

# The `zref-check` package implementation\*

Gustavo Barros<sup>†</sup>

2022-07-05

## 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 . . . . .	12
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 . . . . .	26
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 v0.3.1, released 2022-07-05.

<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 ltcmd-hooks, 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} {2022-07-05} {0.3.1}
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 \newcounter { zc@abschap }
20 \newcounter { zc@abssec } [ zc@abschap ]
```

(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 { \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 Messages

```

\__zrefcheck_message:nnnn
\__zrefcheck_message:nnnx
50 \cs_new_protected:Npn \__zrefcheck_message:nnnn #1#2#3#4
51 {
52   \use:c { msg_ \l__zrefcheck_msglevel_tl :nnnnn }
53   { zref-check } {#1} {#2} {#3} {#4}
54 }
55 \cs_generate_variant:Nn \__zrefcheck_message:nnnn { nnnx }

(End definition for \__zrefcheck_message:nnnn.)

56 \msg_new:nnn { zref-check } { check-failed }
57 {
58   Check~failed~\msg_line_context:~
59   Failed-check~'#1'~for~label~'#2'~on~page~#3.
60 }
61 \msg_new:nnn { zref-check } { double-check }

```

```

62 {
63   Same-page-check~\msg_line_context:..~
64   Double-check~'#1'~for~label~'#2'~on~page~#3.
65 }

66 \msg_new:nnn { zref-check } { check-missing }
67 { Check~'#1'~not~defined~\msg_line_context:.. }
68 \msg_new:nnn { zref-check } { property-undefined }
69 { Property~'#1'~not~defined~\msg_line_context:.. }
70 \msg_new:nnn { zref-check } { property-not-in-label }
71 { Label~'#1'~has~no~property~'#2'~\msg_line_context:.. }
72 \msg_new:nnn { zref-check } { property-not-integer }
73 { Property~'#1'~for~label~'#2'~not~an~integer~\msg_line_context:.. }

74 \msg_new:nnn { zref-check } { hyperref-preamble-only }
75 {
76   Option~'hyperref'~only~available~in~the~preamble. \iow_newline:
77   Use~the~starred~version~of~'\iow_char:N\zcheck'~instead.
78 }
79 \msg_new:nnn { zref-check } { missing-hyperref }
80 { Missing~'hyperref'~package. \iow_newline: Setting~'hyperref=false'. }
81 \msg_new:nnn { zref-check } { ignore-document-only }
82 {
83   Option~'ignore'~only~available~in~the~document. \iow_newline:
84   Use~option~'msglevel'~instead.
85 }
86 \msg_new:nnn { zref-check } { option-preamble-only }
87 { Option~'#1'~is~preamble-only~\msg_line_context:.. }
88 \msg_new:nnn { zref-check } { closerange-not-positive-integer }
89 {
90   Option~'closerange'~not~a~positive~integer~\msg_line_context:..~
91   Using~default~value.
92 }
93 \msg_new:nnn { zref-check } { labelcmd-undefined }
94 {
95   Control~sequence~named~'#1'~used~in~option~'labelcmd'~is~not~defined.~
96   Using~default~value.
97 }
98 \msg_new:nnn { zref-check } { option-deprecated-with-alternative }
99 {
100   Option~'#1'~has~been~deprecated~\msg_line_context:.\iow_newline:
101   Use~'#2'~instead.
102 }
103 \msg_new:nnn { zref-check } { option-deprecated }
104 { Option~'#1'~has~been~deprecated~\msg_line_context:.. }
105 \msg_new:nnn { zref-check } { load-time-options }
106 {
107   'zref-check'~does~not~accept~load~time~options.~
108   To~configure~package~options,~use~'\iow_char:N\zrefchecksetup'.
109 }

```

## 4.2 Integer testing

`\__zrefcheck_is_integer:n` From <https://tex.stackexchange.com/a/244405> (thanks Enrico Gregorio, aka 'egreg'),  
`\__zrefcheck_int_to_roman:w` 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.

```

110 \cs_new_eq:NN \__zrefcheck_int_to_roman:w \__int_to_roman:w
111 \prg_new_conditional:Npnn \__zrefcheck_is_integer:n #1 { p, T , F , TF }
112 {
113   \tl_if_empty:oTF {#1}
114   { \prg_return_false: }
115   {
116     \tl_if_empty:oTF { \__zrefcheck_int_to_roman:w -0#1 }
117     { \prg_return_true: }
118     { \prg_return_false: }
119   }
120 }
```

(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.

```

121 \prg_new_protected_conditional:Npnn \__zrefcheck_is_integer_rgx:n #1 { TF }
122 {
123   \regex_match:nnTF { \A\d+\Z } {#1}
124   { \prg_return_true: }
125   { \prg_return_false: }
126 }
```

(End definition for `\__zrefcheck_is_integer_rgx:n`.)

## 4.3 Options

### hyperref option

```

\l__zrefcheck_use_hyperref_bool
\l__zrefcheck_warn_hyperref_bool
127 \bool_new:N \l__zrefcheck_use_hyperref_bool
128 \bool_new:N \l__zrefcheck_warn_hyperref_bool
129 \keys_define:nn { zref-check }
130 {
131   hyperref .choice: ,
132   hyperref / auto .code:n =
133   {
134     \bool_set_true:N \l__zrefcheck_use_hyperref_bool
135     \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
136   } ,
```

```

137   hyperref / true .code:n =
138   {
139     \bool_set_true:N \l__zrefcheck_use_hyperref_bool
140     \bool_set_true:N \l__zrefcheck_warn_hyperref_bool
141   } ,
142   hyperref / false .code:n =
143   {
144     \bool_set_false:N \l__zrefcheck_use_hyperref_bool
145     \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
146   } ,
147   hyperref .initial:n = auto ,
148   hyperref .default:n = auto
149 }

```

(End definition for \l\_\_zrefcheck\_use\_hyperref\_bool and \l\_\_zrefcheck\_warn\_hyperref\_bool.)

```

150 \AddToHook { begindocument }
151 {
152   \@ifpackageloaded { hyperref }
153   {
154     \bool_if:NT \l__zrefcheck_use_hyperref_bool
155     { \RequirePackage { zref-hyperref } }
156   }
157   {
158     \bool_if:NT \l__zrefcheck_warn_hyperref_bool
159     { \msg_warning:nn { zref-check } { missing-hyperref } }
160     \bool_set_false:N \l__zrefcheck_use_hyperref_bool
161   }
162   \keys_define:nn { zref-check }
163   {
164     hyperref .code:n =
165     { \msg_warning:nn { zref-check } { hyperref-preamble-only } }
166   }
167 }

```

### msglevel option

\l\_\_zrefcheck\_msglevel\_tl

```

168 \tl_new:N \l__zrefcheck_msglevel_tl
169 \keys_define:nn { zref-check }
170 {
171   msglevel .choice: ,
172   msglevel / warn .code:n =
173   { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } } ,
174   msglevel / info .code:n =
175   { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } } ,
176   msglevel / none .code:n =
177   { \tl_set:Nn \l__zrefcheck_msglevel_tl { none } } ,
178   msglevel / infoifdraft .code:n =
179   {
180     \ifdraft
181     { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
182     { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
183   } ,
184   msglevel / warniffinal .code:n =

```

```

185     {
186       \ifoptionfinal
187       { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
188       { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
189     } ,
190     msglevel / obeydraft .code:n =
191     {
192       % NOTE Option value deprecated in 2021-12-07 for v0.2.2.
193       \msg_warning:nnnn { zref-check }{ option-deprecated-with-alternative }
194       { msglevel=obeydraft } { msglevel=infoifdraft }
195     } ,
196     msglevel / obeyfinal .code:n =
197     {
198       % NOTE Option value deprecated in 2021-12-07 for v0.2.2.
199       \msg_warning:nnnn { zref-check }{ option-deprecated-with-alternative }
200       { msglevel=obeyfinal } { msglevel=warniffinal }
201     } ,
202     msglevel .value_required:n = true ,
203     msglevel .initial:n = warn ,

```

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

```

204     ignore .code:n =
205     { \msg_warning:nn { zref-check } { ignore-document-only } } ,
206     ignore .value_forbidden:n = true
207   }

```

(End definition for `\l__zrefcheck_msglevel_tl`.)

