abs:n { \l__zrefcheck_lbl_int - \l__zrefcheck_ref_int } }
824     <
825     { \l__zrefcheck_close_range_int + 1 } &&
826     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
827     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
828   }
829   { \group_insert_after:N \prg_return_true: }
830   { \group_insert_after:N \prg_return_false: }
831 \group_end:
832 }
833 \prg_new_protected_conditional:Npnn \__zrefcheck_check_far:nn #1#2 { F }
834 {
835   \__zrefcheck_check_close:nnTF {#1} {#2}
836   { \prg_return_false: }
837   { \prg_return_true: }
838 }

```

(End 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
839 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thischap:nn #1#2 { F }
840 {
841   \group_begin:
842     \bool_set_true:N \l__zrefcheck_integer_bool
843     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
844     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
845     \bool_lazy_and:nnTF
846     { \l__zrefcheck_integer_bool }
847     {
848       \int_compare_p:nNn
849         { \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.

```

850     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
851     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
852   }
853   { \group_insert_after:N \prg_return_true: }
854   { \group_insert_after:N \prg_return_false: }
855 \group_end:
856 }
857 \prg_new_protected_conditional:Npnn \__zrefcheck_check_nextchap:nn #1#2 { F }

```

```

858 {
859   \group_begin:
860     \bool_set_true:N \l__zrefcheck_integer_bool
861     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
862     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
863     \bool_lazy_and:nnTF
864       { \l__zrefcheck_integer_bool }
865       {
866         \int_compare_p:nNn
867           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
868           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
869       }
870       { \group_insert_after:N \prg_return_true: }
871       { \group_insert_after:N \prg_return_false: }
872   \group_end:
873 }
874 \prg_new_protected_conditional:Npnn \__zrefcheck_check_prevchap:nn #1#2 { F }
875 {
876   \group_begin:
877     \bool_set_true:N \l__zrefcheck_integer_bool
878     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
879     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
880     \bool_lazy_and:nnTF
881       { \l__zrefcheck_integer_bool }
882       {
883         \int_compare_p:nNn
884           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
885           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
886           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
887       }
888       { \group_insert_after:N \prg_return_true: }
889       { \group_insert_after:N \prg_return_false: }
890   \group_end:
891 }
892 \prg_new_protected_conditional:Npnn \__zrefcheck_check_chapsafter:nn #1#2 { F }
893 {
894   \group_begin:
895     \bool_set_true:N \l__zrefcheck_integer_bool
896     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
897     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
898     \bool_lazy_and:nnTF
899       { \l__zrefcheck_integer_bool }
900       {
901         \int_compare_p:nNn
902           { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
903           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
904       }
905       { \group_insert_after:N \prg_return_true: }
906       { \group_insert_after:N \prg_return_false: }
907   \group_end:
908 }
909 \prg_new_protected_conditional:Npnn \__zrefcheck_check_chapsbefore:nn #1#2 { F }
910 {
911   \group_begin:

```

```

912 \bool_set_true:N \l__zrefcheck_integer_bool
913 \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
914 \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
915 \bool_lazy_and:nnTF
916 { \l__zrefcheck_integer_bool }
917 {
918   \int_compare_p:nNn
919     { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
920     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
921     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
922   }
923   { \group_insert_after:N \prg_return_true: }
924   { \group_insert_after:N \prg_return_false: }
925 \group_end:
926 }

```

(End 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
927 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thissec:nn #1#2 { F }
928 {
929   \group_begin:
930   \bool_set_true:N \l__zrefcheck_integer_bool
931   \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
932   \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
933   \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
934   \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
935   \bool_lazy_and:nnTF
936   { \l__zrefcheck_integer_bool }
937   {
938     \int_compare_p:nNn
939       { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
940     \int_compare_p:nNn
941       { \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.

