

The zref-check package*

Gustavo Barros[†]

2021-07-27

Contents

I	\zref-check implementation	1
1	Initial setup	2
2	Dependencies	2
3	zref setup	2
4	Plumbing	3
4.1	Messages	3
4.2	Options	4
4.3	Position on page	8
4.4	Counter	10
4.5	Label formats	10
4.6	Property values	11
5	User interface	14
5.1	\zcheck	14
5.2	Targets	15
6	Checks	16
6.1	Running	16
6.2	Conditionals	19
6.2.1	This page	19
6.2.2	On page	19
6.2.3	Before / After	20
6.2.4	Pages	20
6.2.5	Close / Far	23
6.2.6	Chapter	23
6.2.7	Section	25

*This file describes v0.1.0-alpha, last revised 2021-07-27.

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

File I

\zref-check implementation

Start the DocStrip guards.

```

1 <*package>
    Identify the internal prefix (LATEX3 DocStrip convention).
2 <@@=zrefcheck>

```

1 Initial setup

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 }%

14 \ProvidesExplPackage {zref-check} {2021-07-27} {0.1.0-alpha}
15 {Flexible cross-references with contextual checks based on zref}

```

2 Dependencies

```

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

```

3 zref setup

`\g__zrefcheck_abschap_int` Provide absolute counters for section and chapter, and respective `zref` properties, so that we can make checks about relation of chapters/sections regardless of internal counters, since we don't get those for the unnumbered (starred) ones. About the proper place to make the hooks for this purpose, see <https://tex.stackexchange.com/q/605533/105447>, thanks Ulrike Fischer.

```

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

```

This is the list 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 }
33 \zref@addprops { zrefcheck }
34 {
35   abspage ,
36   abschap ,
37   abssec ,
38   page
39 }

```

4 Plumbing

4.1 Messages

```

\__zrefcheck_message:nnnn
\__zrefcheck_message:nnnx
40 \cs_new:Npn \__zrefcheck_message:nnnn #1#2#3#4
41 {
42   \use:c { msg_ \l__zrefcheck_msglevel_tl :nnnnn }
43   { zref-check } {#1} {#2} {#3} {#4}
44 }
45 \cs_generate_variant:Nn \__zrefcheck_message:nnnn { nnnx }

```

(End definition for `__zrefcheck_message:nnnn`.)

```

46 \msg_new:nnn { zref-check } { check-failed }
47 {
48   Failed-check~'#1'~for~label~'#2' \iow_newline:
49   on~page~#3~on~input~line~\msg_line_number:.
50 }
51 \msg_new:nnn { zref-check } { double-check }
52 {
53   Double-check~'#1'~for~label~'#2' \iow_newline:
54   on~page~#3~on~input~line~\msg_line_number:.
55 }

```

```

56 \msg_new:nnn { zref-check } { check-missing }
57 { Check~'#1'~not~defined~on~input~line~\msg_line_number:. }
58 \msg_new:nnn { zref-check } { property-undefined }
59 { Property~'#1'~not~defined~on~input~line~\msg_line_number:. }
60 \msg_new:nnn { zref-check } { property-not-in-label }
61 { Label~'#1'~has~no~property~'#2'~on~input~line~\msg_line_number:. }
62 \msg_new:nnn { zref-check } { property-not-integer }
63 {
64   Property~'#1'~for~label~'#2'~not~an~integer \iow_newline:
65   on~input~line~\msg_line_number:.
66 }
67 \msg_new:nnn { zref-check } { hyperref-preamble-only }
68 {
69   Option~'hyperref'~only~available~in~the~preamble. \iow_newline:
70   Use~the~starred~version~of~'\noexpand\zcheck'~instead.
71 }
72 \msg_new:nnn { zref-check } { missing-hyperref }
73 { Missing~'hyperref'~package. \iow_newline: Setting~'hyperref=false'. }
74 \msg_new:nnn { zref-check } { ignore-document-only }
75 {
76   Option~'ignore'~only~available~in~the~document. \iow_newline:
77   Use~option~'msglevel'~instead.
78 }
79 \msg_new:nnn { zref-check } { option-preamble-only }
80 {
81   Option~'#1'~only~available~in~the~preamble \iow_newline:
82   on~input~line~\msg_line_number:.
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 }

