

The `zref-check` package implementation*

Gustavo Barros[†]

2021-12-07

Contents

1	Initial setup	2
2	Dependencies	2
3	<code>zref</code> setup	2
4	Plumbing	3
4.1	Messages	3
4.2	Integer testing	4
4.3	Options	5
4.4	Position on page	9
4.5	Counter	12
4.6	Label formats	12
4.7	Property values	13
5	User interface	15
5.1	<code>\zcheck</code>	15
5.2	Targets	16
6	Checks	17
6.1	Single label checks	18
6.2	Setup	18
6.3	Running	19
6.4	Conditionals	22
6.4.1	This page	22
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
7	<code>zref-clever</code> integration	31

*This file describes v0.2.2, released 2021-12-07.

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

1 Initial setup

Start the DocStrip guards.

```
1 <*package>
```

Identify the internal prefix (L^AT_EX3 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^AT_EX kernel.

```
3 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
```

```
4 \IfFormatAtLeastTF{2021-06-01}
```

```
5 {}
```

```
6 {%
```

```
7   \PackageError{zref-check}{LaTeX kernel too old}
```

```
8   {%
```

```
9     'zref-check' requires a LaTeX kernel newer than 2021-06-01.%
```

```
10    \MessageBreak Loading will abort!%
```

```
11    }%
```

```
12  \endinput
```

```
13 }%
```

Identify the package.

```
14 \ProvidesExplPackage {zref-check} {2021-12-07} {0.2.2}
```

```
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
\g__zrefcheck_abssec_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_zero:N \g__zrefcheck_abssec_int
```

```
25 }
```

```
26 \zref@newprop { zc@abschap } [0] { \int_use:N \g__zrefcheck_abschap_int }
```

```
27 \zref@addprop \ZREF@mainlist { zc@abschap }
```

```

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

```

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.

```

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-target }
41 \zref@addprops { zrefcheck-target }
42 {
43   page , % so that \zpageref can refer to it
44   abspage ,
45   zc@abschap ,
46   zc@abssec
47 }
48 \zref@newlist { zrefcheck-end }
49 \zref@addprops { zrefcheck-end }
50 {
51   abspage ,
52   zc@abschap ,
53   zc@abssec
54 }

```

4 Plumbing

4.1 Messages

```

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

```

(End definition for __zrefcheck_message:nnnn.)

```

61 \msg_new:nnn { zref-check } { check-failed }
62 { Failed-check~'#1'~for~label~'#2'~on~page~#3~\msg_line_context:. }
63 \msg_new:nnn { zref-check } { double-check }
64 { Double-check~'#1'~for~label~'#2'~on~page~#3~\msg_line_context:. }
65 \msg_new:nnn { zref-check } { check-missing }
66 { Check~'#1'~not~defined~\msg_line_context:. }
67 \msg_new:nnn { zref-check } { property-undefined }
68 { Property~'#1'~not~defined~\msg_line_context:. }

```

```

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

```

4.2 Integer testing

`__zrefcheck_is_integer:n`
`__zrefcheck_int_to_roman:w`

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

```

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     {
148       \RequirePackage { zref-hyperref }
149       \zref@addprop { zrefcheck-target } { anchor }
150     }
151   }
152   {
153     \bool_if:NT \l__zrefcheck_warn_hyperref_bool
154     { \msg_warning:nn { zref-check } { missing-hyperref } }
155     \bool_set_false:N \l__zrefcheck_use_hyperref_bool
156   }
157   \keys_define:nn { zref-check }
158   {
159     hyperref .code:n =
160     { \msg_warning:nn { zref-check } { hyperref-preamble-only } }
161   }
162 }

```

msglevel option

\l__zrefcheck_msglevel_tl

```

163 \tl_new:N \l__zrefcheck_msglevel_tl
164 \keys_define:nn { zref-check }
165 {
166   msglevel .choice: ,
167   msglevel / warn .code:n =
168   { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } } ,
169   msglevel / info .code:n =
170   { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } } ,
171   msglevel / none .code:n =
172   { \tl_set:Nn \l__zrefcheck_msglevel_tl { none } } ,
173   msglevel / infoifdraft .code:n =
174   {
175     \ifdraft
176     { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
177     { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
178   } ,
179   msglevel / warniffinal .code:n =
180   {
181     \ifoptionfinal
182     { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
183     { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
184   } ,
185   msglevel / obeydraft .code:n =
186   {
187     % NOTE Option value deprecated in 2021-12-07 for v0.2.2.
188     \msg_warning:nnnn { zref-check } { option-deprecated }
189     { msglevel=obeydraft } { msglevel=infoifdraft }

```

```

190     } ,
191     msglevel / obeyfinal .code:n =
192     {
193         % NOTE Option value deprecated in 2021-12-07 for v0.2.2.
194         \msg_warning:nnnn { zref-check }{ option-deprecated }
195         { msglevel=obeyfinal } { msglevel=warniffinal }
196     } ,
197     msglevel .value_required:n = true ,
198     msglevel .initial:n = warn ,

```

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

```

199     ignore .code:n =
200     { \msg_warning:nn { zref-check } { ignore-document-only } } ,
201     ignore .value_forbidden:n = true
202 }

```

(End definition for `\l__zrefcheck_msglevel_tl`.)

```

203 \AddToHook { begindocument }
204 {
205     \keys_define:nn { zref-check }
206     { ignore .meta:n = { msglevel = none } }
207 }

```

onpage option

`\l__zrefcheck_msgonpage_bool`

```

208 \bool_new:N \l__zrefcheck_msgonpage_bool
209 \keys_define:nn { zref-check }
210 {
211     onpage .choice: ,
212     onpage / labelseq .code:n =
213     {
214         \bool_set_false:N \l__zrefcheck_msgonpage_bool
215     } ,
216     onpage / msg .code:n =
217     {
218         \bool_set_true:N \l__zrefcheck_msgonpage_bool
219     } ,
220     onpage / labelseqifdraft .code:n =
221     {
222         \ifdraft
223         { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
224         { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
225     } ,
226     onpage / msgiffinal .code:n =
227     {
228         \ifoptionfinal
229         { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
230         { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
231     } ,
232     onpage / obeydraft .code:n =
233     {
234         % NOTE Option value deprecated in 2021-12-07 for v0.2.2.
235         \msg_warning:nnnn { zref-check }{ option-deprecated }

```

```

236         { onpage=obeydraft } { onpage=labelseqifdraft }
237     } ,
238     onpage / obeyfinal .code:n =
239     {
240         % NOTE Option value deprecated in 2021-12-07 for v0.2.2.
241         \msg_warning:nnnn { zref-check }{ option-deprecated }
242         { onpage=obeyfinal } { onpage=msgiffinal }
243     } ,
244     onpage .value_required:n = true ,
245     onpage .initial:n = labelseq
246 }

```

(End definition for \l__zrefcheck_msgonpage_bool.)

closerange option

\l__zrefcheck_close_range_int

```

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

```

(End definition for \l__zrefcheck_close_range_int.)

labelcmd option

\l__zrefcheck_target_label_tl

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

```

262 \tl_new:N \l__zrefcheck_target_label_tl
263 \bool_new:N \l__zrefcheck_target_label_bool
264 \keys_define:nn { zref-check }
265 {
266     labelcmd .code:n =
267     {
268         \tl_set:Nn \l__zrefcheck_target_label_tl {#1}
269         \bool_set_true:N \l__zrefcheck_target_label_bool
270     } ,
271     labelcmd .value_required:n = true ,
272 }

```

(End definition for \l__zrefcheck_target_label_tl.)