```

942     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
943     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
944   }
945   { \group_insert_after:N \prg_return_true: }
946   { \group_insert_after:N \prg_return_false: }
947 \group_end:
948 }
949 \prg_new_protected_conditional:Npnn \__zrefcheck_check_nextsec:nn #1#2 { F }
950 {
951   \group_begin:
952   \bool_set_true:N \l__zrefcheck_integer_bool
953   \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }

```

```

954 \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
955 \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
956 \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
957 \bool_lazy_and:nnTF
958 { \l__zrefcheck_integer_bool }
959 {
960   \int_compare_p:nNn
961     { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
962   \int_compare_p:nNn
963     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
964   ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
965 }
966 { \group_insert_after:N \prg_return_true: }
967 { \group_insert_after:N \prg_return_false: }
968 \group_end:
969 }
970 \prg_new_protected_conditional:Npnn \__zrefcheck_check_prevsec:nn #1#2 { F }
971 {
972   \group_begin:
973     \bool_set_true:N \l__zrefcheck_integer_bool
974     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
975     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
976     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
977     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
978     \bool_lazy_and:nnTF
979     { \l__zrefcheck_integer_bool }
980     {
981       \int_compare_p:nNn
982         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
983       \int_compare_p:nNn
984         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
985       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
986       ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
987     }
988     { \group_insert_after:N \prg_return_true: }
989     { \group_insert_after:N \prg_return_false: }
990   \group_end:
991 }
992 \prg_new_protected_conditional:Npnn \__zrefcheck_check_secsafter:nn #1#2 { F }
993 {
994   \group_begin:
995     \bool_set_true:N \l__zrefcheck_integer_bool
996     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
997     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
998     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
999     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
1000     \bool_lazy_and:nnTF
1001     { \l__zrefcheck_integer_bool }
1002     {
1003       \int_compare_p:nNn
1004         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
1005       \int_compare_p:nNn
1006         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
1007       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }

```

```

1008     }
1009     { \group_insert_after:N \prg_return_true: }
1010     { \group_insert_after:N \prg_return_false: }
1011   \group_end:
1012 }
1013 \prg_new_protected_conditional:Npnn \__zrefcheck_check_secsbefore:nn #1#2 { F }
1014 {
1015   \group_begin:
1016     \bool_set_true:N \l__zrefcheck_integer_bool
1017     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
1018     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
1019     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
1020     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
1021     \bool_lazy_and:nnTF
1022       { \l__zrefcheck_integer_bool }
1023       {
1024         \int_compare_p:nNn
1025           { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
1026         \int_compare_p:nNn
1027           { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
1028         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
1029         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
1030       }
1031     { \group_insert_after:N \prg_return_true: }
1032     { \group_insert_after:N \prg_return_false: }
1033   \group_end:
1034 }

```

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

## 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 `\zceref`: 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_zceref_beg_label:
  \zrefcheck_zceref_end_label_maybe:
  \zrefcheck_zceref_run_checks_on_labels:n
1035 \cs_new_protected:Npn \zrefcheck_zceref_beg_label:
1036 {
1037   \int_gincr:N \g__zrefcheck_id_int
1038   \tl_set:Nx \l__zrefcheck_checkbeg_tl
1039     { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
1040   \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrefcheck-check }
1041 }
1042 \cs_new_protected:Npn \zrefcheck_zceref_end_label_maybe:
1043 {
1044   \bool_if:NT \l__zrefcheck_zcheck_end_label_bool
1045   {
1046     \zref@labelbylist
1047       { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
1048       { zrefcheck-end }

```

```

1049     }
1050   }
1051   \cs_new_protected:Npn \zrefcheck_zceref_run_checks_on_labels:n #1
1052   {
1053     \__zrefcheck_run_checks:nnx
1054     { \l__zrefcheck_zcheck_checks_seq } {#1} { \l__zrefcheck_checkbeg_tl }
1055   }

```

(End definition for `\zrefcheck_zceref_beg_label:`, `\zrefcheck_zceref_end_label_maybe:`, and `\zrefcheck_zceref_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
1056 \cs_new_protected:Npn \zrefcheck_zrefvario_label:
1057 {
1058   \int_gincr:N \g__zrefcheck_id_int
1059   \tl_set:Nx \l__zrefcheck_checkbeg_tl
1060   { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
1061   \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrefcheck-zrefvario }
1062 }
1063 \cs_new_protected:Npn \zrefcheck_zrefvario_run_check_on_label:nn #1#2
1064 { \__zrefcheck_do_check:nnV {#1} {#2} \l__zrefcheck_checkbeg_tl }
1065 \cs_generate_variant:Nn \zrefcheck_zrefvario_run_check_on_label:nn { Vn }

```

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