```

208 \AddToHook { begindocument }
209 {
210   \keys_define:nn { zref-check }
211   { ignore .meta:n = { msglevel = none } }
212 }

```

## onpage option

`\l__zrefcheck_msgonpage_bool`

```

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

```

```

231     onpage / msgiffinal .code:n =
232     {
233         \ifoptionfinal
234         { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
235         { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
236     } ,
237     onpage / obeydraft .code:n =
238     {
239         % NOTE Option value deprecated in 2021-12-07 for v0.2.2.
240         \msg_warning:nnnn { zref-check }{ option-deprecated-with-alternative }
241         { onpage=obeydraft } { onpage=labelseqifdraft }
242     } ,
243     onpage / obeyfinal .code:n =
244     {
245         % NOTE Option value deprecated in 2021-12-07 for v0.2.2.
246         \msg_warning:nnnn { zref-check }{ option-deprecated-with-alternative }
247         { onpage=obeyfinal } { onpage=msgiffinal }
248     } ,
249     onpage .value_required:n = true ,
250     onpage .initial:n = labelseq
251 }

```

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

### closerange option

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

```

252 \int_new:N \l__zrefcheck_close_range_int
253 \keys_define:nn { zref-check }
254 {
255     closerange .code:n =
256     {
257         \__zrefcheck_is_integer_rgx:nTF {#1}
258         { \int_set:Nn \l__zrefcheck_close_range_int { \int_eval:n {#1} } }
259         {
260             \msg_warning:nn { zref-check } { closerange-not-positive-integer }
261             \int_set:Nn \l__zrefcheck_close_range_int { 5 }
262         }
263     } ,
264     closerange .value_required:n = true ,
265     closerange .initial:n = 5
266 }

```

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

### labelcmd option

```

267 \keys_define:nn { zref-check }
268 {
269     labelcmd .code:n =
270     {
271         % NOTE Option value deprecated in 2022-02-08 for v0.2.4.
272         \msg_warning:nnn { zref-check }{ option-deprecated }
273         { labelcmd }
274     } ,

```

275 }

## 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.

```
276 \bool_lazy_and:nnT
277 { \tl_if_exist_p:c { opt@ zref-check.sty } }
278 { ! \tl_if_empty_p:c { opt@ zref-check.sty } }
279 { \msg_warning:nn { zref-check } { load-time-options } }
```

\zrefchecksetup Provide \zrefchecksetup.

```
280 \NewDocumentCommand \zrefchecksetup { m }
281 { \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@files\write\@mainaux{...}`. In other words, the catcode change is written in the `.aux` file itself! Indeed, if you inspect the file, you’ll find them there. Besides, there is still the ominous “except with `KeepShorthandsActive`”.

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

For reference: About what are valid characters for use in labels: <https://tex.stackexchange.com/a/18312>. About some problems with active colons: <https://tex.stackexchange.com/a/89470>. About the difference between L3 strings and token lists, see <https://tex.stackexchange.com/a/446381>, in particular Joseph Wright’s comment: “Strings are for data that will never be typeset, for example file names, identifiers, etc.: if the material may be used in typesetting, it should be a token list.” See also moewe’s (CW) answer in the same lines. Which suggests using L3 strings for the reference labels might be a good catch all approach, and possibly more robust. David Carlisle’s comment about `inputenc` and how the strings work is a caveat (see [https://tex.stackexchange.com/q/446123#comment1516961\\_446381](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`

```
282 \prop_new:N \g_zrefcheck_auxfile_lblseq_prop
```

(End definition for `\g_zrefcheck_auxfile_lblseq_prop`.)

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

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

```
285 {
```

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.

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

```
287 \group_begin:
```

```
288 \int_zero:N \l_tmpa_int
```

```
289 \tl_clear:N \l_tmpa_tl
```

```
290 \tl_clear:N \l_tmpb_tl
```

```
291 \bool_set_false:N \l_tmpa_bool
```

```
292 \ior_map_variable:NNn \g_tmpa_ior \l_tmpa_tl
```

```
293 {
```

```

294         \tl_map_variable:NNn \l_tmpa_tl \l_tmpb_tl
295         {
296             \tl_if_eq:NnTF \l_tmpb_tl { \zref@newlabel }
297             {

```

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

```

298         \bool_set_true:N \l_tmpa_bool
299     }
300     {
301         \bool_if:NTF \l_tmpa_bool
302         {
303             \bool_set_false:N \l_tmpa_bool
304             \int_incr:N \l_tmpa_int
305             \prop_gput:Nxx \g__zrefcheck_auxfile_lblseq_prop
306             { \l_tmpb_tl } { \int_use:N \l_tmpa_int }
307         }
308     }

```

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.

```

309         \tl_map_break:
310     }
311 }
312 }
313 }
314 \group_end:
315 \ior_close:N \g_tmpa_ior
316 }

```

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.

```

317 \begingroup
318 \let \@addtoreset \ltx@gobbletwo
319 \newcounter { zrefcheck }
320 \endgroup
321 \setcounter { zrefcheck } { 0 }

```

## 4.6 Label formats

```

\__zrefcheck_check_lblfmt:n \__zrefcheck_check_lblfmt:n {<check id int>}
322 \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>}
323 \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>}
324 \cs_new_protected:Npn \zrefcheck_get_astl:nnn #1#2#3
325 {
326   \tl_clear:N #3
327   \tl_if_eq:nnTF {#2} { lblseq }
328   {
329     \prop_get:NnNF \g__zrefcheck_auxfile_lblseq_prop {#1} #3
330     {
331       \msg_warning:nnnn { zref-check }
332       { property-not-in-label } {#1} {#2}
333     }
334   }
335   {

```

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: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.

```

336     \zref@ifrefundefined {#1}
337     {}
338     {
339         \zref@ifpropundefined {#2}
340         { \msg_warning:nnnn { zref-check } { property-undefined } {#2} }
341         {
342             \zref@ifrefcontainsprop {#1} {#2}
343             {
344                 \tl_set:Nx #3
345                 { \zref@extractdefault {#1} {#2} { \c_empty_tl } }
346             }
347             {
348                 \msg_warning:nnnn
349                 { zref-check } { property-not-in-label } {#1} {#2}
350             }
351         }
352     }
353 }
354 }
```

(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).

```

355 \bool_new:N \l__zrefcheck_integer_bool
```

(End definition for `\l__zrefcheck_integer_bool`.)

`\l__zrefcheck_propval_tl`

```

356 \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}
357 \cs_new_protected:Npn \zrefcheck_get_asint:nnn #1#2#3
358 {
359   \zrefcheck_get_astl:nnn {#1} {#2} { \l__zrefcheck_propval_tl }
360   \__zrefcheck_is_integer:nTF { \l__zrefcheck_propval_tl }
361   {

```

Make it an integer data type.

```

362     \int_set:Nn #3 { \int_eval:n { \l__zrefcheck_propval_tl } }
363   }
364   {
365     \bool_set_false:N \l__zrefcheck_integer_bool
366     \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`.

