

The `zref-check` package implementation*

Gustavo Barros[†]

2021-12-20

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	12
5	User interface	14
5.1	<code>\zcheck</code>	14
5.2	Targets	16
6	Checks	17
6.1	Single label checks	17
6.2	Setup	18
6.3	Running	19
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
7	<code>zref-clever</code> integration	30

*This file describes v0.2.3, released 2021-12-20.

[†]<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} {2021-12-20} {0.2.3}
```

```
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. For labels set with `__zrefcheck_target_label:n`, that is, user facing labels, we must use the `main` property list, so that zref-clever can also retrieve the properties it needs to refer to them.

```

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

```

4 Plumbing

4.1 Messages

```

\__zrefcheck_message:nnnn
\__zrefcheck_message:nnnx
47 \cs_new_protected:Npn \__zrefcheck_message:nnnn #1#2#3#4
48 {
49   \use:c { msg_ \l__zrefcheck_msglevel_tl :nnnnn }
50   { zref-check } {#1} {#2} {#3} {#4}
51 }
52 \cs_generate_variant:Nn \__zrefcheck_message:nnnn { nnnx }

(End definition for \__zrefcheck_message:nnnn.)

53 \msg_new:nnn { zref-check } { check-failed }
54 { Failed~check~'~#1'~for~label~'~#2'~on~page~#3~\msg_line_context:. }
55 \msg_new:nnn { zref-check } { double-check }
56 { Double-check~'~#1'~for~label~'~#2'~on~page~#3~\msg_line_context:. }

57 \msg_new:nnn { zref-check } { check-missing }
58 { Check~'~#1'~not~defined~\msg_line_context:. }
59 \msg_new:nnn { zref-check } { property-undefined }
60 { Property~'~#1'~not~defined~\msg_line_context:. }
61 \msg_new:nnn { zref-check } { property-not-in-label }
62 { Label~'~#1'~has~no~property~'~#2'~\msg_line_context:. }
63 \msg_new:nnn { zref-check } { property-not-integer }
64 { Property~'~#1'~for~label~'~#2'~not~an~integer~\msg_line_context:. }

```

```

65 \msg_new:nnn { zref-check } { hyperref-preamble-only }
66 {
67   Option~'hyperref'~only~available~in~the~preamble. \iow_newline:
68   Use~the~starred~version~of~'\iow_char:N\zcheck'~instead.
69 }
70 \msg_new:nnn { zref-check } { missing-hyperref }
71 { Missing~'hyperref'~package. \iow_newline: Setting~'hyperref=false'. }
72 \msg_new:nnn { zref-check } { ignore-document-only }
73 {
74   Option~'ignore'~only~available~in~the~document. \iow_newline:
75   Use~option~'msglevel'~instead.
76 }
77 \msg_new:nnn { zref-check } { option-preamble-only }
78 { Option~'#1'~only~available~in~the~preamble~\msg_line_context:. }
79 \msg_new:nnn { zref-check } { closerange-not-positive-integer }
80 {
81   Option~'closerange'~not~a~positive~integer~\msg_line_context:.~
82   Using~default~value.
83 }
84 \msg_new:nnn { zref-check } { labelcmd-undefined }
85 {
86   Control~sequence~named~'#1'~used~in~option~'labelcmd'~is~not~defined.~
87   Using~default~value.
88 }
89 \msg_new:nnn { zref-check } { option-deprecated }
90 {
91   Option~'#1'~has~been~deprecated.\iow_newline:
92   Use~'#2'~as~a~replacement.
93 }

```

4.2 Integer testing

```

\__zrefcheck_is_integer:n
\__zrefcheck_int_to_roman:w

```

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

```

94 \cs_new_eq:NN \__zrefcheck_int_to_roman:w \__int_to_roman:w
95 \prg_new_conditional:Npnn \__zrefcheck_is_integer:n #1 { p, T , F , TF }
96 {
97   \tl_if_empty:oTF {#1}
98   { \prg_return_false: }
99   {
100     \tl_if_empty:oTF { \__zrefcheck_int_to_roman:w -0#1 }
101     { \prg_return_true: }
102     { \prg_return_false: }

```

```

103     }
104 }

```

(End definition for `_zrefcheck_is_integer:n` and `_zrefcheck_int_to_roman:w`.)

`_zrefcheck_is_integer_rgx:n` A possible alternative to `_zrefcheck_is_integer:n` is to use a straightforward regexp match (see <https://tex.stackexchange.com/a/427559>). It does not suffer from the mentioned caveats from the `_int_to_roman:w` technique, however, while `_zrefcheck_is_integer:n` is expandable, `_zrefcheck_is_integer_rgx:n` is not. Also, `_zrefcheck_is_integer_rgx:n` is probably slower.

```

105 \prg_new_protected_conditional:Npnn \_zrefcheck_is_integer_rgx:n #1 { TF }
106 {
107   \regex_match:nnTF { \A\d+\Z } {#1}
108   { \prg_return_true: }
109   { \prg_return_false: }
110 }

```

(End definition for `_zrefcheck_is_integer_rgx:n`.)

4.3 Options

hyperref option

```

\l_zrefcheck_use_hyperref_bool
\l_zrefcheck_warn_hyperref_bool
111 \bool_new:N \l__zrefcheck_use_hyperref_bool
112 \bool_new:N \l__zrefcheck_warn_hyperref_bool
113 \keys_define:nn { zref-check }
114 {
115   hyperref .choice: ,
116   hyperref / auto .code:n =
117   {
118     \bool_set_true:N \l__zrefcheck_use_hyperref_bool
119     \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
120   } ,
121   hyperref / true .code:n =
122   {
123     \bool_set_true:N \l__zrefcheck_use_hyperref_bool
124     \bool_set_true:N \l__zrefcheck_warn_hyperref_bool
125   } ,
126   hyperref / false .code:n =
127   {
128     \bool_set_false:N \l__zrefcheck_use_hyperref_bool
129     \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
130   } ,
131   hyperref .initial:n = auto ,
132   hyperref .default:n = auto
133 }

```

(End definition for `\l__zrefcheck_use_hyperref_bool` and `\l__zrefcheck_warn_hyperref_bool`.)

```

134 \AddToHook { begindocument }
135 {
136   \@ifpackageloaded { hyperref }
137   {
138     \bool_if:NT \l__zrefcheck_use_hyperref_bool

```

```

139         { \RequirePackage { zref-hyperref } }
140     }
141     {
142         \bool_if:NT \l__zrefcheck_warn_hyperref_bool
143         { \msg_warning:nn { zref-check } { missing-hyperref } }
144         \bool_set_false:N \l__zrefcheck_use_hyperref_bool
145     }
146     \keys_define:nn { zref-check }
147     {
148         hyperref .code:n =
149         { \msg_warning:nn { zref-check } { hyperref-preamble-only } }
150     }
151 }