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

```

273 \cs_new_protected:Npn \__zrefcheck_target_label:n #1
274 { \zref@labelbylist {#1} { zrefcheck-target } }

(End definition for \__zrefcheck_target_label:n.)

275 \AddToHook { begindocument }
276 {
277   \bool_if:NT \l__zrefcheck_target_label_bool
278   {
279     \tl_if_blank:VT \l__zrefcheck_target_label_tl
280     { \tl_clear:N \l__zrefcheck_target_label_tl }
281     \cs_if_exist:cTF { \l__zrefcheck_target_label_tl }
282     {
283       \cs_set_protected:Npx \__zrefcheck_target_label:n #1
284       {
285         \exp_not:o
286         { \cs:w \l__zrefcheck_target_label_tl \cs_end: }
287         {#1}
288       }
289     }
290     {
291       \exp_args:Nnno \msg_warning:nnn { zref-check }
292       { labelcmd-undefined } { \l__zrefcheck_target_label_tl }
293     }
294   }
295   \keys_define:nn { zref-check }
296   {
297     labelcmd .code:n =
298     {
299       \msg_warning:nnn { zref-check }
300       { option-preamble-only } { labelcmd }
301     }
302   }
303 }

```

Package options

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

```

304 \RequirePackage { l3keys2e }
305 \ProcessKeysOptions { zref-check }

```

`\zrefchecksetup` Provide `\zrefchecksetup`.

```

306 \NewDocumentCommand \zrefchecksetup { m }
307 { \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, 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`

```
308 \prop_new:N \g_zrefcheck_auxfile_lblseq_prop
```

(End definition for `\g_zrefcheck_auxfile_lblseq_prop`.)

```
309 \tl_set:Nn \g_tmpa_tl { \c_sys_jobname_str .aux }
```

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

```
311 {
```

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.

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

```
313 \group_begin:
```

```
314 \int_zero:N \l_tmpa_int
```

```
315 \tl_clear:N \l_tmpa_tl
```

```
316 \tl_clear:N \l_tmpb_tl
```

```
317 \bool_set_false:N \l_tmpa_bool
```

```
318 \ior_map_variable:NNn \g_tmpa_ior \l_tmpa_tl
```

```
319 {
```

```
320 \tl_map_variable:NNn \l_tmpa_tl \l_tmpb_tl
```

```
321 {
```

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

```
323 {
```

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

```
324 \bool_set_true:N \l_tmpa_bool
```

```
325 }
```

```
326 {
```

```
327 \bool_if:NnTF \l_tmpa_bool
```

```
328 {
```

```
329 \bool_set_false:N \l_tmpa_bool
```

```
330 \int_incr:N \l_tmpa_int
```

```
331 \prop_gput:Nxx \g_zrefcheck_auxfile_lblseq_prop
```

```
332 { \l_tmpb_tl } { \int_use:N \l_tmpa_int }
```

```
333 }
```

```
334 {
```

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.

```
335 \tl_map_break:
```

```
336 }
```

```
337 }
```

```
338 }
```

```

339     }
340     \group_end:
341     \ior_close:N \g_tmpa_ior
342 }

```

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

4.5 Counter

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

```

343 \begingroup
344   \let \@addtoreset \ltx@gobbletwo
345   \newcounter { zrefcheck }
346 \endgroup
347 \setcounter { zrefcheck } { 0 }

```

4.6 Label formats

```

\__zrefcheck_check_lblfmt:n      \__zrefcheck_check_lblfmt:n {<check id int>}
348 \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>}
349 \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}

350 \cs_new_protected:Npn \zrefcheck_get_astl:nnn #1#2#3
351 {
352   \tl_clear:N #3
353   \tl_if_eq:nnTF {#2} { lblseq }
354   {
355     \prop_get:NnNF \g__zrefcheck_auxfile_lblseq_prop {#1} #3
356     {
357       \msg_warning:nnnn { zref-check }
358       { property-not-in-label } {#1} {#2}
359     }
360   }
361   {

```

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.

```

362   \zref@ifrefundefined {#1}
363   {}
364   {
365     \zref@ifpropundefined {#2}
366     { \msg_warning:nnnn { zref-check } { property-undefined } {#2} }
367     {
368       \zref@ifrefcontainsprop {#1} {#2}
369       {
370         \tl_set:Nx #3
371         { \zref@extractdefault {#1} {#2} { \c_empty_tl } }
372       }

```

```

373         {
374             \msg_warning:nnnn
375             { zref-check } { property-not-in-label } {#1} {#2}
376         }
377     }
378 }
379 }
380 }

```

(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).

```

381 \bool_new:N \l__zrefcheck_integer_bool

```

(End definition for \l__zrefcheck_integer_bool.)

\l__zrefcheck_propval_tl

```

382 \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>}
383 \cs_new_protected:Npn \zrefcheck_get_asint:nnn #1#2#3
384 {
385     \zrefcheck_get_astl:nnn {#1} {#2} { \l__zrefcheck_propval_tl }
386     \__zrefcheck_is_integer:nTF { \l__zrefcheck_propval_tl }
387     {

```

Make it an integer data type.