```

367     { }
368     {
369       \msg_warning:nnnn { zref-check }
370       { property-not-integer } {#2} {#1}
371     }
372   }
373 }

```

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

## 5 User interface

### 5.1 `\zcheck`

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

```

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

```

```

374 \NewDocumentCommand \zcheck { s O { } m m }
375 { \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
376 \seq_new:N \l__zrefcheck_zcheck_labels_seq
377 \int_new:N \g__zrefcheck_id_int
378 \tl_new:N \l__zrefcheck_checkbeg_tl
379 \tl_new:N \l__zrefcheck_link_label_tl
380 \tl_new:N \l__zrefcheck_link_anchor_tl
381 \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 `{\labels}` as first argument, so that it can be protected by `\zref@wrapper@babel` in `\zcheck`. This is the same procedure as the one used in the definition of `\zref` in `zref-user.sty` for protection of `babel` active characters.

```
\__zrefcheck_zcheck:nnnn {\labels} {\langle*\rangle} {\langlechecks/options\rangle} {\langletext\rangle}
```

```
382 \cs_new_protected:Npn \__zrefcheck_zcheck:nnnn #1#2#3#4
383 {
384   \group_begin:
```

Process local options and checks.

```
385   \keys_set:nn { zref-check / zcheck } {#3}
386   \seq_set_from_clist:Nn \l__zrefcheck_zcheck_labels_seq {#1}
```

Names of the labels for this `zcheck` call.

```
387   \int_gincr:N \g__zrefcheck_id_int
388   \tl_set:Nx \l__zrefcheck_checkbeg_tl
389     { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
```

Set `checkbeg` label.

```
390   \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrefcheck-check }
```

Typeset `{\langletext\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.

```
391   \seq_get:NN \l__zrefcheck_zcheck_labels_seq \l__zrefcheck_link_label_tl
392   \bool_set:Nn \l__zrefcheck_link_star_bool {#2}
393   \zref@ifrefundefined { \l__zrefcheck_link_label_tl }
```

If the reference is undefined, just typeset.

```
394     {#4}
395     {
396       \bool_if:nTF
397         {
398           \l__zrefcheck_use_hyperref_bool &&
399           ! \l__zrefcheck_link_star_bool
400         }
401         {
402           \exp_args:Nx \zrefcheck_get_astl:nnn
403             { \l__zrefcheck_link_label_tl }
404             { anchor } { \l__zrefcheck_link_anchor_tl }
405           \hyperlink { \l__zrefcheck_link_anchor_tl } {#4}
406         }
407       {#4}
408     }
```

Set `checkend` label.

```
409   \bool_if:NT \l__zrefcheck_zcheck_end_label_bool
410   {
411     \zref@labelbylist
412       { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
413       { zrefcheck-end }
414   }
```

Check if `\langlelabels\rangle` are defined.

```
415   \seq_map_function:NN \l__zrefcheck_zcheck_labels_seq \zref@refused
```

Run the checks.

```

416     \_zrefcheck_run_checks:nmx { \l_zrefcheck_zcheck_checks_seq }
417     { \l_zrefcheck_zcheck_labels_seq } { \l_zrefcheck_checkbeg_tl }
418     \group_end:
419 }

```

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

## 5.2 Targets

```

\zctarget      \zctarget{<label>}{<text>}

420 \NewDocumentCommand \zctarget { m +m }
421 {

```

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.

```

422     \group_begin:
423     \refstepcounter { zrefcheck }
424     \zref@wrapper@babel \zref@label {#1}
425     #2
426     \tl_if_empty:nF {#2}
427     {
428         \zref@wrapper@babel
429         \zref@labelbylist { \_zrefcheck_end_lblfmt:n {#1} } { zrefcheck-end }
430     }
431     \group_end:
432 }

```

(End definition for `\zctarget`.)

```

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

433 \NewDocumentEnvironment {zcregion} { m }
434 {
435     \refstepcounter { zrefcheck }
436     \zref@wrapper@babel \zref@label {#1}
437 }
438 {
439     \zref@wrapper@babel
440     \zref@labelbylist { \_zrefcheck_end_lblfmt:n {#1} } { zrefcheck-end }
441 }

```

(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 `<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  $\langle reference \rangle$ . That is, the “before” check should return true if the  $\langle label \rangle$  occurs before the “reference”.

The check conditionals are expected to retrieve `zref`’s label information with `\zrefcheck_get_astl:nnn` or `\zrefcheck_get_asint:nnn`. Also, technically speaking, the  $\langle reference \rangle$  argument is also a label, actually a pair of them, as set by `\zcheck`. For the “labels”, any `zref` property in `zref`’s main list is available, the “references” store the properties in the `zrefcheck` list. Besides those, there is also the `lblseq` (fake) property (for either “labels” or “references”), stored in `\g__zrefcheck_auxfile_lblseq_prop`.

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

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

```

442 \seq_const_from_clist:Nn \c__zrefcheck_single_label_checks_seq
443 {
444   thischap ,
445   prevchap ,
446   nextchap ,
447   chapsbefore ,
448   chapsafter ,
449   thissec ,
450   prevsec ,
451   nextsec ,
452   secsbefore ,
453   secsafter ,
454 }
```

(End definition for `\c__zrefcheck_single_label_checks_seq`.)

## 6.2 Setup

`\l__zrefcheck_zcheck_checks_seq`  
`\l__zrefcheck_end_label_required_bool`

```

455 \seq_new:N \l__zrefcheck_zcheck_checks_seq
456 \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`.

```

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

Then we add the checks to it.

```

458 \clist_map_inline:nn
459 {
460   thispage ,
461   prevpage ,
462   nextpage ,
463   facing ,
464   otherpage ,
465   pagegap ,
466   above ,
467   below ,
468   pagesbefore ,
469   ppbefore ,
470   pagesafter ,
471   ppafter ,
472   before ,
473   after ,
474   thischap ,
475   prevchap ,
476   nextchap ,
477   chapsbefore ,
478   chapsafter ,
479   thissec ,
480   prevsec ,
481   nextsec ,
482   secsbefore ,
483   secsafter ,
484   close ,
485   far ,
486 }
487 {
488   \keys_define:nn { zref-check / zcheck }
489   {
490     #1 .code:n =
491     {
492       \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq {#1}
493       \seq_if_in:NnF \c__zrefcheck_single_label_checks_seq {#1}
494       { \bool_set_true:N \l__zrefcheck_zcheck_end_label_bool }
495     } ,
496     #1 .value_forbidden:n = true ,
497   }
498 }

```

### 6.3 Running

```

\__zrefcheck_run_checks:nnn
\__zrefcheck_run_checks:nnn {<checks>} {<labels>} {<reference>}
<checks> are expected to be received as a sequence variable.
499 \cs_new_protected:Npn \__zrefcheck_run_checks:nnn #1#2#3
500 {
501   \group_begin:
502   \seq_map_inline:Nn #2
503   {
504     \seq_map_inline:Nn #1

```

```

505         { \_zrefcheck_do_check:nnn {####1} {##1} {#3} }
506     }
507     \group_end:
508 }
509 \cs_generate_variant:Nn \_zrefcheck_run_checks:nnn { nnn }

```

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

```

\l_zrefcheck_passedcheck_bool
\l_zrefcheck_onpage_bool
\c_zrefcheck_onpage_checks_seq
510 \bool_new:N \l_zrefcheck_passedcheck_bool
511 \bool_new:N \l_zrefcheck_onpage_bool
512 \seq_const_from_clist:Nn \c_zrefcheck_onpage_checks_seq
513 { 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.