```

4.2 Options

hyperref option

```

\l__zrefcheck_use_hyperref_bool
\l__zrefcheck_warn_hyperref_bool
89 \bool_new:N \l__zrefcheck_use_hyperref_bool
90 \bool_new:N \l__zrefcheck_warn_hyperref_bool
91 \keys_define:nn { zref-check }
92 {
93   hyperref .choice: ,
94   hyperref / auto .code:n =
95   {
96     \bool_set_true:N \l__zrefcheck_use_hyperref_bool
97     \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
98   } ,
99   hyperref / true .code:n =
100   {
101     \bool_set_true:N \l__zrefcheck_use_hyperref_bool
102     \bool_set_true:N \l__zrefcheck_warn_hyperref_bool

```

```

103     } ,
104     hyperref / false .code:n =
105     {
106         \bool_set_false:N \l__zrefcheck_use_hyperref_bool
107         \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
108     } ,
109     hyperref .initial:n = auto ,
110     hyperref .default:n = auto
111 }

```

(End definition for \l__zrefcheck_use_hyperref_bool and \l__zrefcheck_warn_hyperref_bool.)

```

112 \AtBeginDocument
113 {
114     \ifpackageloaded { hyperref }
115     {
116         \bool_if:NT \l__zrefcheck_use_hyperref_bool
117         {
118             \RequirePackage { zref-hyperref }
119             \zref@addprop { zrefcheck } { anchor }
120         }
121     }
122     {
123         \bool_if:NT \l__zrefcheck_warn_hyperref_bool
124         { \msg_warning:nn { zref-check } { missing-hyperref } }
125         \bool_set_false:N \l__zrefcheck_use_hyperref_bool
126     }
127     \keys_define:nn { zref-check }
128     {
129         hyperref .code:n =
130         { \msg_warning:nn { zref-check } { hyperref-preamble-only } }
131     }
132 }

```

msglevel option

\l__zrefcheck_msglevel_tl

```

133 \tl_new:N \l__zrefcheck_msglevel_tl
134 \keys_define:nn { zref-check }
135 {
136     msglevel .choice: ,
137     msglevel / warn .code:n =
138     { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } } ,
139     msglevel / info .code:n =
140     { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } } ,
141     msglevel / none .code:n =
142     { \tl_set:Nn \l__zrefcheck_msglevel_tl { none } } ,
143     msglevel / obeydraft .code:n =
144     {
145         \ifdraft
146         { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
147         { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
148     } ,
149     msglevel / obeyfinal .code:n =
150     {

```

```

151         \ifoptionfinal
152         { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
153         { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
154     } ,
155     msglevel .value_required:n = true ,
156     msglevel .initial:n = warn ,

```

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

```

157     ignore .code:n =
158     { \msg_warning:nn { zref-check } { ignore-document-only } } ,
159     ignore .value_forbidden:n = true
160 }

```

(End definition for `\l__zrefcheck_msglevel_tl`.)

```

161 \AtBeginDocument
162 {
163     \keys_define:nn { zref-check }
164     { ignore .meta:n = { msglevel = none } }
165 }

```

`onpage` option

`\l__zrefcheck_msgonpage_bool`

```

166 \bool_new:N \l__zrefcheck_msgonpage_bool
167 \keys_define:nn { zref-check }
168 {
169     onpage .choice: ,
170     onpage / labelseq .code:n =
171     {
172         \bool_set_false:N \l__zrefcheck_msgonpage_bool
173     } ,
174     onpage / msg .code:n =
175     {
176         \bool_set_true:N \l__zrefcheck_msgonpage_bool
177     } ,
178     onpage / obeydraft .code:n =
179     {
180         \ifdraft
181         { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
182         { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
183     } ,
184     onpage / obeyfinal .code:n =
185     {
186         \ifoptionfinal
187         { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
188         { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
189     } ,
190     onpage .value_required:n = true ,
191     onpage .initial:n = labelseq
192 }

```

(End definition for `\l__zrefcheck_msgonpage_bool`.)