```

msglevel option

\l__zrefcheck_msglevel_tl

```

152 \tl_new:N \l__zrefcheck_msglevel_tl
153 \keys_define:nn { zref-check }
154 {
155     msglevel .choice: ,
156     msglevel / warn .code:n =
157     { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } } ,
158     msglevel / info .code:n =
159     { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } } ,
160     msglevel / none .code:n =
161     { \tl_set:Nn \l__zrefcheck_msglevel_tl { none } } ,
162     msglevel / infoifdraft .code:n =
163     {
164         \ifdraft
165         { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
166         { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
167     } ,
168     msglevel / warniffinal .code:n =
169     {
170         \ifoptionfinal
171         { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
172         { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
173     } ,
174     msglevel / obeydraft .code:n =
175     {
176         % NOTE Option value deprecated in 2021-12-07 for v0.2.2.
177         \msg_warning:nnnn { zref-check } { option-deprecated }
178         { msglevel=obeydraft } { msglevel=infoifdraft }
179     } ,
180     msglevel / obeyfinal .code:n =
181     {
182         % NOTE Option value deprecated in 2021-12-07 for v0.2.2.
183         \msg_warning:nnnn { zref-check } { option-deprecated }
184         { msglevel=obeyfinal } { msglevel=warniffinal }
185     } ,
186     msglevel .value_required:n = true ,
187     msglevel .initial:n = warn ,

```

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

```
188     ignore .code:n =
189       { \msg_warning:nn { zref-check } { ignore-document-only } } ,
190     ignore .value_forbidden:n = true
191   }
```

(End definition for `\l__zrefcheck_msglevel_tl`.)

```
192 \AddToHook { begindocument }
193 {
194   \keys_define:nn { zref-check }
195     { ignore .meta:n = { msglevel = none } }
196 }
```

onpage option

`\l__zrefcheck_msgonpage_bool`

```
197 \bool_new:N \l__zrefcheck_msgonpage_bool
198 \keys_define:nn { zref-check }
199 {
200   onpage .choice: ,
201   onpage / labelseq .code:n =
202     {
203       \bool_set_false:N \l__zrefcheck_msgonpage_bool
204     } ,
205   onpage / msg .code:n =
206     {
207       \bool_set_true:N \l__zrefcheck_msgonpage_bool
208     } ,
209   onpage / labelseqifdraft .code:n =
210     {
211       \ifdraft
212         { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
213         { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
214     } ,
215   onpage / msgiffinal .code:n =
216     {
217       \ifoptionfinal
218         { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
219         { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
220     } ,
221   onpage / obeydraft .code:n =
222     {
223       % NOTE Option value deprecated in 2021-12-07 for v0.2.2.
224       \msg_warning:nnnn { zref-check }{ option-deprecated }
225       { onpage=obeydraft } { onpage=labelseqifdraft }
226     } ,
227   onpage / obeyfinal .code:n =
228     {
229       % NOTE Option value deprecated in 2021-12-07 for v0.2.2.
230       \msg_warning:nnnn { zref-check }{ option-deprecated }
231       { onpage=obeyfinal } { onpage=msgiffinal }
232     } ,
233   onpage .value_required:n = true ,
```

```

234     onpage .initial:n = labelseq
235 }

```

(End definition for \l__zrefcheck_msgonpage_bool.)

closerange option

\l__zrefcheck_close_range_int

```

236 \int_new:N \l__zrefcheck_close_range_int
237 \keys_define:nn { zref-check }
238 {
239     closerange .code:n =
240     {
241         \__zrefcheck_is_integer_rgx:nTF {#1}
242         { \int_set:Nn \l__zrefcheck_close_range_int { \int_eval:n {#1} } }
243         {
244             \msg_warning:nn { zref-check } { closerange-not-positive-integer }
245             \int_set:Nn \l__zrefcheck_close_range_int { 5 }
246         }
247     } ,
248     closerange .value_required:n = true ,
249     closerange .initial:n = 5
250 }

```

(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>.

```

251 \tl_new:N \l__zrefcheck_target_label_tl
252 \bool_new:N \l__zrefcheck_target_label_bool
253 \keys_define:nn { zref-check }
254 {
255     labelcmd .code:n =
256     {
257         \tl_set:Nn \l__zrefcheck_target_label_tl {#1}
258         \bool_set_true:N \l__zrefcheck_target_label_bool
259     } ,
260     labelcmd .value_required:n = true ,
261 }

```

(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.

```

262 \cs_new_protected:Npn \__zrefcheck_target_label:n #1
263 { \zref@label {#1} }

```

(End definition for __zrefcheck_target_label:n.)

```

264 \AddToHook { begindocument }
265 {
266     \bool_if:NT \l__zrefcheck_target_label_bool
267     {

```



```

268 \tl_if_blank:VT \l__zrefcheck_target_label_tl
269 { \tl_clear:N \l__zrefcheck_target_label_tl }
270 \cs_if_exist:cTF { \l__zrefcheck_target_label_tl }
271 {
272   \cs_set_protected:Npx \__zrefcheck_target_label:n #1
273   {
274     \exp_not:o
275     { \cs:w \l__zrefcheck_target_label_tl \cs_end: }
276     {#1}
277   }
278 }
279 {
280   \exp_args:Nnno \msg_warning:nnn { zref-check }
281   { labelcmd-undefined } { \l__zrefcheck_target_label_tl }
282 }
283 }
284 \keys_define:nn { zref-check }
285 {
286   labelcmd .code:n =
287   {
288     \msg_warning:nnn { zref-check }
289     { option-preamble-only } { labelcmd }
290   }
291 }
292 }

```

Package options

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

```

293 \RequirePackage { l3keys2e }
294 \ProcessKeysOptions { zref-check }

```

`\zrefchecksetup` Provide `\zrefchecksetup`.

```

295 \NewDocumentCommand \zrefchecksetup { m }
296 { \keys_set:nn { zref-check } {#1} }

```

(End definition for \zrefchecksetup.)

4.4 Position on page

Method for determining relative position within the page: the sequence in which the labels get shipped out, inferred from the sequence in which the labels occur in the `.aux` file.

Some relevant info about the sequence of things: <https://tex.stackexchange.com/a/120978> and `texdoc lthooks`, section “Hooks provided by `\begin{document}`”.

One first attempt at this was to use `\zref@newlabel`, which is the macro in which `zref` stores the label information in the aux file. When the `.aux` file is read at the beginning of the compilation, this macro is expanded for each of the labels. So, by redefining this macro we can feed a variable (a L3 sequence), and then do what it usually does, which is to define each label with the internal macro `\@newl@bel`, when the `.aux` file is read.

Patching this macro for this is not possible. First, `\zref@newlabel` is one of those “commands that look ahead” mentioned in `ltxcmdhooks` documentation. Indeed,

`\@newl@bel` receives 3 arguments, and `\zref@newlabel` just passes the first, the following two will be scanned ahead. Second, the `\lcmdhooks` hooks are not actually available when the `.aux` file is read, they come only after `\begin{document}`. Hence, re-definition 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

```
297 \prop_new:N \g_zrefcheck_auxfile_lblseq_prop
```

(End definition for \g_zrefcheck_auxfile_lblseq_prop.)