```

1066 \</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	
<code>\</code>	74
A	
<code>\A</code>	115
<code>\AddToHook</code>	21, 28, 142, 200
B	
<code>\begingroup</code>	307
bool commands:	
<code>\bool_if:N</code>	146, 150, 291, 399, 589, 1044
<code>\bool_if:n</code>	386, 591
<code>\bool_lazy_and:n</code>	13, 622, 648, 698, 716, 734, 753, 772, 790, 819, 845, 863, 880, 898, 915, 935, 957, 978, 1000, 1021
C	
<code>\catcode</code>	11
<code>\chapter</code>	2, 26
clist commands:	
<code>\clist_map_inline:nn</code>	448
<code>\bool_new:N</code>	119, 120, 205, 345, 371, 446, 500, 501
<code>\bool_set:Nn</code>	382
<code>\bool_set_false:N</code>	127, 136, 137, 152, 211, 220, 227, 281, 293, 355, 514, 520, 533, 540, 549, 554
<code>\bool_set_true:N</code>	126, 131, 132, 215, 221, 226, 288, 484, 513, 561, 569, 575, 581, 585, 619, 645, 695, 713, 731, 750, 769, 787, 816, 842, 860, 877, 895, 912, 930, 952, 973, 995, 1016
<code>\l_tmpa_bool</code>	11, 281, 288, 291, 293

cs commands:		\int_compare_p:nNn .....	
\cs_generate_variant:Nn .....		..... 625, 627, 628, 651,	
..... 58, 499, 504, 611, 1065		653, 654, 701, 703, 704, 719, 721,	
\cs_if_exist:NTF .....	515	722, 737, 739, 740, 756, 758, 759,	
\cs_new:Npn .....	312, 313	775, 777, 778, 797, 802, 806, 807,	
\cs_new_eq:NN .....	102, 746, 765	822, 826, 827, 848, 850, 851, 866,	
\cs_new_protected:Npn .....		868, 883, 885, 886, 901, 903, 918,	
..... 53, 314, 347, 372,		920, 921, 938, 940, 942, 943, 960,	
489, 505, 1035, 1042, 1051, 1056, 1063		962, 964, 981, 983, 985, 986, 1003,	
		1005, 1007, 1024, 1026, 1028, 1029	
D		\int_eval:n .....	250, 352
\d .....	115	\int_gincr:N .. 23, 29, 377, 1037, 1058	
E		\int_gzero:N .....	24
\emph .....	521, 525, 541	\int_if_even_p:n .....	801
\endgroup .....	310	\int_if_odd_p:n .....	796
\endinput .....	12	\int_incr:N .....	294
exp commands:		\int_new:N .....	
\exp_args:Nnno .....	504, 538	.. 19, 20, 244, 367, 612, 613, 614, 615	
\exp_args:Nno .....	531, 547	\int_set:Nn .....	250, 253, 352
\exp_args:Nnoo .....	551	\int_use:N .....	26, 30, 296, 312
\exp_args:Nx .....	392	\int_zero:N .....	278
F		\l_tmpa_int .....	278, 294, 296
file commands:		int internal commands:	
\file_if_exist:nTF .....	274	\__int_to_roman:w .....	4, 5, 102
\fmtversion .....	3	ior commands:	
G		\ior_close:N .....	305
group commands:		\ior_map_variable:NNn .....	282
\group_begin: .....	277, 374,	\ior_open:Nn .....	276
412, 491, 507, 618, 642, 694, 712,		\g_tmpa_ior .....	276, 282, 305
730, 749, 768, 786, 815, 841, 859,		iow commands:	
876, 894, 911, 929, 951, 972, 994, 1015		\iow_char:N .....	74
\group_end: .....	304, 408,	\iow_newline: .....	73, 77, 80, 97
421, 497, 609, 632, 660, 708, 726,		K	
744, 763, 782, 811, 831, 855, 872,		keys commands:	
890, 907, 925, 947, 968, 990, 1011, 1033		\keys_define:nn .....	121,
\group_insert_after:N .....		154, 161, 202, 206, 245, 259, 447, 478	
..... 630, 631, 656, 657,		\keys_set:nn .....	271, 375
659, 706, 707, 724, 725, 742, 743,		L	
761, 762, 780, 781, 809, 810, 829,		\label .....	11
830, 853, 854, 870, 871, 888, 889,		legacy commands:	
905, 906, 923, 924, 945, 946, 966,		\legacy_if_p:n .....	793
967, 988, 989, 1009, 1010, 1031, 1032		\let .....	308
H		M	
\hyperlink .....	14, 395	\MessageBreak .....	10
I		msg commands:	
\ifdraft .....	172, 219	\msg_line_context: .....	60,
\IfFormatAtLeastTF .....	3, 4	62, 64, 66, 68, 70, 84, 87, 97, 101	
\ifoptionfinal .....	178, 225	\msg_new:nnn .....	59, 61, 63, 65,
int commands:		67, 69, 71, 76, 78, 83, 85, 90, 95, 100	
\int_abs:n .....	776, 823	\msg_warning:nn ... 151, 157, 197, 252	
		\msg_warning:nnn .....	264, 606