`closerange` option

`\l_zrefcheck_close_range_int`

```

193 \int_new:N \l__zrefcheck_close_range_int
194 \keys_define:nn { zref-check }
195 {
196   closerange .int_set:N = \l__zrefcheck_close_range_int ,
197   closerange .value_required:n = true ,
198   closerange .initial:n = 5
199 }

```

(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>.

```

200 \tl_new:N \l__zrefcheck_target_label_tl
201 \bool_new:N \l__zrefcheck_target_label_bool
202 \keys_define:nn { zref-check }
203 {
204   labelcmd .code:n =
205   {
206     \tl_set:NV \l__zrefcheck_target_label_tl \l_keys_value_tl
207     \bool_set_true:N \l__zrefcheck_target_label_bool
208   } ,
209   labelcmd .value_required:n = true ,
210 }

```

(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`.

```

211 \cs_new:Npn \__zrefcheck_target_label:n #1
212 { \zref@labelbylist {#1} { zrefcheck } }

```

(End definition for __zrefcheck_target_label:n.)

```

213 \AtBeginDocument
214 {
215   \bool_if:NT \l__zrefcheck_target_label_bool
216   {
217     \tl_if_blank:VT \l__zrefcheck_target_label_tl
218     { \tl_clear:N \l__zrefcheck_target_label_tl }
219     \cs_if_exist:cTF { \l__zrefcheck_target_label_tl }
220     {
221       \cs_set:Npx \__zrefcheck_target_label:n #1
222       {
223         \exp_not:o
224         { \cs:w \l__zrefcheck_target_label_tl \cs_end: }
225         {#1}
226       }
227     }
228     {
229       \exp_args:NnnV \msg_warning:nnn { zref-check }
230       { labelcmd-undefined } { \l__zrefcheck_target_label_tl }
231     }
232   }

```

```

233 \keys_define:nn { zref-check }
234 {
235     labelcmd .code:n =
236     {
237         \msg_warning:nnn { zref-check }
238         { option-preamble-only } { labelcmd }
239     }
240 }
241 }

```

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

```

242 \RequirePackage { l3keys2e }
243 \ProcessKeysOptions { zref-check }

```

`\zrefchecksetup` Provide `\zrefchecksetup`.

```

244 \NewDocumentCommand \zrefchecksetup { m }
245 { \keys_set:nn { zref-check } {#1} }

```

(End definition for `\zrefchecksetup`. This function is documented on page ??.)

4.3 Position on page

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

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

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

Patching this macro for this is not possible. First, `\zref@newlabel` is one of those “commands that look ahead” mentioned in `ltxcmds` documentation. Indeed, `\@newl@bel` receives 3 arguments, and `\zref@newlabel` just passes the first, the following two will be scanned ahead. Second, the `ltxcmds` hooks are not actually available when the `.aux` file is read, they come only after `\begin{document}`. Hence, redefinition would be the only alternative. My attempts at this ended up registered at <https://tex.stackexchange.com/a/604744>. But the best result in these lines was:

```

\ZREF@Robust\edef\zref@newlabel#1{
  \noexpand\seq_gput_right:Nn \noexpand\g__zrefcheck_auxfile_lblseq_seq {#1}
  \noexpand\@newl@bel{\ZREF@RefPrefix}{#1}
}

```

However, better than the above is to just read it from the `.aux` file directly, which relieves us from hacking into any internals. That’s what David Carlisle’s answer at <https://tex.stackexchange.com/a/147705> does. This answer has actually been converted into the package `listbls` by Norbert Melzer, but it is made to work with regular labels, not with `zref`’s. And it also does not really expose the information in a retrievable way (as far as I can tell). So, the below is adapted from Carlisle’s answer’s technique (a poor man’s version of it...).

There is some subtlety here as to whether this approach makes it safe for us to read the labels at this point without `\zref@wrapper@babel`. The common wisdom is that babel’s shorthands are only active after `\begin{document}` (e.g., <https://tex.stackexchange.com/a/98897>). Alas, it is more complicated than that. Babel’s documentation says (in section 9.5 Shorthands): “To prevent problems with the loading of other packages after babel we reset the catcode of the character to the original one at the end of the package and of each language file (except with `KeepShorthandsActive`). It is re-activate[d] again at `\begin{document}`. We also need to make sure that the shorthands are active during the processing of the `.aux` file. Otherwise some citations may give unexpected results in the printout when a shorthand was used in the optional argument of `\bibitem` for example.” This is done with `\if@files\write\@mainaux{...}`. In other words, the catcode change is written in the `.aux` file itself! Indeed, if you inspect the file, you’ll find them there. Besides, there is still the ominous “except with `KeepShorthandsActive`”.