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

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

```
300 {
```

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.

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

```
302 \group_begin:
```

```
303 \int_zero:N \l_tmpa_int
```

```
304 \tl_clear:N \l_tmpa_tl
```

```
305 \tl_clear:N \l_tmpb_tl
```

```
306 \bool_set_false:N \l_tmpa_bool
```

```
307 \ior_map_variable:NNn \g_tmpa_ior \l_tmpa_tl
```

```
308 {
```

```
309 \tl_map_variable:NNn \l_tmpa_tl \l_tmpb_tl
```

```
310 {
```

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

```
312 {
```

Found a \zref@label, signal it.

```
313 \bool_set_true:N \l_tmpa_bool
```

```
314 }
```

```
315 {
```

```
316 \bool_if:NNTF \l_tmpa_bool
```

```
317 {
```

```
318 \bool_set_false:N \l_tmpa_bool
```

```
319 \int_incr:N \l_tmpa_int
```

```
320 \prop_gput:Nxx \g_zrefcheck_auxfile_lblseq_prop
```

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

```
322 }
```

```
323 {
```

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.

```
324 \tl_map_break:
```

```
325 }
```

```
326 }
```

```
327 }
```

```
328 }
```

```
329 \group_end:
```

```
330 \ior_close:N \g_tmpa_ior
```

```
331 }
```

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.

```

332 \begingroup
333   \let \@addtoreset \ltx@gobbletwo
334   \newcounter { zrefcheck }
335 \endgroup
336 \setcounter { zrefcheck } { 0 }
```

4.6 Label formats

```

\__zrefcheck_check_lblfmt:n      \__zrefcheck_check_lblfmt:n {<check id int>}
337 \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>}
338 \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}

339 \cs_new_protected:Npn \zrefcheck_get_astl:nnn #1#2#3
340 {
341   \tl_clear:N #3
342   \tl_if_eq:nnTF {#2} { lblseq }
343   {
344     \prop_get:NnNF \g__zrefcheck_auxfile_lblseq_prop {#1} #3
345     {
346       \msg_warning:nnnn { zref-check }
347       { property-not-in-label } {#1} {#2}
348     }
349   }
350 {

```

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.

```

351   \zref@ifrefundefined {#1}
352   {}
353   {
354     \zref@ifpropundefined {#2}
355     { \msg_warning:nnnn { zref-check } { property-undefined } {#2} }
356     {
357       \zref@ifrefcontainsprop {#1} {#2}
358       {
359         \tl_set:Nx #3
360         { \zref@extractdefault {#1} {#2} { \c_empty_tl } }
361       }
362       {
363         \msg_warning:nnnn
364         { zref-check } { property-not-in-label } {#1} {#2}
365       }
366     }
367   }
368 }
369 }

```

(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).

```
370 \bool_new:N \l__zrefcheck_integer_bool
```

(End definition for `\l__zrefcheck_integer_bool`.)

```
\l__zrefcheck_propval_tl
```

```
371 \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}
```

```
372 \cs_new_protected:Npn \zrefcheck_get_asint:nnn #1#2#3
```

```
373 {
```

```
374   \zrefcheck_get_astl:nnn {#1} {#2} { \l__zrefcheck_propval_tl }
```

```
375   \__zrefcheck_is_integer:nTF { \l__zrefcheck_propval_tl }
```

```
376   {
```

Make it an integer data type.

```
377     \int_set:Nn #3 { \int_eval:n { \l__zrefcheck_propval_tl } }
```

```
378   }
```

```
379 {
```

```
380   \bool_set_false:N \l__zrefcheck_integer_bool
```

```
381   \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`.

```
382   { }
```

```
383   {
```

```
384     \msg_warning:nnnn { zref-check }
```

```
385     { property-not-integer } {#2} {#1}
```

```
386   }
```

```
387 }
```

```
388 }
```

(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}
```

```

389 \NewDocumentCommand \zcheck { s O { } m m }
390 { \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
391 \seq_new:N \l_zrefcheck_zcheck_labels_seq
392 \int_new:N \g_zrefcheck_id_int
393 \tl_new:N \l_zrefcheck_checkbeg_tl
394 \tl_new:N \l_zrefcheck_link_label_tl
395 \tl_new:N \l_zrefcheck_link_anchor_tl
396 \bool_new:N \l_zrefcheck_link_star_bool

```

(End definition for \l_zrefcheck_zcheck_labels_seq and others.)

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

```

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

```

```

397 \cs_new_protected:Npn \_zrefcheck_zcheck:nnnn #1#2#3#4
398 {
399   \group_begin:

```

Process local options and checks.

```

400   \keys_set:nn { zref-check / zcheck } {#3}
401   \seq_set_from_clist:Nn \l_zrefcheck_zcheck_labels_seq {#1}

```

Names of the labels for this zcheck call.

```

402   \int_gincr:N \g_zrefcheck_id_int
403   \tl_set:Nx \l_zrefcheck_checkbeg_tl
404   { \_zrefcheck_check_lblfmt:n { \g_zrefcheck_id_int } }

```

Set checkbeg label.

```

405   \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.

```

406   \seq_get:NN \l_zrefcheck_zcheck_labels_seq \l_zrefcheck_link_label_tl
407   \bool_set:Nn \l_zrefcheck_link_star_bool {#2}
408   \zref@ifrefundefined { \l_zrefcheck_link_label_tl }

```

If the reference is undefined, just typeset.

```

409   {#4}
410   {
411     \bool_if:nTF
412     {
413       \l_zrefcheck_use_hyperref_bool &&
414       ! \l_zrefcheck_link_star_bool
415     }
416     {
417       \exp_args:Nx \zrefcheck_get_astl:nnn
418       { \l_zrefcheck_link_label_tl }

```

```

419         { anchor } { \l__zrefcheck_link_anchor_tl }
420         \hyperlink { \l__zrefcheck_link_anchor_tl } {#4}
421     }
422     {#4}
423 }

```

Set checkend label.

```

424     \bool_if:NT \l__zrefcheck_zcheck_end_label_bool
425     {
426         \zref@labelbylist
427         { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
428         { zrefcheck-end }
429     }

```

Check if *(labels)* are defined.

```

430     \seq_map_function:NN \l__zrefcheck_zcheck_labels_seq \zref@refused

```

Run the checks.

```

431     \__zrefcheck_run_checks:nxx { \l__zrefcheck_zcheck_checks_seq }
432     { \l__zrefcheck_zcheck_labels_seq } { \l__zrefcheck_checkbeg_tl }
433     \group_end:
434 }

```

(End definition for `__zrefcheck_zcheck:nnnn`.)

5.2 Targets

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

```

435 \NewDocumentCommand \zctarget { m +m }
436 {

```

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.