<code>\msg_warning:nnnn</code> . . . . .	185, 191, 232, 238, 321, 330, 338, 359	sys commands:	
		<code>\c_sys_jobname_str</code> . . . . .	273
<b>N</b>			
<code>\newcounter</code> . . . . .	309	<b>T</b>	
<code>\NewDocumentCommand</code> . . . . .	270, 364, 410	TeX and L <sup>A</sup> T <sub>E</sub> X 2 <sub>ε</sub> commands:	
<code>\NewDocumentEnvironment</code> . . . . .	423	<code>\@addtoreset</code> . . . . .	308
<b>P</b>		<code>\@currentlabel</code> . . . . .	16
<code>\PackageError</code> . . . . .	7	<code>\@ifl@t@r</code> . . . . .	3
<code>\pageref</code> . . . . .	11	<code>\@ifpackageloaded</code> . . . . .	144
prg commands:		<code>\@newl@bel</code> . . . . .	9
<code>\prg_new_conditional:Npnn</code> . . . . .	103	<code>\ltx@gobbletwo</code> . . . . .	308
<code>\prg_new_protected_conditional:Npnn</code>		<code>\zref@addprop</code> . . . . .	17, 27, 31
. . . . .	16, 113, 616, 634,	<code>\zref@addprops</code> . . . . .	33, 41, 48
640, 662, 672, 682, 692, 710, 728,		<code>\zref@extractdefault</code> 12, 335, 598, 603	
747, 766, 784, 813, 833, 839, 857,		<code>\zref@ifpropundefined</code> . . . . .	329
874, 892, 909, 927, 949, 970, 992, 1013		<code>\zref@ifrefcontainsprop</code> . . . . .	332
<code>\prg_return_false:</code> . . . . .	16,	<code>\zref@ifrefundefined</code> 12, 326, 356,	
106, 110, 117, 631, 637, 657, 659,		383, 508, 523, 527, 543, 562, 564, 576	
667, 670, 679, 689, 707, 725, 743,		<code>\ZREF@label</code> . . . . .	11
762, 781, 810, 830, 836, 854, 871,		<code>\zref@label</code> . . . . .	10, 11, 414, 426
889, 906, 924, 946, 967, 989, 1010, 1032		<code>\zref@labelbylist</code> . . . . .	
<code>\prg_return_true:</code> . . . . .	16,	. . . . .	380, 401, 419, 430, 1040, 1046, 1061
109, 116, 630, 638, 656, 668, 675,		<code>\ZREF@mainlist</code> . . . . .	27, 31
678, 685, 688, 706, 724, 742, 761,		<code>\zref@newlabel</code> . . . . .	9–11, 286
780, 809, 829, 837, 853, 870, 888,		<code>\zref@newlist</code> . . . . .	32, 40, 47
905, 923, 945, 966, 988, 1009, 1031		<code>\zref@newprop</code> . . . . .	17, 26, 30