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

For reference: About what are valid characters for use in labels: <https://tex.stackexchange.com/a/18312>. About some problems with active colons: <https://tex.stackexchange.com/a/89470>. About the difference between L3 strings and token lists, see <https://tex.stackexchange.com/a/446381>, in particular Joseph Wright’s comment: “Strings are for data that will never be typeset, for example file names, identifiers, etc.: if the material may be used in typesetting, it should be a token list.” See also moewe’s (CW) answer in the same lines. Which suggests using L3 strings for the reference labels might be a good catch all approach, and possibly more robust. David Carlisle’s comment about `inputenc` is a caveat (see https://tex.stackexchange.com/q/446123#comment1516961_446381). Still... let’s stick to tradition as long as it works, `zref` already does a great job here anyway.

`\g_zrefcheck_auxfile_lblseq_prop`

```
246 \prop_new:N \g_zrefcheck_auxfile_lblseq_prop
```

(End definition for `\g_zrefcheck_auxfile_lblseq_prop`.)

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

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

```
249 {
```

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.

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

```
251 \group_begin:
```

```
252 \int_zero:N \l_tmpa_int
```

```
253 \tl_clear:N \l_tmpa_tl
```

```
254 \tl_clear:N \l_tmpb_tl
```

```
255 \bool_set_false:N \l_tmpa_bool
```

```
256 \ior_map_variable:NNn \g_tmpa_ior \l_tmpa_tl
```

```
257 {
```

```
258 \tl_map_variable:NNn \l_tmpa_tl \l_tmpb_tl
```

```

259         {
260             \tl_if_eq:NnTF \l_tmpb_tl { \zref@newlabel }
261             {
Found a \zref@label, signal it.
262                 \bool_set_true:N \l_tmpa_bool
263             }
264             {
265                 \bool_if:NTF \l_tmpa_bool
266                 {
267                     \bool_set_false:N \l_tmpa_bool
268                     \int_incr:N \l_tmpa_int
269                     \prop_gput:Nxx \g__zrefcheck_auxfile_lblseq_prop
270                     { \l_tmpb_tl } { \int_use:N \l_tmpa_int }
271                 }
272             }

```

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.

```

273             \tl_map_break:
274         }
275     }
276 }
277 }
278 \group_end:
279 \ior_close:N \g_tmpa_ior
280 }

```

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. 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.

4.4 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`. I don’t know why it is needed, but if Oberdiek does it, there must be a reason. In any case, the requirements are the same, we need numbers ensured to be *unique* in the counter.

```

281 \begingroup
282   \let \@addtoreset \ltx@gobbletwo
283   \newcounter { zrefcheck }
284 \endgroup
285 \setcounter { zrefcheck } { 0 }

```

4.5 Label formats

```

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

286 \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>}

287 \cs_new:Npn \__zrefcheck_end_lblfmt:n #1 { #1 @zrefcheck }

(End definition for \__zrefcheck_end_lblfmt:n.)

```

4.6 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. 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>}

288 \cs_new:Npn \zrefcheck_get_astl:nnn #1#2#3
289 {
290   \tl_clear:N #3
291   \tl_if_eq:nnTF {#2} { lblseq }
292   {
293     \prop_get:NnNF \g__zrefcheck_auxfile_lblseq_prop {#1} #3
294     {
295       \msg_warning:nnnn { zref-check }
296       { property-not-in-label } {#1} {#2}
297     }
298   }
299   {

```

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.

```

300     \zref@ifrefundefined {#1}
301     {}
302     {
303         \zref@ifpropundefined {#2}
304         { \msg_warning:nnnn { zref-check } { property-undefined } {#2} }
305         {
306             \zref@ifrefcontainsprop {#1} {#2}
307             {
308                 \tl_set:Nx #3
309                 { \zref@extractdefault {#1} {#2} { \c_empty_tl } }
310             }
311             {
312                 \msg_warning:nnnn
313                 { zref-check } { property-not-in-label } {#1} {#2}
314             }
315         }
316     }
317 }
318 }
```

(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).