```

388         \int_set:Nn #3 { \int_eval:n { \l__zrefcheck_propval_tl } }
389     }
390 }
391 \bool_set_false:N \l__zrefcheck_integer_bool
392 \zref@ifrefundefined {#1}

```

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

```

393     { }
394     {
395         \msg_warning:nnnn { zref-check }
396         { property-not-integer } {#2} {#1}
397     }

```

```

398     }
399 }

```

(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}
```

```

400 \NewDocumentCommand \zcheck { s O { } m m }
401 { \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
402 \seq_new:N \l__zrefcheck_zcheck_labels_seq
403 \int_new:N \g__zrefcheck_id_int
404 \tl_new:N \l__zrefcheck_checkbeg_tl
405 \tl_new:N \l__zrefcheck_link_label_tl
406 \tl_new:N \l__zrefcheck_link_anchor_tl
407 \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} {\*} {\checks/options} {\text}
```

```

408 \cs_new_protected:Npn \__zrefcheck_zcheck:nnnn #1#2#3#4
409 {
410   \group_begin:

```

Process local options and checks.

```

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

```

Names of the labels for this `zcheck` call.

```

413     \int_gincr:N \g__zrefcheck_id_int
414     \tl_set:Nx \l__zrefcheck_checkbeg_tl
415     { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }

```

Set `checkbeg` label.

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

Typeset $\langle text \rangle$, 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.

```

417 \seq_get:NN \l__zrefcheck_zcheck_labels_seq \l__zrefcheck_link_label_tl
418 \bool_set:Nn \l__zrefcheck_link_star_bool {#2}
419 \zref@ifrefundefined { \l__zrefcheck_link_label_tl }

```

If the reference is undefined, just typeset.

```

420 {#4}
421 {
422   \bool_if:nTF
423   {
424     \l__zrefcheck_use_hyperref_bool &&
425     ! \l__zrefcheck_link_star_bool
426   }
427   {
428     \exp_args:Nx \zrefcheck_get_astl:nnn
429     { \l__zrefcheck_link_label_tl }
430     { anchor } { \l__zrefcheck_link_anchor_tl }
431     \hyperlink { \l__zrefcheck_link_anchor_tl } {#4}
432   }
433   {#4}
434 }

```

Set checkend label.

```

435 \bool_if:NT \l__zrefcheck_zcheck_end_label_bool
436 {
437   \zref@labelbylist
438   { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
439   { zrefcheck-end }
440 }

```

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

```

441 \seq_map_function:NN \l__zrefcheck_zcheck_labels_seq \zref@refused

```

Run the checks.

```

442 \__zrefcheck_run_checks:nnx { \l__zrefcheck_zcheck_checks_seq }
443 { \l__zrefcheck_zcheck_labels_seq } { \l__zrefcheck_checkbeg_tl }
444 \group_end:
445 }

```

(End definition for $_zrefcheck_zcheck:nnnn$.)

5.2 Targets

```

\zctarget \zctarget{<label>}{<text>}
446 \NewDocumentCommand \zctarget { m +m }
447 {

```

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.

```

448 \group_begin:
449 \refstepcounter { zrefcheck }
450 \zref@wrapper@babel \_zrefcheck_target_label:n {#1}

```



```

451     #2
452     \tl_if_empty:nF {#2}
453     {
454         \zref@wrapper@babel
455         \zref@labelbylist { \_zrefcheck\_end\_lbfmt:n {#1} } { zrefcheck-end }
456     }
457     \group\_end:
458 }

```

(End definition for `\zctarget`.)

```

zcregion
    \begin{zcregion}{\langle label \rangle}
    ...
    \end{zcregion}
459 \NewDocumentEnvironment {zcregion} { m }
460 {
461     \refstepcounter { zrefcheck }
462     \zref@wrapper@babel \_zrefcheck\_target\_label:n {#1}
463 }
464 {
465     \zref@wrapper@babel
466     \zref@labelbylist { \_zrefcheck\_end\_lbfmt:n {#1} } { zrefcheck-end }
467 }

```

(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_\langle check \rangle:nn #1#2 { F }
```

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

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

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

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

Third, the check must be registered as a key which gets setup in `\zcheck` by 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
468 \seq_new:N \c__zrefcheck_single_label_checks_seq
469 \seq_set_from_clist:Nn \c__zrefcheck_single_label_checks_seq
470 {
471   thischap ,
472   prevchap ,
473   nextchap ,
474   chapsbefore ,
475   chapsafter ,
476   thissec ,
477   prevsec ,
478   nextsec ,
479   secsbefore ,
480   secsafter ,
481 }
```

(End definition for `\c__zrefcheck_single_label_checks_seq`.)

6.2 Setup

```
\l__zrefcheck_zcheck_checks_seq
\l__zrefcheck_end_label_required_bool
482 \seq_new:N \l__zrefcheck_zcheck_checks_seq
483 \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`.

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

Then we add the checks to it.

```
485 \clist_map_inline:nn
486 {
487   thispage ,
488   prevpage ,
489   nextpage ,
490   facing ,
491   pagegap ,
492   above ,
493   below ,
494   pagesbefore ,
495   ppbefore ,
496   pagesafter ,
497   ppafter ,
498   before ,
499   after ,
500   thischap ,
501   prevchap ,
502   nextchap ,
503   chapsbefore ,
```

```

504     chapsafter ,
505     thissec ,
506     prevsec ,
507     nextsec ,
508     secsbefore ,
509     secsafter ,
510     close ,
511     far ,
512 }
513 {
514   \keys_define:nn { zref-check / zcheck }
515   {
516     #1 .code:n =
517     {
518       \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq {#1}
519       \seq_if_in:NnF \c__zrefcheck_single_label_checks_seq {#1}
520       { \bool_set_true:N \l__zrefcheck_zcheck_end_label_bool }
521     } ,
522     #1 .value_forbidden:n = true ,
523   }
524 }

```

6.3 Running

```

\__zrefcheck_run_checks:nnn   \__zrefcheck_run_checks:nnn {<checks>} {<labels>} {<reference>}
<checks> are expected to be received as a sequence variable.
525 \cs_new_protected:Npn \__zrefcheck_run_checks:nnn #1#2#3
526 {
527   \group_begin:
528   \seq_map_inline:Nn #2
529   {
530     \seq_map_inline:Nn #1
531     { \__zrefcheck_do_check:nnn {####1} {##1} {#3} }
532   }
533   \group_end:
534 }
535 \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
536 \bool_new:N \l__zrefcheck_passedcheck_bool
537 \bool_new:N \l__zrefcheck_onpage_bool
538 \seq_new:N \c__zrefcheck_onpage_checks_seq
539 \seq_set_from_clist:Nn \c__zrefcheck_onpage_checks_seq
540 { 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`.

```

541 \cs_generate_variant:Nn \exp_args:Nno { Nnoo }