```

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

```

```

\_zrefcheck_do_check:nnn
    \_zrefcheck_do_check:nnn {<check>} {<label beg>} {<reference beg>}
515 \cs_new_protected:Npn \_zrefcheck_do_check:nnn #1#2#3
516 {
517     \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.

```

518     \zref@ifrefundefined {#2}
519     {}
520     {
521         \tl_if_empty:nF {#1}
522         {
523             \bool_set_true:N \l_zrefcheck_passedcheck_bool
524             \bool_set_false:N \l_zrefcheck_onpage_bool
525             \cs_if_exist:cTF { \_zrefcheck_check_ #1 :nnF }
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     % ‘‘label beg’’ vs ‘‘reference end’’.
548     \exp_args:Nnno \use:c { __zrefcheck_check_ #1 :nnF }
549     {#2} { \__zrefcheck_end_lblfmt:n {#3} }
550     { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
551     % ‘‘label end’’ \emph{may} have been created by the
552     % target commands.
553     \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
554     {}
555     {
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 \bool_if:NTF \l__zrefcheck_passedcheck_bool
600 {
601     \bool_if:nT
602     {
603         \l__zrefcheck_msgonpage_bool &&
604         \l__zrefcheck_onpage_bool
605     }
606     {
607         \__zrefcheck_message:nnnx { double-check } {#1} {#2}
608         { \zref@extractdefault {#3} {page} {'unknown'} }
609     }
610 }
611 {
612     \__zrefcheck_message:nnnx { check-failed } {#1} {#2}
613     { \zref@extractdefault {#3} {page} {'unknown'} }
614 }
615 }
616 { \msg_warning:nnn { zref-check } { check-missing } {#1} }
617 }
618 }
619 \group_end:
620 }
621 \cs_generate_variant:Nn \__zrefcheck_do_check:nnn { nnV }

```

(End definition for \\_\_zrefcheck\_do\_check:nnn.)

## 6.4 Conditionals

```

\l__zrefcheck_lbl_int More readable scratch variables for the tests.
\l__zrefcheck_ref_int
\l__zrefcheck_lbl_b_int
\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
626 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thispage:nn #1#2 { T , F , TF }
627 {
628     \group_begin:
629         \bool_set_true:N \l__zrefcheck_integer_bool
630         \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
631         \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
632         \bool_lazy_and:nnTF
633         { \l__zrefcheck_integer_bool }
634         {
635             \int_compare_p:nNn

```

```

636             { \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.

```

637             ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
638             ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
639         }
640         { \group_insert_after:N \prg_return_true: }
641         { \group_insert_after:N \prg_return_false: }
642     \group_end:
643 }
644 \prg_new_protected_conditional:Npnn \__zrefcheck_check_otherpage:nn #1#2 { T , F , TF }
645 {
646     \__zrefcheck_check_thispage:nnTF {#1} {#2}
647     { \prg_return_false: }
648     { \prg_return_true: }
649 }

```

(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

```

```

650 \prg_new_protected_conditional:Npnn \__zrefcheck_check_above:nn #1#2 { F , TF }
651 {
652     \group_begin:
653     \__zrefcheck_check_thispage:nnTF {#1} {#2}
654     {
655         \bool_set_true:N \l__zrefcheck_integer_bool
656         \zrefcheck_get_asint:nnn {#1} { lblseq } { \l__zrefcheck_lbl_int }
657         \zrefcheck_get_asint:nnn {#2} { lblseq } { \l__zrefcheck_ref_int }
658         \bool_lazy_and:nnTF
659         { \l__zrefcheck_integer_bool }
660         {
661             \int_compare_p:nNn
662             { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
663             ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
664             ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
665         }
666         { \group_insert_after:N \prg_return_true: }
667         { \group_insert_after:N \prg_return_false: }
668     }
669     { \group_insert_after:N \prg_return_false: }
670 \group_end:
671 }
672 \prg_new_protected_conditional:Npnn \__zrefcheck_check_below:nn #1#2 { F , TF }
673 {
674     \__zrefcheck_check_thispage:nnTF {#1} {#2}
675     {
676         \__zrefcheck_check_above:nnTF {#1} {#2}
677         { \prg_return_false: }
678         { \prg_return_true: }

```

```

679     }
680     { \prg_return_false: }
681 }

```

(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
682 \prg_new_protected_conditional:Npnn \_zrefcheck\_check\_before:nn #1#2 { F }
683 {
684   \_zrefcheck\_check\_pagesbefore:nnTF {#1} {#2}
685   { \prg_return_true: }
686   {
687     \_zrefcheck\_check\_above:nnTF {#1} {#2}
688     { \prg_return_true: }
689     { \prg_return_false: }
690   }
691 }
692 \prg_new_protected_conditional:Npnn \_zrefcheck\_check\_after:nn #1#2 { F }
693 {
694   \_zrefcheck\_check\_pagesafter:nnTF {#1} {#2}
695   { \prg_return_true: }
696   {
697     \_zrefcheck\_check\_below:nnTF {#1} {#2}
698     { \prg_return_true: }
699     { \prg_return_false: }
700   }
701 }

```

(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
702 \prg_new_protected_conditional:Npnn \_zrefcheck\_check\_nextpage:nn #1#2 { F }
703 {
704   \group_begin:
705   \bool_set_true:N \l__zrefcheck_integer_bool
706   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
707   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
708   \bool_lazy_and:nnTF
709   { \l__zrefcheck_integer_bool }
710   {
711     \int_compare_p:nNn
712     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
713     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
714     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
715   }
716   { \group_insert_after:N \prg_return_true: }
717   { \group_insert_after:N \prg_return_false: }
718   \group_end:
719 }
720 \prg_new_protected_conditional:Npnn \_zrefcheck\_check\_prevpage:nn #1#2 { F }

```

```

721 {
722   \group_begin:
723     \bool_set_true:N \l__zrefcheck_integer_bool
724     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
725     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
726     \bool_lazy_and:nnTF
727       { \l__zrefcheck_integer_bool }
728       {
729         \int_compare_p:nNn
730           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
731           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
732           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
733       }
734       { \group_insert_after:N \prg_return_true: }
735       { \group_insert_after:N \prg_return_false: }
736   \group_end:
737 }
738 \prg_new_protected_conditional:Npnn \__zrefcheck_check_pagesbefore:nn #1#2 { F , TF }
739 {
740   \group_begin:
741     \bool_set_true:N \l__zrefcheck_integer_bool
742     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
743     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
744     \bool_lazy_and:nnTF
745       { \l__zrefcheck_integer_bool }
746       {
747         \int_compare_p:nNn
748           { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
749           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
750           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
751       }
752       { \group_insert_after:N \prg_return_true: }
753       { \group_insert_after:N \prg_return_false: }
754   \group_end:
755 }
756 \cs_new_eq:NN \__zrefcheck_check_ppbefore:nnF \__zrefcheck_check_pagesbefore:nnF
757 \prg_new_protected_conditional:Npnn \__zrefcheck_check_pagesafter:nn #1#2 { F , TF }
758 {
759   \group_begin:
760     \bool_set_true:N \l__zrefcheck_integer_bool
761     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
762     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
763     \bool_lazy_and:nnTF
764       { \l__zrefcheck_integer_bool }
765       {
766         \int_compare_p:nNn
767           { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
768           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
769           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
770       }
771       { \group_insert_after:N \prg_return_true: }
772       { \group_insert_after:N \prg_return_false: }
773   \group_end:
774 }