<code>\ProcessKeysOptions</code> . . . . .	269	<code>\zref@refused</code> . . . . .	12, 405
prop commands:		<code>\zref@require@unique</code> . . . . .	11
<code>\prop_get:NnNTF</code> . . . . .	319	<code>\zref@wrapper@babel</code> . . . . .	
<code>\prop_gput:Nnn</code> . . . . .	295	. . . . .	9, 14, 365, 414, 418, 426, 429
<code>\prop_new:N</code> . . . . .	272	tl commands:	
<code>\providecommand</code> . . . . .	3	<code>\c_empty_tl</code> . . . . .	12, 335
<code>\ProvidesExplPackage</code> . . . . .	14	<code>\tl_clear:N</code> . . . . .	12, 279, 280, 316
<b>R</b>		<code>\tl_gset:Nn</code> . . . . .	273
<code>\refstepcounter</code> . . . . .	11, 16, 413, 425	<code>\tl_if_blank:nTF</code> . . . . .	4
regex commands:		<code>\tl_if_empty:nTF</code> . . . . .	4, 105, 108, 416, 511
<code>\regex_match:nnTF</code> . . . . .	115	<code>\tl_if_eq:NnTF</code> . . . . .	286
<code>\RequirePackage</code> . . . . .	16, 17, 18, 147, 268	<code>\tl_if_eq:nnTF</code> . . . . .	317
<code>\romannumeral</code> . . . . .	4	<code>\tl_map_break:</code> . . . . .	299
<b>S</b>		<code>\tl_map_variable:NNn</code> . . . . .	284
<code>\section</code> . . . . .	28	<code>\tl_new:N</code> . . . . .	160, 346, 368, 369, 370
seq commands:		<code>\tl_set:Nn</code> . . . . .	165, 167, 169,
<code>\seq_const_from_clist:Nn</code> . . . . .	432, 502	. . . . .	173, 174, 179, 180, 334, 378, 1038, 1059
<code>\seq_get:NN</code> . . . . .	381	<code>\g_tmpa_tl</code> . . . . .	273, 274, 276
<code>\seq_if_in:NnTF</code> . . . . .	483, 557	<code>\l_tmpa_tl</code> . . . . .	279, 282, 284
<code>\seq_map_function:NN</code> . . . . .	405	<code>\l_tmpb_tl</code> . . . . .	280, 284, 286, 296
<code>\seq_map_inline:Nn</code> . . . . .	492, 494	<b>U</b>	
<code>\seq_new:N</code> . . . . .	366, 445	use commands:	
<code>\seq_put_right:Nn</code> . . . . .	482	<code>\use:N</code> . . . . .	55, 518, 531, 538, 547, 551
<code>\seq_set_from_clist:Nn</code> . . . . .	376	<b>Z</b>	
<code>\setcounter</code> . . . . .	311	<code>\Z</code> . . . . .	115
		<code>\zcheck</code> . . . . .	14, 17, 364