```

```

\__zrefcheck_do_check:nnn      \__zrefcheck_do_check:nnn {\check} {\label beg} {\reference beg}
542 \cs_new_protected:Npn \__zrefcheck_do_check:nnn #1#2#3
543 {
544   \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.
545   \zref@ifrefundefined {#2}
546   {}
547   {
548     \tl_if_empty:nF {#1}
549     {
550       \bool_set_true:N \l__zrefcheck_passedcheck_bool
551       \bool_set_false:N \l__zrefcheck_onpage_bool
552       \cs_if_exist:cTF { __zrefcheck_check_ #1 :nnF }
553       {
554         % ‘‘label beg’’ vs ‘‘reference beg’’.
555         \use:c { __zrefcheck_check_ #1 :nnF }
556         {#2} {#3}
557         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
558         % ‘‘reference end’’ \emph{may} exist or not depending on the
559         % checks.
560         \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#3} }
561         {
562           % ‘‘label end’’ \emph{may} have been created by the
563           % target commands.
564           \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
565           {}
566           {
567             % ‘‘label end’’ vs ‘‘reference beg’’.
568             \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
569             { \__zrefcheck_end_lblfmt:n {#2} } {#3}
570             { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
571           }
572         }
573         {
574           % ‘‘label beg’’ vs ‘‘reference end’’.
575           \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
576           {#2} { \__zrefcheck_end_lblfmt:n {#3} }
577           { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
578           % ‘‘label end’’ \emph{may} have been created by the
579           % target commands.
580           \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
581           {}
582           {
583             % ‘‘label end’’ vs ‘‘reference beg’’.
584             \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
585             { \__zrefcheck_end_lblfmt:n {#2} } {#3}
586             { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
587             % ‘‘label end’’ vs ‘‘reference end’’.
588             \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
589             { \__zrefcheck_end_lblfmt:n {#2} }

```

```

590         { \__zrefcheck_end_lblfmt:n {#3} }
591         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
592     }
593 }

```

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.

```

594     \seq_if_in:NnT \c__zrefcheck_onpage_checks_seq {#1}
595     {
596         \__zrefcheck_check_thispage:nnT
597         {#2} {#3}
598         { \bool_set_true:N \l__zrefcheck_onpage_bool }
599         \__zrefcheck_check_thispage:nnT
600         {#2} { \__zrefcheck_end_lblfmt:n {#3} }
601         { \bool_set_true:N \l__zrefcheck_onpage_bool }
602         \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
603         {}
604         {
605             \__zrefcheck_check_thispage:nnT
606             { \__zrefcheck_end_lblfmt:n {#2} } {#3}
607             { \bool_set_true:N \l__zrefcheck_onpage_bool }
608             \__zrefcheck_check_thispage:nnT
609             { \__zrefcheck_end_lblfmt:n {#2} }
610             { \__zrefcheck_end_lblfmt:n {#3} }
611             { \bool_set_true:N \l__zrefcheck_onpage_bool }
612         }
613     }
614     \bool_if:NtF \l__zrefcheck_passedcheck_bool
615     {
616         \bool_if:Nt
617         {
618             \l__zrefcheck_msgonpage_bool &&
619             \l__zrefcheck_onpage_bool
620         }
621         {
622             \__zrefcheck_message:nnnx { double-check } {#1} {#2}
623             { \zref@extractdefault {#3} {page} {'unknown'} }
624         }
625     }
626     {
627         \__zrefcheck_message:nnnx { check-failed } {#1} {#2}
628         { \zref@extractdefault {#3} {page} {'unknown'} }
629     }
630 }
631 { \msg_warning:nnn { zref-check } { check-missing } {#1} }
632 }
633 }
634 \group_end:
635 }

```

(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.

```

636 \int_new:N \l__zrefcheck_lbl_int
637 \int_new:N \l__zrefcheck_ref_int
638 \int_new:N \l__zrefcheck_lbl_b_int
639 \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

```

```

640 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thispage:nn #1#2 { T , F , TF }
641 {
642   \group_begin:
643     \bool_set_true:N \l__zrefcheck_integer_bool
644     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
645     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
646     \bool_lazy_and:nnTF
647       { \l__zrefcheck_integer_bool }
648     {
649       \int_compare_p:nNn
650         { \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.

```

651       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
652       ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
653     }
654     { \group_insert_after:N \prg_return_true: }
655     { \group_insert_after:N \prg_return_false: }
656   \group_end:
657 }

```

(End definition for `__zrefcheck_check_thispage:nn`.)

6.4.2 On page

```

\__zrefcheck_check_above:nn
\__zrefcheck_check_below:nn

```

```

658 \prg_new_protected_conditional:Npnn \__zrefcheck_check_above:nn #1#2 { F , TF }
659 {
660   \group_begin:
661     \__zrefcheck_check_thispage:nnTF {#1} {#2}
662     {
663       \bool_set_true:N \l__zrefcheck_integer_bool
664       \zrefcheck_get_asint:nnn {#1} { lblseq } { \l__zrefcheck_lbl_int }
665       \zrefcheck_get_asint:nnn {#2} { lblseq } { \l__zrefcheck_ref_int }
666       \bool_lazy_and:nnTF
667         { \l__zrefcheck_integer_bool }
668       {
669         \int_compare_p:nNn

```

```

670         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
671         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
672         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
673     }
674     { \group_insert_after:N \prg_return_true: }
675     { \group_insert_after:N \prg_return_false: }
676 }
677 { \group_insert_after:N \prg_return_false: }
678 \group_end:
679 }
680 \prg_new_protected_conditional:Npnn \__zrefcheck_check_below:nn #1#2 { F , TF }
681 {
682     \__zrefcheck_check_thispage:nnTF {#1} {#2}
683     {
684         \__zrefcheck_check_above:nnTF {#1} {#2}
685         { \prg_return_false: }
686         { \prg_return_true: }
687     }
688     { \prg_return_false: }
689 }

```

(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
690 \prg_new_protected_conditional:Npnn \__zrefcheck_check_before:nn #1#2 { F }
691 {
692     \__zrefcheck_check_pagesbefore:nnTF {#1} {#2}
693     { \prg_return_true: }
694     {
695         \__zrefcheck_check_above:nnTF {#1} {#2}
696         { \prg_return_true: }
697         { \prg_return_false: }
698     }
699 }
700 \prg_new_protected_conditional:Npnn \__zrefcheck_check_after:nn #1#2 { F }
701 {
702     \__zrefcheck_check_pagesafter:nnTF {#1} {#2}
703     { \prg_return_true: }
704     {
705         \__zrefcheck_check_below:nnTF {#1} {#2}
706         { \prg_return_true: }
707         { \prg_return_false: }
708     }
709 }

```

(End definition for __zrefcheck_check_before:nn and __zrefcheck_check_after:nn.)