```

```

775 \cs_new_eq:NN \__zrefcheck_check_ppafter:nnF \__zrefcheck_check_pagesafter:nnF
776 \prg_new_protected_conditional:Npnn \__zrefcheck_check_pagegap:nn #1#2 { F }
777 {
778   \group_begin:
779     \bool_set_true:N \l__zrefcheck_integer_bool
780     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
781     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
782     \bool_lazy_and:nnTF
783       { \l__zrefcheck_integer_bool }
784       {
785         \int_compare_p:nNn
786           { \int_abs:n { \l__zrefcheck_lbl_int - \l__zrefcheck_ref_int } } > { 1 } &&
787           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
788           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
789       }
790     { \group_insert_after:N \prg_return_true: }
791     { \group_insert_after:N \prg_return_false: }
792   \group_end:
793 }
794 \prg_new_protected_conditional:Npnn \__zrefcheck_check_facing:nn #1#2 { F }
795 {
796   \group_begin:
797     \bool_set_true:N \l__zrefcheck_integer_bool
798     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
799     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
800     \bool_lazy_and:nnTF
801       { \l__zrefcheck_integer_bool }
802       {

```

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

```

803       \legacy_if_p:n { @twoside } &&

```

Now we test “facing”.

```

804     (
805       (
806         \int_if_odd_p:n { \l__zrefcheck_ref_int } &&
807         \int_compare_p:nNn
808           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 }
809       ) ||
810       (
811         \int_if_even_p:n { \l__zrefcheck_ref_int } &&
812         \int_compare_p:nNn
813           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 }
814       )
815     ) &&
816     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
817     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
818   }
819   { \group_insert_after:N \prg_return_true: }
820   { \group_insert_after:N \prg_return_false: }
821 \group_end:
822 }

```

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

### 6.4.5 Close / Far

```

\__zrefcheck_check_close:nn
\__zrefcheck_check_far:nn
823 \prg_new_protected_conditional:Npnn \__zrefcheck_check_close:nn #1#2 { F , TF }
824 {
825   \group_begin:
826   \bool_set_true:N \l__zrefcheck_integer_bool
827   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
828   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
829   \bool_lazy_and:nnTF
830   { \l__zrefcheck_integer_bool }
831   {
832     \int_compare_p:nNn
833     { \int_abs:n { \l__zrefcheck_lbl_int - \l__zrefcheck_ref_int } }
834     <
835     { \l__zrefcheck_close_range_int + 1 } &&
836     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
837     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
838   }
839   { \group_insert_after:N \prg_return_true: }
840   { \group_insert_after:N \prg_return_false: }
841 \group_end:
842 }
843 \prg_new_protected_conditional:Npnn \__zrefcheck_check_far:nn #1#2 { F }
844 {
845   \__zrefcheck_check_close:nnTF {#1} {#2}
846   { \prg_return_false: }
847   { \prg_return_true: }
848 }

```

(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
849 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thischap:nn #1#2 { F }
850 {
851   \group_begin:
852   \bool_set_true:N \l__zrefcheck_integer_bool
853   \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
854   \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
855   \bool_lazy_and:nnTF
856   { \l__zrefcheck_integer_bool }
857   {
858     \int_compare_p:nNn
859     { \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.

```

860     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
861     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }

```

```

862     }
863     { \group_insert_after:N \prg_return_true: }
864     { \group_insert_after:N \prg_return_false: }
865   \group_end:
866 }
867 \prg_new_protected_conditional:Npnn \__zrefcheck_check_nextchap:nn #1#2 { F }
868 {
869   \group_begin:
870     \bool_set_true:N \l__zrefcheck_integer_bool
871     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
872     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
873     \bool_lazy_and:nnTF
874       { \l__zrefcheck_integer_bool }
875       {
876         \int_compare_p:nNn
877           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
878           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
879       }
880     { \group_insert_after:N \prg_return_true: }
881     { \group_insert_after:N \prg_return_false: }
882   \group_end:
883 }
884 \prg_new_protected_conditional:Npnn \__zrefcheck_check_prevchap:nn #1#2 { F }
885 {
886   \group_begin:
887     \bool_set_true:N \l__zrefcheck_integer_bool
888     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
889     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
890     \bool_lazy_and:nnTF
891       { \l__zrefcheck_integer_bool }
892       {
893         \int_compare_p:nNn
894           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
895           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
896           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
897       }
898     { \group_insert_after:N \prg_return_true: }
899     { \group_insert_after:N \prg_return_false: }
900   \group_end:
901 }
902 \prg_new_protected_conditional:Npnn \__zrefcheck_check_chapsafter:nn #1#2 { F }
903 {
904   \group_begin:
905     \bool_set_true:N \l__zrefcheck_integer_bool
906     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
907     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
908     \bool_lazy_and:nnTF
909       { \l__zrefcheck_integer_bool }
910       {
911         \int_compare_p:nNn
912           { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
913           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
914       }
915     { \group_insert_after:N \prg_return_true: }

```

```

916         { \group_insert_after:N \prg_return_false: }
917     \group_end:
918 }
919 \prg_new_protected_conditional:Npnn \__zrefcheck_check_chapsbefore:nn #1#2 { F }
920 {
921     \group_begin:
922     \bool_set_true:N \l__zrefcheck_integer_bool
923     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
924     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
925     \bool_lazy_and:nnTF
926     { \l__zrefcheck_integer_bool }
927     {
928         \int_compare_p:nNn
929         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
930         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
931         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
932     }
933     { \group_insert_after:N \prg_return_true: }
934     { \group_insert_after:N \prg_return_false: }
935 \group_end:
936 }

```

(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_secsofter:nn
\__zrefcheck_check_secsofter:nn
937 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thissec:nn #1#2 { F }
938 {
939     \group_begin:
940     \bool_set_true:N \l__zrefcheck_integer_bool
941     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
942     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
943     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
944     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
945     \bool_lazy_and:nnTF
946     { \l__zrefcheck_integer_bool }
947     {
948         \int_compare_p:nNn
949         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
950         \int_compare_p:nNn
951         { \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.

```

952         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
953         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
954     }
955     { \group_insert_after:N \prg_return_true: }
956     { \group_insert_after:N \prg_return_false: }
957 \group_end:

```

```

958 }
959 \prg_new_protected_conditional:Npnn \__zrefcheck_check_nextsec:nn #1#2 { F }
960 {
961   \group_begin:
962     \bool_set_true:N \l__zrefcheck_integer_bool
963     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
964     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
965     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
966     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
967     \bool_lazy_and:nnTF
968       { \l__zrefcheck_integer_bool }
969       {
970         \int_compare_p:nNn
971           { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
972         \int_compare_p:nNn
973           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
974         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
975       }
976       { \group_insert_after:N \prg_return_true: }
977       { \group_insert_after:N \prg_return_false: }
978   \group_end:
979 }
980 \prg_new_protected_conditional:Npnn \__zrefcheck_check_prevsec:nn #1#2 { F }
981 {
982   \group_begin:
983     \bool_set_true:N \l__zrefcheck_integer_bool
984     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
985     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
986     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
987     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
988     \bool_lazy_and:nnTF
989       { \l__zrefcheck_integer_bool }
990       {
991         \int_compare_p:nNn
992           { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
993         \int_compare_p:nNn
994           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
995         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
996         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
997       }
998       { \group_insert_after:N \prg_return_true: }
999       { \group_insert_after:N \prg_return_false: }
1000   \group_end:
1001 }
1002 \prg_new_protected_conditional:Npnn \__zrefcheck_check_secsafter:nn #1#2 { F }
1003 {
1004   \group_begin:
1005     \bool_set_true:N \l__zrefcheck_integer_bool
1006     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
1007     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
1008     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
1009     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
1010     \bool_lazy_and:nnTF
1011       { \l__zrefcheck_integer_bool }