```

437     \group_begin:
438     \refstepcounter { zrefcheck }
439     \zref@wrapper@babel \__zrefcheck_target_label:n {#1}
440     #2
441     \tl_if_empty:nF {#2}
442     {
443         \zref@wrapper@babel
444         \zref@labelbylist { \__zrefcheck_end_lblfmt:n {#1} } { zrefcheck-end }
445     }
446     \group_end:
447 }

```

(End definition for `\zctarget`.)

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

```

448 \NewDocumentEnvironment {zcregion} { m }
449 {
450     \refstepcounter { zrefcheck }
451     \zref@wrapper@babel \__zrefcheck_target_label:n {#1}

```



```

452 }
453 {
454   \zref@wrapper@babel
455   \zref@labelbylist { \_zrefcheck\_end\_lblfmt:n {#1} } { zrefcheck-end }
456 }

```

(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 `<reference>`. That is, the “before” check should return true if the `<label>` occurs before the “reference”.

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

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

Third, the check must be registered as a key which gets setup in `\zcheck` by the `zref-check / zcheck` key set.

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

6.1 Single label checks

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

`\c_zrefcheck_single_label_checks_seq`

```

457 \seq_new:N \c\_zrefcheck\_single\_label\_checks\_seq
458 \seq_set_from_clist:Nn \c\_zrefcheck\_single\_label\_checks\_seq
459 {
460   thischap ,
461   prevchap ,
462   nextchap ,
463   chapsbefore ,
464   chapsafter ,
465   thissec ,

```

```

466     prevsec ,
467     nextsec ,
468     secsbefore ,
469     secsafter ,
470 }

```

(End definition for \c__zrefcheck_single_label_checks_seq.)

6.2 Setup

```

\l__zrefcheck_zcheck_checks_seq
\l__zrefcheck_end_label_required_bool

```

```

471 \seq_new:N \l__zrefcheck_zcheck_checks_seq
472 \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.

```

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

```

Then we add the checks to it.

```

474 \clist_map_inline:nn
475 {
476     thispage ,
477     prevpage ,
478     nextpage ,
479     facing ,
480     pagegap ,
481     above ,
482     below ,
483     pagesbefore ,
484     ppbefore ,
485     pagesafter ,
486     ppafter ,
487     before ,
488     after ,
489     thischap ,
490     prevchap ,
491     nextchap ,
492     chapsbefore ,
493     chapsafter ,
494     thissec ,
495     prevsec ,
496     nextsec ,
497     secsbefore ,
498     secsafter ,
499     close ,
500     far ,
501 }
502 {
503     \keys_define:nn { zref-check / zcheck }
504     {
505         #1 .code:n =
506         {
507             \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq {#1}
508             \seq_if_in:NnF \c__zrefcheck_single_label_checks_seq {#1}

```

```

509         { \bool_set_true:N \l__zrefcheck_zcheck_end_label_bool }
510     } ,
511     #1 .value_forbidden:n = true ,
512 }
513 }

```

6.3 Running

`__zrefcheck_run_checks:nnn` `__zrefcheck_run_checks:nnn {⟨checks⟩} {⟨labels⟩} {⟨reference⟩}`
`⟨checks⟩` are expected to be received as a sequence variable.

```

514 \cs_new_protected:Npn \__zrefcheck_run_checks:nnn #1#2#3
515 {
516     \group_begin:
517     \seq_map_inline:Nn #2
518     {
519         \seq_map_inline:Nn #1
520         { \__zrefcheck_do_check:nnn {####1} {##1} {#3} }
521     }
522     \group_end:
523 }
524 \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
525 \bool_new:N \l__zrefcheck_passedcheck_bool
526 \bool_new:N \l__zrefcheck_onpage_bool
527 \seq_new:N \c__zrefcheck_onpage_checks_seq
528 \seq_set_from_clist:Nn \c__zrefcheck_onpage_checks_seq
529 { 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`.

```

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

```

`__zrefcheck_do_check:nnn` `__zrefcheck_do_check:nnn {⟨check⟩} {⟨label beg⟩} {⟨reference beg⟩}`

```

531 \cs_new_protected:Npn \__zrefcheck_do_check:nnn #1#2#3
532 {
533     \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.

```

534     \zref@ifrefundefined {#2}
535     {}
536     {
537         \tl_if_empty:nF {#1}
538         {
539             \bool_set_true:N \l__zrefcheck_passedcheck_bool
540             \bool_set_false:N \l__zrefcheck_onpage_bool
541             \cs_if_exist:cTF { __zrefcheck_check_ #1 :nnF }

```

```

542 {
543   % ‘‘label beg’’ vs ‘‘reference beg’’.
544   \use:c { __zrefcheck_check_ #1 :nnF }
545   {#2} {#3}
546   { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
547   % ‘‘reference end’’ \emph{may} exist or not depending on the
548   % checks.
549   \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#3} }
550   {
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     }
561   }
562   {
563     % ‘‘label beg’’ vs ‘‘reference end’’.
564     \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
565     {#2} { \__zrefcheck_end_lblfmt:n {#3} }
566     { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
567     % ‘‘label end’’ \emph{may} have been created by the
568     % target commands.
569     \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
570     {}
571     {
572       % ‘‘label end’’ vs ‘‘reference beg’’.
573       \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
574       { \__zrefcheck_end_lblfmt:n {#2} } {#3}
575       { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
576       % ‘‘label end’’ vs ‘‘reference end’’.
577       \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
578       { \__zrefcheck_end_lblfmt:n {#2} }
579       { \__zrefcheck_end_lblfmt:n {#3} }
580       { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
581     }
582   }

```

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.

```

583 \seq_if_in:NnT \c__zrefcheck_onpage_checks_seq {#1}
584 {
585   \__zrefcheck_check_thispage:nnT
586   {#2} {#3}
587   { \bool_set_true:N \l__zrefcheck_onpage_bool }
588   \__zrefcheck_check_thispage:nnT
589   {#2} { \__zrefcheck_end_lblfmt:n {#3} }
590   { \bool_set_true:N \l__zrefcheck_onpage_bool }
591   \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }

```

```

592         {}
593     {
594         \__zrefcheck_check_thispage:nnT
595         { \__zrefcheck_end_lblfmt:n {#2} } {#3}
596         { \bool_set_true:N \l__zrefcheck_onpage_bool }
597         \__zrefcheck_check_thispage:nnT
598         { \__zrefcheck_end_lblfmt:n {#2} }
599         { \__zrefcheck_end_lblfmt:n {#3} }
600         { \bool_set_true:N \l__zrefcheck_onpage_bool }
601     }
602 }
603 \bool_if:NTF \l__zrefcheck_passedcheck_bool
604 {
605     \bool_if:nT
606     {
607         \l__zrefcheck_msgonpage_bool &&
608         \l__zrefcheck_onpage_bool
609     }
610     {
611         \__zrefcheck_message:nnnx { double-check } {#1} {#2}
612         { \zref@extractdefault {#3} {page} {'unknown'} }
613     }
614 }
615 {
616     \__zrefcheck_message:nnnx { check-failed } {#1} {#2}
617     { \zref@extractdefault {#3} {page} {'unknown'} }
618 }
619 }
620 { \msg_warning:nnn { zref-check } { check-missing } {#1} }
621 }
622 }
623 \group_end:
624 }

```

(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.

```

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

```

(End definition for \l__zrefcheck_lbl_int and others.)