6.4.4 Pages

```

\__zrefcheck_check_nextpage:nn
\__zrefcheck_check_prevpage:nn
710 \prg_new_protected_conditional:Npnn \__zrefcheck_check_nextpage:nn #1#2 { F }
\__zrefcheck_check_pagesbefore:nn
711 {
\__zrefcheck_check_ppbefore:nn
\__zrefcheck_check_pagesafter:nn
\__zrefcheck_check_ppafter:nn
\__zrefcheck_check_pagegap:nn
\__zrefcheck_check_facing:nn

```

```

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_prevpage:nn #1#2 { F }
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 - 1 } &&
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 \prg_new_protected_conditional:Npnn \__zrefcheck_check_pagesbefore:nn #1#2 { F , TF }
747 {
748 \group_begin:
749 \bool_set_true:N \l__zrefcheck_integer_bool
750 \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
751 \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
752 \bool_lazy_and:nnTF
753 { \l__zrefcheck_integer_bool }
754 {
755 \int_compare_p:nNn
756 { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
757 ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
758 ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
759 }
760 { \group_insert_after:N \prg_return_true: }
761 { \group_insert_after:N \prg_return_false: }
762 \group_end:
763 }
764 \cs_new_eq:NN \__zrefcheck_check_ppbefore:nnF \__zrefcheck_check_pagesbefore:nnF
765 \prg_new_protected_conditional:Npnn \__zrefcheck_check_pagesafter:nn #1#2 { F , TF }

```



```

766 {
767   \group_begin:
768     \bool_set_true:N \l__zrefcheck_integer_bool
769     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
770     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
771     \bool_lazy_and:nnTF
772       { \l__zrefcheck_integer_bool }
773       {
774         \int_compare_p:nNn
775           { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
776           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
777           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
778       }
779       { \group_insert_after:N \prg_return_true: }
780       { \group_insert_after:N \prg_return_false: }
781   \group_end:
782 }
783 \cs_new_eq:NN \__zrefcheck_check_ppafter:nnF \__zrefcheck_check_pagesafter:nnF
784 \prg_new_protected_conditional:Npnn \__zrefcheck_check_pagegap: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       {
793         \int_compare_p:nNn
794           { \int_abs:n { \l__zrefcheck_lbl_int - \l__zrefcheck_ref_int } } > { 1 } &&
795           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
796           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
797       }
798       { \group_insert_after:N \prg_return_true: }
799       { \group_insert_after:N \prg_return_false: }
800   \group_end:
801 }
802 \prg_new_protected_conditional:Npnn \__zrefcheck_check_facing:nn #1#2 { F }
803 {
804   \group_begin:
805     \bool_set_true:N \l__zrefcheck_integer_bool
806     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
807     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
808     \bool_lazy_and:nnTF
809       { \l__zrefcheck_integer_bool }
810       {

```

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

```

811     \legacy_if_p:n { @twoside } &&

```

Now we test “facing”.

```

812     (
813       (
814         \int_if_odd_p:n { \l__zrefcheck_ref_int } &&
815         \int_compare_p:nNn
816           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 }

```

```

817         ) ||
818         (
819             \int_if_even_p:n { \l__zrefcheck_ref_int } &&
820             \int_compare_p:nNn
821                 { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 }
822         )
823     ) &&
824     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
825     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
826 }
827 { \group_insert_after:N \prg_return_true: }
828 { \group_insert_after:N \prg_return_false: }
829 \group_end:
830 }

```

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

6.4.5 Close / Far

```

\__zrefcheck_check_close:nn
\__zrefcheck_check_far:nn
831 \prg_new_protected_conditional:Npnn \__zrefcheck_check_close:nn #1#2 { F , TF }
832 {
833     \group_begin:
834     \bool_set_true:N \l__zrefcheck_integer_bool
835     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
836     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
837     \bool_lazy_and:nnTF
838         { \l__zrefcheck_integer_bool }
839         {
840             \int_compare_p:nNn
841                 { \int_abs:n { \l__zrefcheck_lbl_int - \l__zrefcheck_ref_int } }
842                 <
843                 { \l__zrefcheck_close_range_int + 1 } &&
844             ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
845             ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
846         }
847     { \group_insert_after:N \prg_return_true: }
848     { \group_insert_after:N \prg_return_false: }
849     \group_end:
850 }
851 \prg_new_protected_conditional:Npnn \__zrefcheck_check_far:nn #1#2 { F }
852 {
853     \__zrefcheck_check_close:nnTF {#1} {#2}
854     { \prg_return_false: }
855     { \prg_return_true: }
856 }

```

(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
857 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thischap: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 } &&

```

'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.

```

868 ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
869 ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
870 }
871 { \group_insert_after:N \prg_return_true: }
872 { \group_insert_after:N \prg_return_false: }
873 \group_end:
874 }
875 \prg_new_protected_conditional:Npnn \__zrefcheck_check_nextchap:nn #1#2 { F }
876 {
877 \group_begin:
878 \bool_set_true:N \l__zrefcheck_integer_bool
879 \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
880 \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
881 \bool_lazy_and:nnTF
882 { \l__zrefcheck_integer_bool }
883 {
884 \int_compare_p:nNn
885 { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
886 ! \int_compare_p:nNn { \l__zrefcheck_lbl_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_prevchap: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 - 1 } &&
903 ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
904 ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
905 }
906 { \group_insert_after:N \prg_return_true: }

```

```

907         { \group_insert_after:N \prg_return_false: }
908     \group_end:
909 }
910 \prg_new_protected_conditional:Npnn \__zrefcheck_check_chapsafter:nn #1#2 { F }
911 {
912     \group_begin:
913     \bool_set_true:N \l__zrefcheck_integer_bool
914     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
915     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
916     \bool_lazy_and:nnTF
917     { \l__zrefcheck_integer_bool }
918     {
919         \int_compare_p:nNn
920         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
921         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
922     }
923     { \group_insert_after:N \prg_return_true: }
924     { \group_insert_after:N \prg_return_false: }
925 \group_end:
926 }
927 \prg_new_protected_conditional:Npnn \__zrefcheck_check_chapsbefore:nn #1#2 { F }
928 {
929     \group_begin:
930     \bool_set_true:N \l__zrefcheck_integer_bool
931     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
932     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
933     \bool_lazy_and:nnTF
934     { \l__zrefcheck_integer_bool }
935     {
936         \int_compare_p:nNn
937         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
938         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
939         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
940     }
941     { \group_insert_after:N \prg_return_true: }
942     { \group_insert_after:N \prg_return_false: }
943 \group_end:
944 }

```

(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
945 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thissec:nn #1#2 { F }
946 {
947     \group_begin:
948     \bool_set_true:N \l__zrefcheck_integer_bool
949     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
950     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
951     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
952     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
953     \bool_lazy_and:nnTF
954     { \l__zrefcheck_integer_bool }

```

```

955     {
956         \int_compare_p:nNn
957         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
958         \int_compare_p:nNn
959         { \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.