```

```

1012     {
1013         \int_compare_p:nNn
1014         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
1015         \int_compare_p:nNn
1016         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
1017         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
1018     }
1019     { \group_insert_after:N \prg_return_true: }
1020     { \group_insert_after:N \prg_return_false: }
1021 \group_end:
1022 }
1023 \prg_new_protected_conditional:Npnn \__zrefcheck_check_secsbefore:nn #1#2 { F }
1024 {
1025     \group_begin:
1026     \bool_set_true:N \l__zrefcheck_integer_bool
1027     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
1028     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
1029     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
1030     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
1031     \bool_lazy_and:nnTF
1032     { \l__zrefcheck_integer_bool }
1033     {
1034         \int_compare_p:nNn
1035         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
1036         \int_compare_p:nNn
1037         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
1038         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
1039         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
1040     }
1041     { \group_insert_after:N \prg_return_true: }
1042     { \group_insert_after:N \prg_return_false: }
1043 \group_end:
1044 }

```

(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
1045 \cs_new_protected:Npn \zrefcheck_zceref_beg_label:
1046 {
1047     \int_gincr:N \g__zrefcheck_id_int
1048     \tl_set:Nx \l__zrefcheck_checkbeg_tl
1049     { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
1050     \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrefcheck-check }
1051 }
1052 \cs_new_protected:Npn \zrefcheck_zceref_end_label_maybe:

```

```

1053 {
1054   \bool_if:NT \l__zrefcheck_zcheck_end_label_bool
1055   {
1056     \zref@labelbylist
1057     { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
1058     { zrefcheck-end }
1059   }
1060 }
1061 \cs_new_protected:Npn \zrefcheck_zcref_run_checks_on_labels:n #1
1062 {
1063   \__zrefcheck_run_checks:nnx
1064   { \l__zrefcheck_zcheck_checks_seq } {#1} { \l__zrefcheck_checkbeg_tl }
1065 }

```

(End 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

```

```

1066 \cs_new_protected:Npn \zrefcheck_zrefvario_label:
1067 {
1068   \int_gincr:N \g__zrefcheck_id_int
1069   \tl_set:Nx \l__zrefcheck_checkbeg_tl
1070   { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
1071   \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrefcheck-zrefvario }
1072 }
1073 \cs_new_protected:Npn \zrefcheck_zrefvario_run_check_on_label:nn #1#2
1074 { \__zrefcheck_do_check:nnV {#1} {#2} \l__zrefcheck_checkbeg_tl }
1075 \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 ??.)

```

1076 \</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>	<u>77</u> , <u>108</u>
A	
<code>\A</code>	<u>123</u>
<code>\AddToHook</code>	<u>21</u> , <u>25</u> , <u>150</u> , <u>208</u>
B	
<code>\begingroup</code>	<u>317</u>
bool commands:	
<code>\bool_if:N</code>	<u>154</u> , <u>158</u> , <u>301</u> , <u>409</u> , <u>599</u> , <u>1054</u>
<code>\bool_if:nTF</code>	<u>396</u> , <u>601</u>
<code>\bool_lazy_and:nnTF</code>	<u>13</u> , <u>276</u> , <u>632</u> , <u>658</u> , <u>708</u> , <u>726</u> , <u>744</u> , <u>763</u> , <u>782</u> , <u>800</u> , <u>829</u> , <u>855</u> , <u>873</u> , <u>890</u> , <u>908</u> , <u>925</u> , <u>945</u> , <u>967</u> , <u>988</u> , <u>1010</u> , <u>1031</u>
<code>\bool_new:N</code>	<u>127</u> , <u>128</u> , <u>213</u> , <u>355</u> , <u>381</u> , <u>456</u> , <u>510</u> , <u>511</u>
<code>\bool_set:Nn</code>	<u>392</u>
<code>\bool_set_false:N</code>	<u>135</u> , <u>144</u> , <u>145</u> , <u>160</u> , <u>219</u> , <u>228</u> , <u>235</u> , <u>291</u> , <u>303</u> , <u>365</u> , <u>524</u> , <u>530</u> , <u>543</u> , <u>550</u> , <u>559</u> , <u>564</u>

<code>\bool_set_true:N</code>	134, 139, 140, 223, 229, 234, 298, 494, 523, 571, 579, 585, 591, 595, 629, 655, 705, 723, 741, 760, 779, 797, 826, 852, 870, 887, 905, 922, 940, 962, 983, 1005, 1026	915, 916, 933, 934, 955, 956, 976, 977, 998, 999, 1019, 1020, 1041, 1042
<code>\l_tmpa_bool</code>	11, 291, 298, 301, 303	
<b>C</b>		
<code>\catcode</code>	11	
<code>\chapter</code>	2, 26	
clist commands:		
<code>\clist_map_inline:nn</code>	458	
cs commands:		
<code>\cs_generate_variant:Nn</code>	55, 509, 514, 621, 1075	
<code>\cs_if_exist:NTF</code>	525	
<code>\cs_new:Npn</code>	322, 323	
<code>\cs_new_eq:NN</code>	110, 756, 775	
<code>\cs_new_protected:Npn</code>	50, 324, 357, 382, 499, 515, 1045, 1052, 1061, 1066, 1073	
<b>D</b>		
<code>\d</code>	123	
<b>E</b>		
<code>\emph</code>	531, 535, 551	
<code>\endgroup</code>	320	
<code>\endinput</code>	12	
exp commands:		
<code>\exp_args:Nnno</code>	514, 548	
<code>\exp_args:Nno</code>	541, 557	
<code>\exp_args:Nnoo</code>	561	
<code>\exp_args:Nx</code>	402	
<b>F</b>		
file commands:		
<code>\file_if_exist:nTF</code>	284	
<code>\fmtversion</code>	3	
<b>G</b>		
group commands:		
<code>\group_begin:</code>	287, 384, 422, 501, 517, 628, 652, 704, 722, 740, 759, 778, 796, 825, 851, 869, 886, 904, 921, 939, 961, 982, 1004, 1025	
<code>\group_end:</code>	314, 418, 431, 507, 619, 642, 670, 718, 736, 754, 773, 792, 821, 841, 865, 882, 900, 917, 935, 957, 978, 1000, 1021, 1043	
<code>\group_insert_after:N</code>	640, 641, 666, 667, 669, 716, 717, 734, 735, 752, 753, 771, 772, 790, 791, 819, 820, 839, 840, 863, 864, 880, 881, 898, 899,	
		915, 916, 933, 934, 955, 956, 976, 977, 998, 999, 1019, 1020, 1041, 1042
<b>H</b>		
<code>\hyperlink</code>	14, 405	
<b>I</b>		
<code>\ifdraft</code>	180, 227	
<code>\IfFormatAtLeastTF</code>	3, 4	
<code>\ifoptionfinal</code>	186, 233	
int commands:		
<code>\int_abs:n</code>	786, 833	
<code>\int_compare_p:nNn</code>	635, 637, 638, 661, 663, 664, 711, 713, 714, 729, 731, 732, 747, 749, 750, 766, 768, 769, 785, 787, 788, 807, 812, 816, 817, 832, 836, 837, 858, 860, 861, 876, 878, 893, 895, 896, 911, 913, 928, 930, 931, 948, 950, 952, 953, 970, 972, 974, 991, 993, 995, 996, 1013, 1015, 1017, 1034, 1036, 1038, 1039	
<code>\int_eval:n</code>	258, 362	
<code>\int_gincr:N</code>	387, 1047, 1068	
<code>\int_if_even_p:n</code>	811	
<code>\int_if_odd_p:n</code>	806	
<code>\int_incr:N</code>	304	
<code>\int_new:N</code>	252, 377, 622, 623, 624, 625	
<code>\int_set:Nn</code>	258, 261, 362	
<code>\int_use:N</code>	23, 27, 306, 322	
<code>\int_zero:N</code>	288	
<code>\l_tmpa_int</code>	288, 304, 306	
int internal commands:		
<code>\__int_to_roman:w</code>	5, 110	
ior commands:		
<code>\ior_close:N</code>	315	
<code>\ior_map_variable:NNn</code>	292	
<code>\ior_open:Nn</code>	286	
<code>\g_tmpa_ior</code>	286, 292, 315	
iow commands:		
<code>\iow_char:N</code>	77, 108	
<code>\iow_newline:</code>	76, 80, 83, 100	
<b>K</b>		
keys commands:		
<code>\keys_define:nn</code>	129, 162, 169, 210, 214, 253, 267, 457, 488	
<code>\keys_set:nn</code>	281, 385	
<b>L</b>		
<code>\label</code>	11	
legacy commands:		
<code>\legacy_if_p:n</code>	803	
<code>\let</code>	318	