6.4.1 This page

```

\__zrefcheck_check_thispage:nn

```

```

629 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thispage:nn #1#2 { T , F , TF }
630 {
631     \group_begin:
632     \bool_set_true:N \l__zrefcheck_integer_bool
633     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }

```

```

634 \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
635 \bool_lazy_and:nnTF
636 { \l__zrefcheck_integer_bool }
637 {
638   \int_compare_p:nNn
639   { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int } &&

```

‘0’ is the default value of `abspage`, but this value should not happen normally for this property, since even the first page, after it gets shipped out, will receive value ‘1’. So, if we do find ‘0’ here, better signal something is wrong. This comment extends to all page number checks.

```

640   ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
641   ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
642 }
643 { \group_insert_after:N \prg_return_true: }
644 { \group_insert_after:N \prg_return_false: }
645 \group_end:
646 }

```

(End definition for `__zrefcheck_check_thispage:nn`.)

6.4.2 On page

```

\__zrefcheck_check_above:nn
\__zrefcheck_check_below:nn
647 \prg_new_protected_conditional:Npnn \__zrefcheck_check_above:nn #1#2 { F , TF }
648 {
649   \group_begin:
650   \__zrefcheck_check_thispage:nnTF {#1} {#2}
651   {
652     \bool_set_true:N \l__zrefcheck_integer_bool
653     \zrefcheck_get_asint:nnn {#1} { lblseq } { \l__zrefcheck_lbl_int }
654     \zrefcheck_get_asint:nnn {#2} { lblseq } { \l__zrefcheck_ref_int }
655     \bool_lazy_and:nnTF
656     { \l__zrefcheck_integer_bool }
657     {
658       \int_compare_p:nNn
659       { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
660       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
661       ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
662     }
663     { \group_insert_after:N \prg_return_true: }
664     { \group_insert_after:N \prg_return_false: }
665   }
666   { \group_insert_after:N \prg_return_false: }
667   \group_end:
668 }
669 \prg_new_protected_conditional:Npnn \__zrefcheck_check_below:nn #1#2 { F , TF }
670 {
671   \__zrefcheck_check_thispage:nnTF {#1} {#2}
672   {
673     \__zrefcheck_check_above:nnTF {#1} {#2}
674     { \prg_return_false: }
675     { \prg_return_true: }
676   }

```

```

677     { \prg_return_false: }
678 }

```

(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
679 \prg_new_protected_conditional:Npnn \_zrefcheck\_check\_before:nn #1#2 { F }
680 {
681   \_zrefcheck\_check\_pagesbefore:nnTF {#1} {#2}
682   { \prg_return_true: }
683   {
684     \_zrefcheck\_check\_above:nnTF {#1} {#2}
685     { \prg_return_true: }
686     { \prg_return_false: }
687   }
688 }
689 \prg_new_protected_conditional:Npnn \_zrefcheck\_check\_after:nn #1#2 { F }
690 {
691   \_zrefcheck\_check\_pagesafter:nnTF {#1} {#2}
692   { \prg_return_true: }
693   {
694     \_zrefcheck\_check\_below:nnTF {#1} {#2}
695     { \prg_return_true: }
696     { \prg_return_false: }
697   }
698 }

```

(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
699 \prg_new_protected_conditional:Npnn \_zrefcheck\_check\_nextpage:nn #1#2 { F }
700 {
701   \group_begin:
702     \bool_set_true:N \l__zrefcheck_integer_bool
703     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
704     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
705     \bool_lazy_and:nnTF
706       { \l__zrefcheck_integer_bool }
707       {
708         \int_compare_p:nNn
709           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
710           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
711           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
712       }
713     { \group_insert_after:N \prg_return_true: }
714     { \group_insert_after:N \prg_return_false: }
715   \group_end:
716 }
717 \prg_new_protected_conditional:Npnn \_zrefcheck\_check\_prevpage:nn #1#2 { F }
718 {

```

```

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

```



```

773 \prg_new_protected_conditional:Npnn \__zrefcheck_check_pagegap:nn #1#2 { F }
774 {
775   \group_begin:
776   \bool_set_true:N \l__zrefcheck_integer_bool
777   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
778   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
779   \bool_lazy_and:nnTF
780   { \l__zrefcheck_integer_bool }
781   {
782     \int_compare_p:nNn
783     { \int_abs:n { \l__zrefcheck_lbl_int - \l__zrefcheck_ref_int } } > { 1 } &&
784     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
785     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
786   }
787   { \group_insert_after:N \prg_return_true: }
788   { \group_insert_after:N \prg_return_false: }
789   \group_end:
790 }
791 \prg_new_protected_conditional:Npnn \__zrefcheck_check_facing:nn #1#2 { F }
792 {
793   \group_begin:
794   \bool_set_true:N \l__zrefcheck_integer_bool
795   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
796   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
797   \bool_lazy_and:nnTF
798   { \l__zrefcheck_integer_bool }
799   {

```

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

```

800   \legacy_if_p:n { @twoside } &&

```

Now we test “facing”.

```

801   (
802     (
803       \int_if_odd_p:n { \l__zrefcheck_ref_int } &&
804       \int_compare_p:nNn
805       { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 }
806     ) ||
807     (
808       \int_if_even_p:n { \l__zrefcheck_ref_int } &&
809       \int_compare_p:nNn
810       { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 }
811     )
812   ) &&
813   ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
814   ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
815 }
816 { \group_insert_after:N \prg_return_true: }
817 { \group_insert_after:N \prg_return_false: }
818 \group_end:
819 }

```

(End definition for __zrefcheck_check_nextpage:nn and others.)

6.4.5 Close / Far

```

\__zrefcheck_check_close:nn
\__zrefcheck_check_far:nn
820 \prg_new_protected_conditional:Npnn \__zrefcheck_check_close:nn #1#2 { F , TF }
821 {
822   \group_begin:
823     \bool_set_true:N \l__zrefcheck_integer_bool
824     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
825     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
826     \bool_lazy_and:nnTF
827       { \l__zrefcheck_integer_bool }
828       {
829         \int_compare_p:nNn
830           { \int_abs:n { \l__zrefcheck_lbl_int - \l__zrefcheck_ref_int } }
831           <
832           { \l__zrefcheck_close_range_int + 1 } &&
833           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
834           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
835       }
836       { \group_insert_after:N \prg_return_true: }
837       { \group_insert_after:N \prg_return_false: }
838   \group_end:
839 }
840 \prg_new_protected_conditional:Npnn \__zrefcheck_check_far:nn #1#2 { F }
841 {
842   \__zrefcheck_check_close:nnTF {#1} {#2}
843   { \prg_return_false: }
844   { \prg_return_true: }
845 }