```

319 \bool_new:N \l__zrefcheck_integer_bool
```

(End definition for `\l__zrefcheck_integer_bool`.)

`\l__zrefcheck_propval_tl`

```

320 \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>}
321 \cs_new:Npn \zrefcheck_get_asint:nnn #1#2#3
322 {
323     \zrefcheck_get_astl:nnn {#1} {#2} { \l__zrefcheck_propval_tl }
```

```

324 \__zrefcheck_is_integer:nTF { \l__zrefcheck_propval_tl }
325 {

```

Make it an integer data type.

```

326 \int_set:Nn #3 { \int_eval:n { \l__zrefcheck_propval_tl } }
327 }
328 {
329 \bool_set_false:N \l__zrefcheck_integer_bool
330 \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`.

```

331 { }
332 {
333 \msg_warning:nnnn { zref-check }
334 { property-not-integer } {#2} {#1}
335 }
336 }
337 }

```

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

```

\__zrefcheck_is_integer:n
\__zrefcheck_int_to_roman:w

```

Thanks egreg: <https://tex.stackexchange.com/a/244405>, 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 I'm 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 a zero or negative number as argument, so this must also be controlled for since, in our use case, this may happen.

```

338 \cs_new_eq:NN \__zrefcheck_int_to_roman:w \__int_to_roman:w
339 \prg_new_conditional:Npnn \__zrefcheck_is_integer:n #1 { p , T , F , TF }
340 {
341 \tl_if_empty:oTF {#1}
342 { \prg_return_false: }
343 {
344 \tl_if_empty:oTF { \__zrefcheck_int_to_roman:w -0#1 }
345 { \prg_return_true: }
346 { \prg_return_false: }
347 }
348 }

```

(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 `\tex_romannumeral:D` 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.

```

349 \prg_new_protected_conditional:Npnn \__zrefcheck_is_integer_rgx:n #1 { TF }

```

```

350 {
351   \regex_match:nnTF { \A\d+\Z } {#1}
352   { \prg_return_true: }
353   { \prg_return_false: }
354 }
355 \prg_generate_conditional_variant:Nnn \__zrefcheck_is_integer_rgx:n { x } { TF }
(End definition for \__zrefcheck_is_integer_rgx:n.)

```

5 User interface

5.1 \zcheck

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

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

```

356 \NewDocumentCommand \zcheck
357 { s O { } } > { \SplitList { , } } m > { \SplitList { , } } 0 { } m }
358 { \zref@wrapper@babel \__zrefcheck_zcheck:nnnnn {#3} {#1} {#2} {#4} {#5} }

```

(End definition for `\zcheck`. This function is documented on page ??.)

```

\g__zrefcheck_id_int
\l__zrefcheck_checkbeg_tl
\l__zrefcheck_checkend_tl
\l__zrefcheck_link_label_tl
\l__zrefcheck_link_anchor_tl
\l__zrefcheck_link_star_tl
359 \int_new:N \g__zrefcheck_id_int
360 \tl_new:N \l__zrefcheck_checkbeg_tl
361 \tl_new:N \l__zrefcheck_checkend_tl
362 \tl_new:N \l__zrefcheck_link_label_tl
363 \tl_new:N \l__zrefcheck_link_anchor_tl
364 \bool_new:N \l__zrefcheck_link_star_tl

```

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

`__zrefcheck_zcheck:nnnnn` 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`. This is more or less what the definition of `\zref` in `zref-user.sty` does for this.

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

```

365 \cs_new:Npn \__zrefcheck_zcheck:nnnnn #1#2#3#4#5
366 {
367   \group_begin:

```

Process local options.

```

368   \keys_set:nn { zref-check } {#3}

```

Names of the labels for this zrefcheck call.

```

369   \int_gincr:N \g__zrefcheck_id_int
370   \tl_set:Nx \l__zrefcheck_checkbeg_tl
371   { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
372   \tl_set:Nx \l__zrefcheck_checkend_tl
373   { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }

```

Set checkbeg label.

```
374 \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrefcheck }
```