<b>M</b>		
<code>\MessageBreak</code>	10	<code>\seq_get:NN</code> 391
msg commands:		<code>\seq_if_in:NnTF</code> 493, 567
<code>\msg_line_context:</code>	58, 63, 67, 69, 71, 73, 87, 90, 100, 104	<code>\seq_map_function:NN</code> 415
<code>\msg_new:nnn</code>	56, 61, 66, 68, 70, 72, 74, 79, 81, 86, 88, 93, 98, 103, 105	<code>\seq_map_inline:Nn</code> 502, 504
<code>\msg_warning:nn</code>	159, 165, 205, 260, 279	<code>\seq_new:N</code> 376, 455
<code>\msg_warning:nnn</code>	272, 616	<code>\seq_put_right:Nn</code> 492
<code>\msg_warning:nnnn</code>	193, 199, 240, 246, 331, 340, 348, 369	<code>\seq_set_from_clist:Nn</code> 386
		<code>\setcounter</code> 321
		<code>\stepcounter</code> 22, 26
		sys commands:
		<code>\c_sys_jobname_str</code> 283
<b>N</b>		<b>T</b>
<code>\newcounter</code>	19, 20, 319	$\TeX$ and $\LaTeX$ 2 $\epsilon$ commands:
<code>\NewDocumentCommand</code>	280, 374, 420	<code>\@addtoreset</code> 318
<code>\NewDocumentEnvironment</code>	433	<code>\@currentlabel</code> 16
<b>P</b>		<code>\@ifl@t@r</code> 3
<code>\PackageError</code>	7	<code>\@ifpackageloaded</code> 152
<code>\pageref</code>	11	<code>\@newl@bel</code> 9
prg commands:		<code>\c@z@abschap</code> 23
<code>\prg_new_conditional:Npnn</code>	111	<code>\c@z@abssec</code> 27
<code>\prg_new_protected_conditional:Npnn</code>	16, 121, 626, 644, 650, 672, 682, 692, 702, 720, 738, 757, 776, 794, 823, 843, 849, 867, 884, 902, 919, 937, 959, 980, 1002, 1023	<code>\ltx@gobbletwo</code> 318
<code>\prg_return_false:</code>	17, 114, 118, 125, 641, 647, 667, 669, 677, 680, 689, 699, 717, 735, 753, 772, 791, 820, 840, 846, 864, 881, 899, 916, 934, 956, 977, 999, 1020, 1042	<code>\zref@addprop</code> 17, 24, 28
<code>\prg_return_true:</code>	17, 117, 124, 640, 648, 666, 678, 685, 688, 695, 698, 716, 734, 752, 771, 790, 819, 839, 847, 863, 880, 898, 915, 933, 955, 976, 998, 1019, 1041	<code>\zref@addprops</code> 30, 38, 45
prop commands:		<code>\zref@extractdefault</code> 12, 345, 608, 613
<code>\prop_get:NnNTF</code>	329	<code>\zref@ifpropundefined</code> 339
<code>\prop_gput:Nnn</code>	305	<code>\zref@ifrefcontainsprop</code> 342
<code>\prop_new:N</code>	282	<code>\zref@ifrefundefined</code> 13, 336, 366, 393, 518, 533, 553, 572, 574, 586
<code>\providecommand</code>	3	<code>\ZREF@label</code> 11
<code>\ProvidesExplPackage</code>	14	<code>\zref@label</code> 11, 424, 436
<b>R</b>		<code>\zref@labelbylist</code> 390, 411, 429, 440, 1050, 1056, 1071
<code>\refstepcounter</code>	12, 16, 423, 435	<code>\ZREF@mainlist</code> 24, 28
regex commands:		<code>\zref@newlabel</code> 9–11, 296
<code>\regex_match:nnTF</code>	123	<code>\zref@newlist</code> 29, 37, 44
<code>\RequirePackage</code>	16, 17, 18, 155	<code>\zref@newprop</code> 17, 23, 27
<code>\romannumeral</code>	5	<code>\zref@refused</code> 13, 415
<b>S</b>		<code>\zref@require@unique</code> 12
<code>\section</code>	28	<code>\zref@wrapper@babel</code> 10, 15, 375, 424, 428, 436, 439
seq commands:		tl commands:
<code>\seq_const_from_clist:Nn</code>	442, 512	<code>\c_empty_tl</code> 12, 345
		<code>\tl_clear:N</code> 12, 289, 290, 326
		<code>\tl_gset:Nn</code> 283
		<code>\tl_if_blank:nTF</code> 5
		<code>\tl_if_empty:nTF</code> 5, 113, 116, 426, 521
		<code>\tl_if_empty_p:N</code> 278
		<code>\tl_if_eq:NnTF</code> 296
		<code>\tl_if_eq:nnTF</code> 327
		<code>\tl_if_exist_p:N</code> 277
		<code>\tl_map_break:</code> 309
		<code>\tl_map_variable:NNn</code> 294
		<code>\tl_new:N</code> 168, 356, 378, 379, 380