```

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

```

```

1003         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
1004         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
1005     }
1006     { \group_insert_after:N \prg_return_true: }
1007     { \group_insert_after:N \prg_return_false: }
1008 \group_end:
1009 }
1010 \prg_new_protected_conditional:Npnn \__zrefcheck_check_secsafter:nn #1#2 { F }
1011 {
1012     \group_begin:
1013     \bool_set_true:N \l__zrefcheck_integer_bool
1014     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
1015     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
1016     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
1017     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
1018     \bool_lazy_and:nnTF
1019     { \l__zrefcheck_integer_bool }
1020     {
1021         \int_compare_p:nNn
1022         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
1023         \int_compare_p:nNn
1024         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
1025         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
1026     }
1027     { \group_insert_after:N \prg_return_true: }
1028     { \group_insert_after:N \prg_return_false: }
1029 \group_end:
1030 }
1031 \prg_new_protected_conditional:Npnn \__zrefcheck_check_secsbefore:nn #1#2 { F }
1032 {
1033     \group_begin:
1034     \bool_set_true:N \l__zrefcheck_integer_bool
1035     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
1036     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
1037     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
1038     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
1039     \bool_lazy_and:nnTF
1040     { \l__zrefcheck_integer_bool }
1041     {
1042         \int_compare_p:nNn
1043         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
1044         \int_compare_p:nNn
1045         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
1046         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
1047         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
1048     }
1049     { \group_insert_after:N \prg_return_true: }
1050     { \group_insert_after:N \prg_return_false: }
1051 \group_end:
1052 }

```

(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 \zcref: i) set the “beg label”; ii) set the checks options; iii) run the checks; iv) (possibly) set the “end label”. Since ‘ii)’ can be done directly by running \keys_set:nn { zref-check / zcheck } on the options received, we provide convenience functions for the other three tasks.

```
\zrefcheck_zcref_beg_label:
  \zrefcheck_zcref_end_label_maybe:
  \zrefcheck_zcref_run_checks_on_labels:n
1053 \cs_new_protected:Npn \zrefcheck_zcref_beg_label:
1054 {
1055   \int_gincr:N \g__zrefcheck_id_int
1056   \tl_set:Nx \l__zrefcheck_checkbeg_tl
1057     { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
1058   \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrefcheck-check }
1059 }
1060 \cs_new_protected:Npn \zrefcheck_zcref_end_label_maybe:
1061 {
1062   \bool_if:NT \l__zrefcheck_zcheck_end_label_bool
1063     {
1064       \zref@labelbylist
1065         { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
1066         { zrefcheck-end }
1067     }
1068 }
1069 \cs_new_protected:Npn \zrefcheck_zcref_run_checks_on_labels:n #1
1070 {
1071   \__zrefcheck_run_checks:nnx
1072     { \l__zrefcheck_zcheck_checks_seq } {#1} { \l__zrefcheck_checkbeg_tl }
1073 }
```

(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 ??.)

```
1074 </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	
\\	76
A	
\A	115
\AddToHook	21, 28, 142, 203, 275
B	
\begingroup	343
bool commands:	
\bool_if:N	146, 153, 277, 327, 435, 614, 1062
\bool_if:nTF	422, 616
\bool_lazy_and:nnTF	14, 646, 666, 716, 734, 752, 771, 790, 808, 837, 863, 881, 898, 916, 933, 953, 975, 996, 1018, 1039
\bool_new:N	119, 120, 208, 263, 381, 407, 483, 536, 537
\bool_set:Nn	418
\bool_set_false:N	127, 136, 137, 155, 214, 223, 230, 317, 329, 391, 551, 557, 570, 577, 586, 591
\bool_set_true:N	126, 131, 132, 218,

224, 229, 269, 324, 520, 550, 598, 601, 607, 611, 643, 663, 713, 731, 749, 768, 787, 805, 834, 860, 878, 895, 913, 930, 948, 970, 991, 1013, 1034 \l_tmpa_bool . . . 11, 317, 324, 327, 329		848, 871, 872, 888, 889, 906, 907, 923, 924, 941, 942, 963, 964, 984, 985, 1006, 1007, 1027, 1028, 1049, 1050	
C		H	
\catcode 11		\hyperlink 15, 431	
\chapter 2, 27		I	
clist commands:		\ifdraft 175, 222	
\clist_map_inline:nn 485		\IfFormatAtLeastTF 3, 4	
cs commands:		\ifoptionfinal 181, 228	
\cs:w 286		int commands:	
\cs_end: 286		\int_abs:n 794, 841	
\cs_generate_variant:Nn 60, 535, 541		\int_compare_p:nNn	
\cs_if_exist:NTF 281, 552	 649, 651, 652, 669, 671,	
\cs_new:Npn 348, 349		672, 719, 721, 722, 737, 739, 740,	
\cs_new_eq:NN 102, 764, 783		755, 757, 758, 774, 776, 777, 793,	
\cs_new_protected:Npn 55, 273, 350,		795, 796, 815, 820, 824, 825, 840,	
383, 408, 525, 542, 1053, 1060, 1069		844, 845, 866, 868, 869, 884, 886,	
\cs_set_protected:Npx 283		901, 903, 904, 919, 921, 936, 938,	
		939, 956, 958, 960, 961, 978, 980,	
D		982, 999, 1001, 1003, 1004, 1021,	
\d 115		1023, 1025, 1042, 1044, 1046, 1047	
E		\int_eval:n 253, 388	
\emph 558, 562, 578		\int_gincr:N 23, 29, 413, 1055	
\endgroup 346		\int_if_even_p:n 819	
\endinput 12		\int_if_odd_p:n 814	
exp commands:		\int_incr:N 330	
\exp_args:Nnno 291, 541, 575		\int_new:N	
\exp_args:Nno 568, 584	 19, 20, 247, 403, 636, 637, 638, 639	
\exp_args:Nnoo 588		\int_set:Nn 253, 256, 388	
\exp_args:Nx 428		\int_use:N 26, 30, 332, 348	
\exp_not:n 285		\int_zero:N 24, 314	
F		\l_tmpa_int 314, 330, 332	
file commands:		int internal commands:	
\file_if_exist:nTF 310		__int_to_roman:w 4, 5, 102	
\fmtversion 3		ior commands:	
G		\ior_close:N 341	
group commands:		\ior_map_variable:NNn 318	
\group_begin: 313, 410,		\ior_open:Nn 312	
448, 527, 544, 642, 660, 712, 730,		\g_tmpa_ior 312, 318, 341	
748, 767, 786, 804, 833, 859, 877,		iow commands:	
894, 912, 929, 947, 969, 990, 1012, 1033		\iow_char:N 76	
\group_end: 340, 444, 457,		\iow_newline: 75, 79, 82, 99	
533, 634, 656, 678, 726, 744, 762,		K	
781, 800, 829, 849, 873, 890, 908,		keys commands:	
925, 943, 965, 986, 1008, 1029, 1051		\keys_define:nn 121, 157,	
\group_insert_after:N		164, 205, 209, 248, 264, 295, 484, 514	
. 654, 655, 674, 675,		\keys_set:nn 307, 411	
677, 724, 725, 742, 743, 760, 761,		L	
779, 780, 798, 799, 827, 828, 847,		\label 12	
		legacy commands:	
		\legacy_if_p:n 811	