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.

```
375 \tl_set:Nn \l__zrefcheck_link_label_tl { \tl_head:n {#1} }
376 \bool_set:Nn \l__zrefcheck_link_star_tl {#2}
377 \zref@ifrefundefined { \l__zrefcheck_link_label_tl }
```

If the reference is undefined, just typeset.

```
378 {#5}
379 {
380   \bool_if:nTF
381   {
382     \l__zrefcheck_use_hyperref_bool &&
383     ! \l__zrefcheck_link_star_tl
384   }
385   {
386     \exp_args:Nx \zrefcheck_get_astl:nnn
387     { \l__zrefcheck_link_label_tl }
388     { anchor } { \l__zrefcheck_link_anchor_tl }
389     \hyperlink { \l__zrefcheck_link_anchor_tl } {#5}
390   }
391   {#5}
392 }
```

Set checkend label.

```
393 \zref@labelbylist { \l__zrefcheck_checkend_tl } { zrefcheck }
```

Check definition. Note that, even if not indicated in zref’s documentation by the usual ‘babel’ markup, `\zref@refused` is protected by `\zref@wrapper@babel`.

```
394 \tl_map_function:nN {#1} \zref@refused
```

Run the checks.

```
395 \__zrefcheck_run_checks:nnV {#4} {#1} { \l__zrefcheck_checkbeg_tl }
396 \group_end:
397 }
```

(End definition for `__zrefcheck_zcheck:nnnnn`.)

5.2 Targets

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

```
398 \NewDocumentCommand \zctarget { m +m }
399 {
```

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.

```
400   \group_begin:
401   \refstepcounter { zrefcheck }
402   \zref@wrapper@babel \__zrefcheck_target_label:n {#1}
403   #2
404   \zref@wrapper@babel
```

```

405     \zref@labelbylist { \__zrefcheck_end_lblfmt:n {#1} } { zrefcheck }
406   \group_end:
407 }

```

(End definition for `\zctarget`. This function is documented on page ??.)

```

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

zcregion
408 \NewDocumentEnvironment {zcregion} { m }
409 {
410   \refstepcounter { zrefcheck }
411   \zref@wrapper@babel \__zrefcheck_target_label:n {#1}
412 }
413 {
414   \zref@wrapper@babel
415   \zref@labelbylist { \__zrefcheck_end_lblfmt:n {#1} } { zrefcheck }
416 }

```

(End definition for `zcregion`. This function is documented on page ??.)

6 Checks

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

First, a conditional function defined with:

```
\prg_new_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.

6.1 Running

```

\__zrefcheck_run_checks:nnn
\__zrefcheck_run_checks:nnV
417 \cs_new:Npn \__zrefcheck_run_checks:nnn #1#2#3
418 {
419   \group_begin:
420     \tl_map_inline:nn {#2}

```



```

421     {
422       \tl_map_inline:nn {#1}
423       { \__zrefcheck_do_check:nnn {####1} {##1} {#3} }
424     }
425   \group_end:
426 }
427 \cs_generate_variant:Nn \__zrefcheck_run_checks:nnn { nnV }

```

(End definition for __zrefcheck_run_checks:nnn.)

```

\l__zrefcheck_passedcheck_bool
\l__zrefcheck_onpage_bool
\c__zrefcheck_onpage_checks_seq
428 \bool_new:N \l__zrefcheck_passedcheck_bool
429 \bool_new:N \l__zrefcheck_onpage_bool
430 \seq_new:N \c__zrefcheck_onpage_checks_seq
431 \seq_set_from_clist:Nn \c__zrefcheck_onpage_checks_seq
432 { 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.

```

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

```

```

\__zrefcheck_do_check:nnn \__zrefcheck_do_check:nnn {<check>} {<label beg>} {<reference beg>}

```

```

434 \cs_new:Npn \__zrefcheck_do_check:nnn #1#2#3
435 {
436   \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.

```

437   \zref@ifrefundefined {#2}
438   {}
439   {
440     \tl_if_empty:nTF {#1}
441     {}
442     {
443       \bool_set_true:N \l__zrefcheck_passedcheck_bool
444       \bool_set_false:N \l__zrefcheck_onpage_bool
445       \cs_if_exist:cTF { __zrefcheck_check_ #1 :nnF }
446       {

```

“label beg” vs “reference beg”.