```

(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
846 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thischap:nn #1#2 { F }
847 {
848   \group_begin:
849     \bool_set_true:N \l__zrefcheck_integer_bool
850     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
851     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
852     \bool_lazy_and:nnTF
853       { \l__zrefcheck_integer_bool }
854       {
855         \int_compare_p:nNn
856           { \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.

```

857       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
858       ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }

```

```

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

```

```

913         { \group_insert_after:N \prg_return_false: }
914     \group_end:
915 }
916 \prg_new_protected_conditional:Npnn \__zrefcheck_check_chapsbefore:nn #1#2 { F }
917 {
918     \group_begin:
919     \bool_set_true:N \l__zrefcheck_integer_bool
920     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
921     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
922     \bool_lazy_and:nnTF
923     { \l__zrefcheck_integer_bool }
924     {
925         \int_compare_p:nNn
926         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
927         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
928         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
929     }
930     { \group_insert_after:N \prg_return_true: }
931     { \group_insert_after:N \prg_return_false: }
932 \group_end:
933 }

```

(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
934 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thissec:nn #1#2 { F }
935 {
936     \group_begin:
937     \bool_set_true:N \l__zrefcheck_integer_bool
938     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
939     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
940     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
941     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
942     \bool_lazy_and:nnTF
943     { \l__zrefcheck_integer_bool }
944     {
945         \int_compare_p:nNn
946         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
947         \int_compare_p:nNn
948         { \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.

```

949         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
950         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
951     }
952     { \group_insert_after:N \prg_return_true: }
953     { \group_insert_after:N \prg_return_false: }
954 \group_end:

```

```

955 }
956 \prg_new_protected_conditional:Npnn \__zrefcheck_check_nextsec:nn #1#2 { F }
957 {
958   \group_begin:
959     \bool_set_true:N \l__zrefcheck_integer_bool
960     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
961     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
962     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
963     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
964     \bool_lazy_and:nnTF
965       { \l__zrefcheck_integer_bool }
966       {
967         \int_compare_p:nNn
968           { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
969         \int_compare_p:nNn
970           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
971         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
972       }
973     { \group_insert_after:N \prg_return_true: }
974     { \group_insert_after:N \prg_return_false: }
975   \group_end:
976 }
977 \prg_new_protected_conditional:Npnn \__zrefcheck_check_prevsec:nn #1#2 { F }
978 {
979   \group_begin:
980     \bool_set_true:N \l__zrefcheck_integer_bool
981     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
982     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
983     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
984     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
985     \bool_lazy_and:nnTF
986       { \l__zrefcheck_integer_bool }
987       {
988         \int_compare_p:nNn
989           { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
990         \int_compare_p:nNn
991           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
992         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
993         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
994       }
995     { \group_insert_after:N \prg_return_true: }
996     { \group_insert_after:N \prg_return_false: }
997   \group_end:
998 }
999 \prg_new_protected_conditional:Npnn \__zrefcheck_check_secsafter:nn #1#2 { F }
1000 {
1001   \group_begin:
1002     \bool_set_true:N \l__zrefcheck_integer_bool
1003     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
1004     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
1005     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
1006     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
1007     \bool_lazy_and:nnTF
1008       { \l__zrefcheck_integer_bool }

```

```

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

```

(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
1042 \cs_new_protected:Npn \zrefcheck_zceref_beg_label:
1043 {
1044     \int_gincr:N \g__zrefcheck_id_int
1045     \tl_set:Nx \l__zrefcheck_checkbeg_tl
1046     { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
1047     \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrefcheck-check }
1048 }
1049 \cs_new_protected:Npn \zrefcheck_zceref_end_label_maybe:

```

```

1050 {
1051   \bool_if:NT \l__zrefcheck_zcheck_end_label_bool
1052   {
1053     \zref@labelbylist
1054     { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
1055     { zrefcheck-end }
1056   }
1057 }
1058 \cs_new_protected:Npn \zrefcheck_zcref_run_checks_on_labels:n #1
1059 {
1060   \__zrefcheck_run_checks:nnx
1061   { \l__zrefcheck_zcheck_checks_seq } {#1} { \l__zrefcheck_checkbeg_tl }
1062 }

```

(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 ??.)

```

1063 </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	C
<code>\</code> 68	<code>\catcode</code> 11
	<code>\chapter</code> 2, 26
	clist commands:
	<code>\clist_map_inline:nn</code> 474
	cs commands:
	<code>\cs:w</code> 275
	<code>\cs_end:</code> 275
	<code>\cs_generate_variant:Nn</code> . 52, 524, 530
	<code>\cs_if_exist:NTF</code> 270, 541
	<code>\cs_new:Npn</code> 337, 338
	<code>\cs_new_eq:NN</code> 94, 753, 772
	<code>\cs_new_protected:Npn</code> 47, 262, 339,
	372, 397, 514, 531, 1042, 1049, 1058
	<code>\cs_set_protected:Npx</code> 272
	D
	<code>\d</code> 107
	E
	<code>\emph</code> 547, 551, 567
	<code>\endgroup</code> 335
	<code>\endinput</code> 12
	exp commands:
	<code>\exp_args:Nnno</code> 280, 530, 564
	<code>\exp_args:Nno</code> 557, 573
	<code>\exp_args:Nnoo</code> 577
	<code>\exp_args:Nx</code> 417
	<code>\exp_not:n</code> 274
A	
<code>\A</code> 107	
<code>\AddToHook</code> 21, 28, 134, 192, 264	
B	
<code>\begingroup</code> 332	
bool commands:	
<code>\bool_if:NTF</code> 138, 142, 266, 316, 424, 603, 1051	
<code>\bool_if:nTF</code> 411, 605	
<code>\bool_lazy_and:nnTF</code> 14, 635, 655, 705, 723, 741, 760, 779, 797, 826, 852, 870, 887, 905, 922, 942, 964, 985, 1007, 1028	
<code>\bool_new:N</code> 111, 112, 197, 252, 370, 396, 472, 525, 526	
<code>\bool_set:Nn</code> 407	
<code>\bool_set_false:N</code> 119, 128, 129, 144, 203, 212, 219, 306, 318, 380, 540, 546, 559, 566, 575, 580	
<code>\bool_set_true:N</code> 118, 123, 124, 207, 213, 218, 258, 313, 509, 539, 587, 590, 596, 600, 632, 652, 702, 720, 738, 757, 776, 794, 823, 849, 867, 884, 902, 919, 937, 959, 980, 1002, 1023	
<code>\l_tmpa_bool</code> ... 11, 306, 313, 316, 318	