\zcref .....	30	\__zrefcheck_check_pagesafter:nnTF	
zcregion .....	423	.....	684, 765
\zctarget .....	16, 410	\__zrefcheck_check_pagesbefore:nn	
\zref .....	14	.....	692
zrefcheck commands:			
\zrefcheck_get_asint:nnn .	13, 17,	\__zrefcheck_check_pagesbefore:nnTF	
	347, 620, 621, 646, 647, 696, 697,	.....	674, 746
	714, 715, 732, 733, 751, 752, 770,	\__zrefcheck_check_ppafter:nn .	692
	771, 788, 789, 817, 818, 843, 844,	\__zrefcheck_check_ppafter:nnTF	765
	861, 862, 878, 879, 896, 897, 913,	\__zrefcheck_check_ppbefore:nn .	692
	914, 931, 932, 933, 934, 953, 954,	\__zrefcheck_check_ppbefore:nnTF	746
	955, 956, 974, 975, 976, 977, 996,	\__zrefcheck_check_prevchap:nn .	839
	997, 998, 999, 1017, 1018, 1019, 1020	\__zrefcheck_check_prevpage:nn .	692
\zrefcheck_get_astl:nnn .....		\__zrefcheck_check_prevsec:nn .	927
.....	12, 13, 17, 314, 349, 392	\__zrefcheck_check_secsafter:nn	927
\zrefcheck_zcref_beg_label: .	1035	\__zrefcheck_check_secsbefore:nn	927
\zrefcheck_zcref_end_label_-		\__zrefcheck_check_thischap:nn .	839
maybe: .....	1035	\__zrefcheck_check_thispage:nn .	616
\zrefcheck_zcref_run_checks_on_-		\__zrefcheck_check_thispage:nnTF	
labels:n .....	1035	559, 567, 573, 579, 582, 636, 643, 664	
\zrefcheck_zrefvario_label: .	1056	\__zrefcheck_check_thissec:nn .	927
\zrefcheck_zrefvario_run_check_-		\l__zrefcheck_checkbeg_tl .....	
on_label:n .....	1056	... 366, 378, 380, 402, 407, 1038,	
\zrefcheck_zrefvario_run_check_-		1040, 1047, 1054, 1059, 1061, 1064	
on_label:nn .....	1063, 1065	\l__zrefcheck_close_range_int .	
zrefcheck internal commands:			
\g__zrefcheck_abschap_int .	19, 23, 26	.....	244, 825
\g__zrefcheck_abssec_int	19, 24, 29, 30	\__zrefcheck_do_check:nnn .....	
\g__zrefcheck_auxfile_lblseq_-		.....	19, 495, 505, 1064
prop .....	12, 17, 272, 295, 319	\l__zrefcheck_end_label_required_-	
\__zrefcheck_check_<check>:nn .	16	bool .....	445
\__zrefcheck_check_above:nn .	640	\__zrefcheck_end_lblfmt:n .....	
\__zrefcheck_check_above:nnTF .		. 12, 313, 402, 419, 430, 523, 527,	
.....	666, 677	532, 539, 543, 548, 552, 553, 562,	
\__zrefcheck_check_after:nn .	672	564, 568, 574, 576, 580, 583, 584, 1047	
\__zrefcheck_check_before:nn .	672	\__zrefcheck_get_asint:nnn .	12
\__zrefcheck_check_below:nn .	640	\__zrefcheck_get_astl:nnn .....	12
\__zrefcheck_check_below:nnTF .	687	\g__zrefcheck_id_int .....	
\__zrefcheck_check_chapsafter:nn	839	366, 377, 379, 1037, 1039, 1058, 1060	
\__zrefcheck_check_chapsbefore:nn		\__zrefcheck_int_to_roman:w .	102
.....	839	\l__zrefcheck_integer_bool .....	
\__zrefcheck_check_close:nn .	813	.....	13, 345, 355, 619, 623, 645,
\__zrefcheck_check_close:nnTF .	835	649, 695, 699, 713, 717, 731, 735,	
\__zrefcheck_check_facing:nn .	692	750, 754, 769, 773, 787, 791, 816,	
\__zrefcheck_check_far:nn .	813	820, 842, 846, 860, 864, 877, 881,	
\__zrefcheck_check_lblfmt:n .		895, 899, 912, 916, 930, 936, 952,	
.....	11, 312, 379, 1039, 1060	958, 973, 979, 995, 1001, 1016, 1022	
\__zrefcheck_check_nextchap:nn .	839	\__zrefcheck_is_integer:n .	5, 102
\__zrefcheck_check_nextpage:nn .	692	\__zrefcheck_is_integer:nnTF .	350
\__zrefcheck_check_nextsec:nn .	927	\__zrefcheck_is_integer_rgx:n .	5, 113
\__zrefcheck_check_otherpage:nn	616	\__zrefcheck_is_integer_rgx:nnTF	249
\__zrefcheck_check_pagegap:nn .	692	\l__zrefcheck_lbl_b_int .....	
\__zrefcheck_check_pagesafter:nn	692	.....	612, 933, 939, 955,
		961, 976, 982, 998, 1004, 1019, 1025	
		\l__zrefcheck_lbl_int 612, 620, 626,	
		627, 646, 652, 653, 696, 702, 703,	