\tl_set:Nn	173, 175, 177, 181, 182, 187, 188, 344, 388, 1048, 1069
\g_tmpa_tl	283, 284, 286
\l_tmpa_tl	289, 292, 294
\l_tmpb_tl	290, 294, 296, 306
<b>U</b>	
use commands:	
\use:N	52, 528, 541, 548, 557, 561
<b>Z</b>	
\Z	123
\zcheck	14, 15, 17, 374
\zcref	30
zcregion	433
\zctarget	16, 420
\zref	15
zrefcheck commands:	
\zrefcheck_get_asint:nnn	13, 14, 17, 357, 357, 630, 631, 656, 657, 706, 707, 724, 725, 742, 743, 761, 762, 780, 781, 798, 799, 827, 828, 853, 854, 871, 872, 888, 889, 906, 907, 923, 924, 941, 942, 943, 944, 963, 964, 965, 966, 984, 985, 986, 987, 1006, 1007, 1008, 1009, 1027, 1028, 1029, 1030
\zrefcheck_get_astl:nnn	12, 13, 17, 324, 324, 359, 402
\zrefcheck_zcref_beg_label:	1045, 1045
\zrefcheck_zcref_end_label_- maybe:	1045, 1052
\zrefcheck_zcref_run_checks_on_- labels:n	1045, 1061
\zrefcheck_zrefvario_label:	1066, 1066
\zrefcheck_zrefvario_run_check_- on_label:n	1066
\zrefcheck_zrefvario_run_check_- on_label:nn	1073, 1075
zrefcheck internal commands:	
\g__zrefcheck_abschap_int	19
\g__zrefcheck_abssec_int	19
\g__zrefcheck_auxfile_lblseq_- prop	12, 17, 282, 305, 329
\__zrefcheck_check_(check):nn	16
\__zrefcheck_check_above:nn	650, 650
\__zrefcheck_check_above:nnTF	676, 687
\__zrefcheck_check_after:nn	682, 692
\__zrefcheck_check_before:nn	682, 682
\__zrefcheck_check_below:nn	650, 672
\__zrefcheck_check_below:nnTF	697
\__zrefcheck_check_chapsafter:nn	849, 902
\__zrefcheck_check_chapsbefore:nn	849, 919
\__zrefcheck_check_close:nn	823, 823
\__zrefcheck_check_close:nnTF	845
\__zrefcheck_check_facing:nn	702, 794
\__zrefcheck_check_far:nn	823, 843
\__zrefcheck_check_lblfmt:n	12, 322, 322, 389, 1049, 1070
\__zrefcheck_check_nextchap:nn	849, 867
\__zrefcheck_check_nextpage:nn	702, 702
\__zrefcheck_check_nextsec:nn	937, 959
\__zrefcheck_check_otherpage:nn	626, 644
\__zrefcheck_check_pagegap:nn	702, 776
\__zrefcheck_check_pagesafter:nn	702, 757
\__zrefcheck_check_pagesafter:nnTF	694, 775
\__zrefcheck_check_pagesbefore:nn	702, 738
\__zrefcheck_check_pagesbefore:nnTF	684, 756
\__zrefcheck_check_ppafter:nn	702
\__zrefcheck_check_ppafter:nnTF	775
\__zrefcheck_check_ppbefore:nn	702
\__zrefcheck_check_ppbefore:nnTF	756
\__zrefcheck_check_prevchap:nn	849, 884
\__zrefcheck_check_prevpage:nn	702, 720
\__zrefcheck_check_prevsec:nn	937, 980
\__zrefcheck_check_secsafter:nn	937, 1002
\__zrefcheck_check_secsbefore:nn	937, 1023
\__zrefcheck_check_thischap:nn	849, 849
\__zrefcheck_check_thispage:nn	626, 626
\__zrefcheck_check_thispage:nnTF	569, 577, 583, 589, 592, 646, 653, 674
\__zrefcheck_check_thissec:nn	937, 937
\l__zrefcheck_checkbeg_tl	376, 388, 390, 412, 417, 1048, 1050, 1057, 1064, 1069, 1071, 1074

\l__zrefcheck_close_range_int ...	\l__zrefcheck_link_star_bool ...
..... 252, 835	..... 376, 392, 399
\__zrefcheck_do_check:nnn .....	\__zrefcheck_message:nnnn .....
..... 19, 505, 515, 515, 621, 1074	..... 50, 50, 55, 607, 612
\l__zrefcheck_end_label_required_	\l__zrefcheck_msglevel_tl ... 52, 168
bool .....	\l__zrefcheck_msgonpage_bool 213, 603
455	\l__zrefcheck_onpage_bool .....
\__zrefcheck_end_lblfmt:n ... 12,	510, 524, 571, 579, 585, 591, 595, 604
323, 323, 412, 429, 440, 533, 537,	\c__zrefcheck_onpage_checks_seq .
542, 549, 553, 558, 562, 563, 572,	..... 510, 567
574, 578, 584, 586, 590, 593, 594, 1057	\l__zrefcheck_passedcheck_bool ..
\__zrefcheck_get_asint:nnn .....	510, 523, 530, 543, 550, 559, 564, 599
12	\l__zrefcheck_propval_tl .....
\__zrefcheck_get_astl:nnn .....	356, 359, 360, 362
12	\l__zrefcheck_ref_b_int .....
\g__zrefcheck_id_int .....	622, 944, 949, 966,
376, 387, 389, 1047, 1049, 1068, 1070	971, 987, 992, 1009, 1014, 1030, 1035
\__zrefcheck_int_to_roman:w ...	\l__zrefcheck_ref_int .....
..... 110, 110, 116	622,
\l__zrefcheck_integer_bool .....	631, 636, 638, 657, 662, 664, 707,
..... 13, 355, 365, 629, 633, 655,	712, 714, 725, 730, 732, 743, 748,
659, 705, 709, 723, 727, 741, 745,	750, 762, 767, 769, 781, 786, 788,
760, 764, 779, 783, 797, 801, 826,	799, 806, 808, 811, 813, 817, 828,
830, 852, 856, 870, 874, 887, 891,	833, 837, 854, 859, 861, 872, 877,
905, 909, 922, 926, 940, 946, 962,	889, 894, 896, 907, 912, 924, 929,
968, 983, 989, 1005, 1011, 1026, 1032	931, 942, 951, 953, 964, 973, 985,
\__zrefcheck_is_integer:n 5, 110, 111	994, 996, 1007, 1016, 1028, 1037, 1039
\__zrefcheck_is_integer:nTF ... 360	\__zrefcheck_run_checks:nnn ...
\__zrefcheck_is_integer_rgx:n ...	..... 18, 416, 499, 499, 509, 1063
..... 5, 121, 121	\c__zrefcheck_single_label_
\__zrefcheck_is_integer_rgx:nTF 257	checks_seq .....
\l__zrefcheck_lbl_b_int .....	17, 442, 493
622, 943, 949, 965,	\l__zrefcheck_use_hyperref_bool .
971, 986, 992, 1008, 1014, 1029, 1035	..... 127, 154, 160, 398
\l__zrefcheck_lbl_int 622, 630, 636,	\l__zrefcheck_warn_hyperref_bool
637, 656, 662, 663, 706, 712, 713,	..... 127, 158
724, 730, 731, 742, 748, 749, 761,	\__zrefcheck_zcheck:nnnn .....
767, 768, 780, 786, 787, 798, 808,	..... 15, 375, 382, 382
813, 816, 827, 833, 836, 853, 859,	\__zrefcheck_zcheck:nnnnn 13, 14, 19
860, 871, 877, 878, 888, 894, 895,	\l__zrefcheck_zcheck_checks_seq .
906, 912, 913, 923, 929, 930, 941,	..... 416, 455, 492, 1064
951, 952, 963, 973, 974, 984, 994,	\l__zrefcheck_zcheck_end_label_
995, 1006, 1016, 1017, 1027, 1037, 1038	bool .....
\l__zrefcheck_link_anchor_tl ...	409, 456, 494, 1054
..... 376, 404, 405	\l__zrefcheck_zcheck_labels_seq .
\l__zrefcheck_link_label_tl ...	..... 376, 386, 391, 415, 417
..... 376, 391, 393, 403	\zrefchecksetup .....
	9, 280