F	
file commands:	
\file_if_exist:nTF	299
\fmtversion	3
G	
group commands:	
\group_begin:	302, 399, 437, 516, 533, 631, 649, 701, 719, 737, 756, 775, 793, 822, 848, 866, 883, 901, 918, 936, 958, 979, 1001, 1022
\group_end:	329, 433, 446, 522, 623, 645, 667, 715, 733, 751, 770, 789, 818, 838, 862, 879, 897, 914, 932, 954, 975, 997, 1018, 1040
\group_insert_after:N	643, 644, 663, 664, 666, 713, 714, 731, 732, 749, 750, 768, 769, 787, 788, 816, 817, 836, 837, 860, 861, 877, 878, 895, 896, 912, 913, 930, 931, 952, 953, 973, 974, 995, 996, 1016, 1017, 1038, 1039
H	
\hyperlink	14, 420
I	
\ifdraft	164, 211
\IfFormatAtLeastTF	3, 4
\ifoptionfinal	170, 217
int commands:	
\int_abs:n	783, 830
\int_compare_p:nNn	638, 640, 641, 658, 660, 661, 708, 710, 711, 726, 728, 729, 744, 746, 747, 763, 765, 766, 782, 784, 785, 804, 809, 813, 814, 829, 833, 834, 855, 857, 858, 873, 875, 890, 892, 893, 908, 910, 925, 927, 928, 945, 947, 949, 950, 967, 969, 971, 988, 990, 992, 993, 1010, 1012, 1014, 1031, 1033, 1035, 1036
\int_eval:n	242, 377
\int_gincr:N	23, 29, 402, 1044
\int_if_even_p:n	808
\int_if_odd_p:n	803
\int_incr:N	319
\int_new:N	19, 20, 236, 392, 625, 626, 627, 628
\int_set:Nn	242, 245, 377
\int_use:N	26, 30, 321, 337
\int_zero:N	24, 303
\l_tmpa_int	303, 319, 321
int internal commands:	
__int_to_roman:w	4, 5, 94
ior commands:	
\ior_close:N	330
\ior_map_variable:NNn	307
\ior_open:Nn	301
\g_tmpa_ior	301, 307, 330
iow commands:	
\iow_char:N	68
\iow_newline:	67, 71, 74, 91
K	
keys commands:	
\keys_define:nn	113, 146, 153, 194, 198, 237, 253, 284, 473, 503
\keys_set:nn	296, 400
L	
\label	12
legacy commands:	
\legacy_if_p:n	800
\let	333
M	
\MessageBreak	10
msg commands:	
\msg_line_context:	54, 56, 58, 60, 62, 64, 78, 81
\msg_new:nnn	53, 55, 57, 59, 61, 63, 65, 70, 72, 77, 79, 84, 89
\msg_warning:nn	143, 149, 189, 244
\msg_warning:nnn	280, 288, 620
\msg_warning:nnnn	177, 183, 224, 230, 346, 355, 363, 384
N	
\newcounter	334
\NewDocumentCommand	295, 389, 435
\NewDocumentEnvironment	448
P	
\PackageError	7
\pageref	12
prg commands:	
\prg_new_conditional:Npnn	95
\prg_new_protected_conditional:Npnn	17, 105, 629, 647, 669, 679, 689, 699, 717, 735, 754, 773, 791, 820, 840, 846, 864, 881, 899, 916, 934, 956, 977, 999, 1020
\prg_return_false:	17, 98, 102, 109, 644, 664, 666, 674, 677, 686, 696, 714, 732, 750, 769, 788, 817, 837, 843, 861, 878, 896, 913, 931, 953, 974, 996, 1017, 1039
\prg_return_true:	17, 101, 108, 643, 663, 675, 682, 685, 692, 695, 713, 731, 749,

768, 787, 816, 836, 844, 860, 877, 895, 912, 930, 952, 973, 995, 1016, 1038	
\ProcessKeysOptions	294
prop commands:	
\prop_get:NnNTF	344
\prop_gput:Nnn	320
\prop_new:N	297
\providecommand	3
\ProvidesExplPackage	14
R	
\refstepcounter	12, 16, 438, 450
regex commands:	
\regex_match:nnTF	107
\RequirePackage	16, 17, 18, 139, 293
\romannumeral	4
S	
\section	28
seq commands:	
\seq_get:NN	406
\seq_if_in:NnTF	508, 583
\seq_map_function:NN	430
\seq_map_inline:Nn	517, 519
\seq_new:N	391, 457, 471, 527
\seq_put_right:Nn	507
\seq_set_from_clist:Nn	401, 458, 528
\setcounter	336
sys commands:	
\c_sys_jobname_str	298
T	
T _E X and L ^A T _E X 2 _ε commands:	
\@addtoreset	333
\@currentlabel	16
\@ifl@t@r	3
\@ifpackageloaded	136
\@newl@bel	9, 10
\ltx@gobbletwo	333
\zref@addprop	17, 27, 31
\zref@addprops	33, 41
\zref@extractdefault	12, 360, 612, 617
\zref@ifpropundefined	354
\zref@ifrefcontainsprop	357
\zref@ifrefundefined	13, 351, 381, 408, 534, 549, 553, 569, 591
\ZREF@label	11
\zref@label	11, 263
\zref@labelbylist	405, 426, 444, 455, 1047, 1053
\ZREF@mainlist	27, 31
\zref@newlabel	9–11, 311
\zref@newlist	32, 40
\zref@newprop	17, 26, 30
\zref@refused	13, 430
\zref@require@unique	12
\zref@wrapper@babel	10, 15, 390, 439, 443, 451, 454
tl commands:	
\c_empty_tl	12, 360
\tl_clear:N	12, 269, 304, 305, 341
\tl_if_blank:nTF	4, 268
\tl_if_empty:nTF	4, 97, 100, 441, 537
\tl_if_eq:NnTF	311
\tl_if_eq:nnTF	342
\tl_map_break:	324
\tl_map_variable:NNn	309
\tl_new:N	152, 251, 371, 393, 394, 395
\tl_set:Nn	157, 159, 161, 165, 166, 171, 172, 257, 298, 359, 403, 1045
\g_tmpa_tl	298, 299, 301
\l_tmpa_tl	304, 307, 309
\l_tmpb_tl	305, 309, 311, 321
U	
use commands:	
\use:N	49, 544, 557, 564, 573, 577
Z	
\Z	107
\zcheck	14, 15, 17, 389
\zcref	30
zcregion	448
\zctarget	8, 16, 435
\zref	15
zrefcheck commands:	
\zrefcheck_get_asint:nnn	13, 14, 17, 372, 633, 634, 653, 654, 703, 704, 721, 722, 739, 740, 758, 759, 777, 778, 795, 796, 824, 825, 850, 851, 868, 869, 885, 886, 903, 904, 920, 921, 938, 939, 940, 941, 960, 961, 962, 963, 981, 982, 983, 984, 1003, 1004, 1005, 1006, 1024, 1025, 1026, 1027
\zrefcheck_get_astl:nnn	13, 17, 339, 374, 417
\zrefcheck_zcref_beg_label:	1042
\zrefcheck_zcref_end_label_- maybe:	1042
\zrefcheck_zcref_run_checks_on_- labels:n	1042
zrefcheck internal commands:	
\g__zrefcheck_abschap_int	19, 23, 26
\g__zrefcheck_abssec_int	19, 24, 29, 30
\g__zrefcheck_auxfile_lblseq_- prop	12, 17, 297, 320, 344
__zrefcheck_check_⟨check⟩:nn	17