714, 720, 721, 732, 738, 739, 751,	\l__zrefcheck_ref_int . . . . .	<a href="#">612</a> ,
757, 758, 770, 776, 777, 788, 798,	621, 626, 628, 647, 652, 654, 697,	
803, 806, 817, 823, 826, 843, 849,	702, 704, 715, 720, 722, 733, 738,	
850, 861, 867, 868, 878, 884, 885,	740, 752, 757, 759, 771, 776, 778,	
896, 902, 903, 913, 919, 920, 931,	789, 796, 798, 801, 803, 807, 818,	
941, 942, 953, 963, 964, 974, 984,	823, 827, 844, 849, 851, 862, 867,	
985, 996, 1006, 1007, 1017, 1027, 1028	879, 884, 886, 897, 902, 914, 919,	
\l__zrefcheck_link_anchor_tl . . .	921, 932, 941, 943, 954, 963, 975,	
. . . . .	984, 986, 997, 1006, 1018, 1027, 1029	
\l__zrefcheck_link_label_tl . . . .	\__zrefcheck_run_checks:nnn . . . .	
. . . . .	. . . . .	<a href="#">18</a> , <a href="#">406</a> , <a href="#">489</a> , <a href="#">1053</a>
\l__zrefcheck_link_star_bool . . .	\c__zrefcheck_single_label_-	
. . . . .	checks_seq . . . . .	<a href="#">17</a> , <a href="#">432</a> , <a href="#">483</a>
\__zrefcheck_message:nnnn <a href="#">53</a> , <a href="#">597</a> , <a href="#">602</a>	\l__zrefcheck_use_hyperref_bool .	
\l__zrefcheck_msglevel_tl . . . <a href="#">55</a> , <a href="#">160</a>	. . . . .	<a href="#">119</a> , <a href="#">146</a> , <a href="#">152</a> , <a href="#">388</a>
\l__zrefcheck_msgonpage_bool <a href="#">205</a> , <a href="#">593</a>	\l__zrefcheck_warn_hyperref_bool	
\l__zrefcheck_onpage_bool . . . . .	. . . . .	<a href="#">119</a> , <a href="#">150</a>
<a href="#">500</a> , <a href="#">514</a> , <a href="#">561</a> , <a href="#">569</a> , <a href="#">575</a> , <a href="#">581</a> , <a href="#">585</a> , <a href="#">594</a>	\__zrefcheck_zcheck:nnnn <a href="#">14</a> , <a href="#">365</a> , <a href="#">372</a>	
\c__zrefcheck_onpage_checks_seq .	\__zrefcheck_zcheck:nnnnn . <a href="#">12</a> , <a href="#">14</a> , <a href="#">19</a>	
. . . . .	\l__zrefcheck_zcheck_checks_seq .	
\l__zrefcheck_passedcheck_bool . .	. . . . .	<a href="#">406</a> , <a href="#">445</a> , <a href="#">482</a> , <a href="#">1054</a>
<a href="#">500</a> , <a href="#">513</a> , <a href="#">520</a> , <a href="#">533</a> , <a href="#">540</a> , <a href="#">549</a> , <a href="#">554</a> , <a href="#">589</a>	\l__zrefcheck_zcheck_end_label_-	
\l__zrefcheck_propval_tl . . . . .	bool . . . . .	<a href="#">399</a> , <a href="#">446</a> , <a href="#">484</a> , <a href="#">1044</a>
. . . . .	\l__zrefcheck_zcheck_labels_seq .	
\l__zrefcheck_ref_b_int . . . . .	. . . . .	<a href="#">366</a> , <a href="#">376</a> , <a href="#">381</a> , <a href="#">405</a> , <a href="#">407</a>
. . . . .	\zrefchecksetup . . . . .	<a href="#">9</a> , <a href="#">270</a>
<a href="#">612</a> , <a href="#">934</a> , <a href="#">939</a> , <a href="#">956</a> ,		
<a href="#">961</a> , <a href="#">977</a> , <a href="#">982</a> , <a href="#">999</a> , <a href="#">1004</a> , <a href="#">1020</a> , <a href="#">1025</a>		