<code>\let</code>	344		S	
		M		
<code>\MessageBreak</code>	10		<code>\section</code>	29
msg commands:			seq commands:	
<code>\msg_line_context:</code>			<code>\seq_get:NN</code>	417
.....	62, 64, 66, 68, 70, 72, 86, 89		<code>\seq_if_in:NnTF</code>	519, 594
<code>\msg_new:nnn</code>	61, 63,		<code>\seq_map_function:NN</code>	441
.....	65, 67, 69, 71, 73, 78, 80, 85, 87, 92, 97		<code>\seq_map_inline:Nn</code>	528, 530
<code>\msg_warning:nn</code> ...	154, 160, 200, 255		<code>\seq_new:N</code>	402, 468, 482, 538
<code>\msg_warning:nnn</code>	291, 299, 631		<code>\seq_put_right:Nn</code>	518
<code>\msg_warning:nnnn</code>			<code>\seq_set_from_clist:Nn</code> .	412, 469, 539
.....	188, 194, 235, 241, 357, 366, 374, 395		<code>\setcounter</code>	347
			sys commands:	
			<code>\c_sys_jobname_str</code>	309
		N		
<code>\newcounter</code>	345		T	
<code>\NewDocumentCommand</code>	306, 400, 446		TeX and L ^A T _E X 2 _ε commands:	
<code>\NewDocumentEnvironment</code>	459		<code>\@addtoreset</code>	344
		P	<code>\@currentlabel</code>	16
<code>\PackageError</code>	7		<code>\@ifl@t@r</code>	3
<code>\pageref</code>	12		<code>\@ifpackageloaded</code>	144
prg commands:			<code>\@newl@bel</code>	10
<code>\prg_new_conditional:Npnn</code>	103		<code>\ltx@gobbletwo</code>	344
<code>\prg_new_protected_conditional:Npnn</code>			<code>\zref@addprop</code>	17, 27, 31, 149
.....	17, 113, 640,		<code>\zref@addprops</code>	33, 41, 49
.....	658, 680, 690, 700, 710, 728, 746,		<code>\zref@extractdefault</code> .	13, 371, 623, 628
.....	765, 784, 802, 831, 851, 857, 875,		<code>\zref@ifpropundefined</code>	365
.....	892, 910, 927, 945, 967, 988, 1010, 1031		<code>\zref@ifrefcontainsprop</code>	368
<code>\prg_return_false:</code>	17,		<code>\zref@ifrefundefined</code>	13,
.....	106, 110, 117, 655, 675, 677, 685,		362, 392, 419, 545, 560, 564, 580, 602
.....	688, 697, 707, 725, 743, 761, 780,		<code>\ZREF@label</code>	12
.....	799, 828, 848, 854, 872, 889, 907,		<code>\zref@label</code>	11, 12
.....	924, 942, 964, 985, 1007, 1028, 1050		<code>\zref@labelbylist</code>	
<code>\prg_return_true:</code>	274, 416, 437, 455, 466, 1058, 1064
.....	17, 109, 116, 654, 674, 686, 693,		<code>\ZREF@mainlist</code>	27, 31
.....	696, 703, 706, 724, 742, 760, 779,		<code>\zref@newlabel</code>	10, 11, 322
.....	798, 827, 847, 855, 871, 888, 906,		<code>\zref@newlist</code>	32, 40, 48
.....	923, 941, 963, 984, 1006, 1027, 1049		<code>\zref@newprop</code>	17, 26, 30
<code>\ProcessKeysOptions</code>	305		<code>\zref@refused</code>	13, 441
prop commands:			<code>\zref@require@unique</code>	12
<code>\prop_get:NnNTF</code>	355		<code>\zref@wrapper@babel</code>	
<code>\prop_gput:Nnn</code>	331		10, 15, 401, 450, 454, 462, 465
<code>\prop_new:N</code>	308		tl commands:	
<code>\providecommand</code>	3		<code>\c_empty_tl</code>	13, 371
<code>\ProvidesExplPackage</code>	14		<code>\tl_clear:N</code>	13, 280, 315, 316, 352
		R	<code>\tl_if_blank:nTF</code>	4, 279
<code>\refstepcounter</code>	12, 16, 449, 461		<code>\tl_if_empty:nTF</code> .	4, 105, 108, 452, 548
regex commands:			<code>\tl_if_eq:NnTF</code>	322
<code>\regex_match:nnTF</code>	115		<code>\tl_if_eq:nnTF</code>	353
<code>\RequirePackage</code>	16, 17, 18, 148, 304		<code>\tl_map_break:</code>	335
<code>\romannumeral</code>	4		<code>\tl_map_variable:NNn</code>	320
			<code>\tl_new:N</code> ...	163, 262, 382, 404, 405, 406
			<code>\tl_set:Nn</code>	168, 170, 172, 176,
			177, 182, 183, 268, 309, 370, 414, 1056
			<code>\g_tmpa_tl</code>	309, 310, 312