_zrefcheck_check_above:nn	...	647	_g_zrefcheck_id_int	
_zrefcheck_check_above:nnTF		391 , 402 , 404 , 1044 , 1046
.....		673 , 684	_zrefcheck_int_to_roman:w	94
_zrefcheck_check_after:nn	...	679	_l_zrefcheck_integer_bool	
_zrefcheck_check_before:nn	...	679		14 , 370 , 380 , 632 , 636 , 652 , 656 , 702 , 706 , 720 , 724 , 738 , 742 , 757 , 761 , 776 , 780 , 794 , 798 , 823 , 827 , 849 , 853 , 867 , 871 , 884 , 888 , 902 , 906 , 919 , 923 , 937 , 943 , 959 , 965 , 980 , 986 , 1002 , 1008 , 1023 , 1029
_zrefcheck_check_below:nn	...	647	_zrefcheck_is_integer:n	5 , 94
_zrefcheck_check_below:nnTF	...	694	_zrefcheck_is_integer:nnTF	...	375
_zrefcheck_check_chapsafter:nn		846	_zrefcheck_is_integer_rgx:n	5 , 105	
_zrefcheck_check_chapsbefore:nn	846	_zrefcheck_is_integer_rgx:nnTF		241
_zrefcheck_check_close:nn	...	820	_l_zrefcheck_lbl_b_int	
_zrefcheck_check_close:nnTF	...	842		625 , 940 , 946 , 962 , 968 , 983 , 989 , 1005 , 1011 , 1026 , 1032
_zrefcheck_check_facing:nn	...	699	_l_zrefcheck_lbl_int	625 , 633 , 639 ,	
_zrefcheck_check_far:nn	820		640 , 653 , 659 , 660 , 703 , 709 , 710 ,	
_zrefcheck_check_lblfmt:n	12 , 337 , 404 , 1046		721 , 727 , 728 , 739 , 745 , 746 , 758 ,	
_zrefcheck_check_nextchap:nn	...	846		764 , 765 , 777 , 783 , 784 , 795 , 805 ,	
_zrefcheck_check_nextpage:nn	...	699		810 , 813 , 824 , 830 , 833 , 850 , 856 ,	
_zrefcheck_check_nextsec:nn	...	934		857 , 868 , 874 , 875 , 885 , 891 , 892 ,	
_zrefcheck_check_pagegap:nn	...	699		903 , 909 , 910 , 920 , 926 , 927 , 938 ,	
_zrefcheck_check_pagesafter:nn	699			948 , 949 , 960 , 970 , 971 , 981 , 991 ,	
_zrefcheck_check_pagesafter:nnTF	691 , 772		992 , 1003 , 1013 , 1014 , 1024 , 1034 , 1035	
_zrefcheck_check_pagesbefore:nn	699	_l_zrefcheck_link_anchor_tl	...	
_zrefcheck_check_pagesbefore:nnTF	681 , 753		391 , 419 , 420
_zrefcheck_check_ppafter:nn	...	699	_l_zrefcheck_link_label_tl	...	
_zrefcheck_check_ppafter:nnTF	...	772		391 , 406 , 408 , 418
_zrefcheck_check_ppbefore:nn	...	699	_l_zrefcheck_link_star_bool	...	
_zrefcheck_check_ppbefore:nnTF	...	753		391 , 407 , 414
_zrefcheck_check_prevchap:nn	...	846	_zrefcheck_message:nnnn	47 , 611 , 616	
_zrefcheck_check_prevpage:nn	...	699	_l_zrefcheck_msglevel_tl	...	49 , 152
_zrefcheck_check_prevsec:nn	...	934	_l_zrefcheck_msgonpage_bool	197 , 607	
_zrefcheck_check_secsafter:nn	934		_l_zrefcheck_onpage_bool	
_zrefcheck_check_secsbefore:nn	934			525 , 540 , 587 , 590 , 596 , 600 , 608
_zrefcheck_check_thischap:nn	...	846	_c_zrefcheck_onpage_checks_seq	.	
_zrefcheck_check_thispage:nn	...	629		525 , 583
_zrefcheck_check_thispage:nnTF	585 , 588 , 594 , 597 , 650 , 671	_l_zrefcheck_passedcheck_bool	...	
_zrefcheck_check_thissec:nn	...	934		525 , 539 , 546 , 559 , 566 , 575 , 580 , 603
_l_zrefcheck_checkbeg_tl	391 , 403 ,		_l_zrefcheck_propval_tl	
	405 , 427 , 432 , 1045 , 1047 , 1054 , 1061			371 , 374 , 375 , 377
_l_zrefcheck_close_range_int	...		_l_zrefcheck_ref_b_int	
.....		236 , 832		625 , 941 , 946 , 963 , 968 , 984 , 989 , 1006 , 1011 , 1027 , 1032
_zrefcheck_do_check:nnn	19 , 520 , 531		_l_zrefcheck_ref_int	625 ,
_l_zrefcheck_end_label_required_bool	471		634 , 639 , 641 , 654 , 659 , 661 , 704 , 709 , 711 , 722 , 727 , 729 , 740 , 745 , 747 , 759 , 764 , 766 , 778 , 783 , 785 , 796 , 803 , 805 , 808 , 810 , 814 , 825 , 830 , 834 , 851 , 856 , 858 , 869 , 874 , 886 , 891 , 893 , 904 , 909 , 921 , 926 ,
_zrefcheck_end_lblfmt:n	12 , 338 , 427 , 444 ,			
.....		455 , 549 , 553 , 558 , 565 , 569 , 574 ,			
.....		578 , 579 , 589 , 591 , 595 , 598 , 599 , 1054			
_zrefcheck_get_asint:nnn	12			
_zrefcheck_get_astl:nnn	12			

928, 939, 948, 950, 961, 970, 982, 111, 138, 144, 413
991, 993, 1004, 1013, 1025, 1034, 1036	
_zrefcheck_run_checks:nnn	\l_zrefcheck_warn_hyperref_bool
..... 19, 431, 514, 1060 111, 142
\c_zrefcheck_single_label-	_zrefcheck_zcheck:nnnn 15, 390, 397
checks_seq 17, 457, 508	_zrefcheck_zcheck:nnnnn 13, 14, 19
_zrefcheck_target_label:n	\l_zrefcheck_zcheck_checks_seq .
..... 3, 262, 272, 439, 451 431, 471, 507, 1061
\l_zrefcheck_target_label_bool .	\l_zrefcheck_zcheck_end_label-
..... 252, 258, 266	bool 424, 472, 509, 1051
\l_zrefcheck_target_label_tl ...	\l_zrefcheck_zcheck_labels_seq .
..... 251, 268, 269, 270, 275, 281 391, 401, 406, 430, 432
\l_zrefcheck_use_hyperref_bool .	\zrefchecksetup 9, 295