\l_tmpa_tl	315, 318, 320	__zrefcheck_check_nextchap:nn	857
\l_tmpb_tl	316, 320, 322, 332	__zrefcheck_check_nextpage:nn	710
U			
use commands:			
\use:N	57, 555, 568, 575, 584, 588	__zrefcheck_check_nextsec:nn	945
Z			
\Z	115	__zrefcheck_check_pagegap:nn	710
\zcheck	15, 17, 18, 400	__zrefcheck_check_pagesafter:nn	710
\zceref	31	__zrefcheck_check_pagesafter:nnTF	702, 783
zcregion	459	__zrefcheck_check_pagesbefore:nn	710
\zctarget	9, 16, 446	__zrefcheck_check_pagesbefore:nnTF	692, 764
\zpageref	43	__zrefcheck_check_ppafter:nn	710
\zref	15	__zrefcheck_check_ppafter:nnTF	783
zrefcheck commands:			
\zrefcheck_get_asint:nnn	14, 17, 383, 644, 645, 664, 665, 714, 715, 732, 733, 750, 751, 769, 770, 788, 789, 806, 807, 835, 836, 861, 862, 879, 880, 896, 897, 914, 915, 931, 932, 949, 950, 951, 952, 971, 972, 973, 974, 992, 993, 994, 995, 1014, 1015, 1016, 1017, 1035, 1036, 1037, 1038	__zrefcheck_check_ppbefore:nn	710
\zrefcheck_get_astl:nnn	13, 14, 17, 350, 385, 428	__zrefcheck_check_ppbefore:nnTF	764
\zrefcheck_zceref_beg_label:	1053	__zrefcheck_check_prevchap:nn	857
\zrefcheck_zceref_end_label_-maybe:	1053	__zrefcheck_check_prevpage:nn	710
\zrefcheck_zceref_run_checks_on_-labels:n	1053	__zrefcheck_check_prevsec:nn	945
zrefcheck internal commands:			
\g__zrefcheck_abschap_int	19, 23, 26	__zrefcheck_check_secsafter:nn	945
\g__zrefcheck_abssec_int	19, 24, 29, 30	__zrefcheck_check_secsbefore:nn	945
\g__zrefcheck_auxfile_lblseq_-prop	13, 17, 308, 331, 355	__zrefcheck_check_thischap:nn	857
__zrefcheck_check_<check>:nn	17	__zrefcheck_check_thispage:nn	640
__zrefcheck_check_above:nn	658	__zrefcheck_check_thispage:nnTF	596, 599, 605, 608, 661, 682
__zrefcheck_check_above:nnTF	684, 695	__zrefcheck_check_thissec:nn	945
__zrefcheck_check_after:nn	690	\l__zrefcheck_checkbeg_tl	402, 414, 416, 438, 443, 1056, 1058, 1065, 1072
__zrefcheck_check_before:nn	690	\l__zrefcheck_close_range_int	247, 843
__zrefcheck_check_below:nn	658	__zrefcheck_do_check:nnn	20, 531, 542
__zrefcheck_check_below:nnTF	705	\l__zrefcheck_end_label_required_-bool	482
__zrefcheck_check_chapsafter:nn	857	__zrefcheck_end_lblfmt:n	12, 349, 438, 455, 466, 560, 564, 569, 576, 580, 585, 589, 590, 600, 602, 606, 609, 610, 1065
__zrefcheck_check_chapsbefore:nn	857	__zrefcheck_get_asint:nnn	13
__zrefcheck_check_close:nn	831	__zrefcheck_get_astl:nnn	13
__zrefcheck_check_close:nnTF	853	\g__zrefcheck_id_int	402, 413, 415, 1055, 1057
__zrefcheck_check_facing:nn	710	__zrefcheck_int_to_roman:w	102
__zrefcheck_check_far:nn	831	\l__zrefcheck_integer_bool	14, 381, 391, 643, 647, 663, 667, 713, 717, 731, 735, 749, 753, 768, 772, 787, 791, 805, 809, 834, 838, 860, 864, 878, 882, 895, 899, 913, 917, 930, 934, 948, 954, 970, 976, 991, 997, 1013, 1019, 1034, 1040
__zrefcheck_check_lblfmt:n	12, 348, 415, 1057	__zrefcheck_is_integer:n	5, 102
		__zrefcheck_is_integer:nnTF	386
		__zrefcheck_is_integer_rgx:n	5, 113
		__zrefcheck_is_integer_rgx:nnTF	252

\l__zrefcheck_lbl_b_int	\l__zrefcheck_ref_int ..
..... 636, 951, 957, 973,	636, 645,
979, 994, 1000, 1016, 1022, 1037, 1043	650, 652, 665, 670, 672, 715, 720,
\l__zrefcheck_lbl_int	722, 733, 738, 740, 751, 756, 758,
..... 636, 644, 650, 651,	770, 775, 777, 789, 794, 796, 807,
664, 670, 671, 714, 720, 721, 732,	814, 816, 819, 821, 825, 836, 841,
738, 739, 750, 756, 757, 769, 775,	845, 862, 867, 869, 880, 885, 897,
776, 788, 794, 795, 806, 816, 821,	902, 904, 915, 920, 932, 937, 939,
824, 835, 841, 844, 861, 867, 868,	950, 959, 961, 972, 981, 993, 1002,
879, 885, 886, 896, 902, 903, 914,	1004, 1015, 1024, 1036, 1045, 1047
920, 921, 931, 937, 938, 949, 959,	__zrefcheck_run_checks:nnn
960, 971, 981, 982, 992, 1002, 1003, 19, 442, 525, 1071
1014, 1024, 1025, 1035, 1045, 1046	\c__zrefcheck_single_label_-
\l__zrefcheck_link_anchor_tl ...	checks_seq
..... 402, 430, 431 17, 468, 519
\l__zrefcheck_link_label_tl ...	__zrefcheck_target_label:n
..... 402, 417, 419, 429 273, 283, 450, 462
\l__zrefcheck_link_star_bool ...	\l__zrefcheck_target_label_bool .
..... 402, 418, 425 263, 269, 277
__zrefcheck_message:nnnn 55, 622, 627	\l__zrefcheck_target_label_tl ...
\l__zrefcheck_msglevel_tl ... 57, 163 262, 279, 280, 281, 286, 292
\l__zrefcheck_msgonpage_bool 208, 618	\l__zrefcheck_use_hyperref_bool .
\l__zrefcheck_onpage_bool 119, 146, 155, 424
..... 536, 551, 598, 601, 607, 611, 619	\l__zrefcheck_warn_hyperref_bool
\c__zrefcheck_onpage_checks_seq 119, 153
..... 536, 594	__zrefcheck_zcheck:nnnn 15, 401, 408
\l__zrefcheck_passedcheck_bool ..	__zrefcheck_zcheck:nnnnn 13, 14, 20
..... 536, 550, 557, 570, 577, 586, 591, 614	\l__zrefcheck_zcheck_checks_seq .
\l__zrefcheck_propval_tl 442, 482, 518, 1072
..... 382, 385, 386, 388	\l__zrefcheck_zcheck_end_label_-
\l__zrefcheck_ref_b_int	bool
..... 636, 952, 957, 974, 435, 483, 520, 1062
979, 995, 1000, 1017, 1022, 1038, 1043	\l__zrefcheck_zcheck_labels_seq .
 402, 412, 417, 441, 443
	\zrefchecksetup
 9, 306