```

447       \use:c { __zrefcheck_check_ #1 :nnF }
448       {#2} {#3}
449       { \bool_set_false:N \l__zrefcheck_passedcheck_bool }

```

“label beg” vs “reference end”.

```

450       \exp_args:Nnno \use:c { __zrefcheck_check_ #1 :nnF }
451       {#2} { \__zrefcheck_end_lblfmt:n {#3} }
452       { \bool_set_false:N \l__zrefcheck_passedcheck_bool }

```

“label end” may have been created by the target commands.

```

453       \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
454       {}
455       {

```

“label end” vs “reference beg”.

```

456         \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
457         { \__zrefcheck_end_lblfmt:n {#2} } {#3}
458         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }

```

“label end” vs “reference end”.

```

459         \exp_args:Nnoo \use:c { __zrefcheck_check_ #1 :nnF }
460         { \__zrefcheck_end_lblfmt:n {#2} }
461         { \__zrefcheck_end_lblfmt:n {#3} }
462         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
463     }

```

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.

```

464         \seq_if_in:NnT \c__zrefcheck_onpage_checks_seq {#1}
465         {
466             \__zrefcheck_check_thispage:nnT
467             {#2} {#3}
468             { \bool_set_true:N \l__zrefcheck_onpage_bool }
469             \__zrefcheck_check_thispage:nnT
470             {#2} { \__zrefcheck_end_lblfmt:n {#3} }
471             { \bool_set_true:N \l__zrefcheck_onpage_bool }
472             \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
473             {}
474             {
475                 \__zrefcheck_check_thispage:nnT
476                 { \__zrefcheck_end_lblfmt:n {#2} } {#3}
477                 { \bool_set_true:N \l__zrefcheck_onpage_bool }
478                 \__zrefcheck_check_thispage:nnT
479                 { \__zrefcheck_end_lblfmt:n {#2} }
480                 { \__zrefcheck_end_lblfmt:n {#3} }
481                 { \bool_set_true:N \l__zrefcheck_onpage_bool }
482             }
483         }
484         \bool_if:NTF \l__zrefcheck_passedcheck_bool
485         {
486             \bool_if:nT
487             {
488                 \l__zrefcheck_msgonpage_bool &&
489                 \l__zrefcheck_onpage_bool
490             }
491             {
492                 \__zrefcheck_message:nnnx { double-check } {#1} {#2}
493                 { \zref@extractdefault {#3} {page} {'unknown'} }
494             }
495         }
496         {
497             \__zrefcheck_message:nnnx { check-failed } {#1} {#2}
498             { \zref@extractdefault {#3} {page} {'unknown'} }
499         }
500     }
501     { \msg_warning:nnn { zref-check } { check-missing } {#1} }
502 }

```

```

503     }
504     \group_end:
505 }

```

(End definition for `__zrefcheck_do_check:nnn`.)

6.2 Conditionals

```

\l__zrefcheck_lbl_int More readable scratch variables for the tests.
\l__zrefcheck_ref_int
\l__zrefcheck_lbl_b_int
\l__zrefcheck_ref_b_int

```

```

506 \int_new:N \l__zrefcheck_lbl_int
507 \int_new:N \l__zrefcheck_ref_int
508 \int_new:N \l__zrefcheck_lbl_b_int
509 \int_new:N \l__zrefcheck_ref_b_int

```

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

6.2.1 This page

```

\__zrefcheck_check_thispage:nn

```

```

510 \prg_new_conditional:Npnn \__zrefcheck_check_thispage:nn #1#2 { T , F , TF }
511 {
512     \group_begin:
513     \bool_set_true:N \l__zrefcheck_integer_bool
514     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
515     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
516     \bool_lazy_and:nnTF
517     { \l__zrefcheck_integer_bool }
518     {
519         \int_compare_p:nNn
520         { \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.

```

521         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
522         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
523     }
524     { \group_insert_after:N \prg_return_true: }
525     { \group_insert_after:N \prg_return_false: }
526     \group_end:
527 }

```

(End definition for `__zrefcheck_check_thispage:nn`.)

6.2.2 On page

```

\__zrefcheck_check_above:nn
\__zrefcheck_check_below:nn

```

```

528 \prg_new_conditional:Npnn \__zrefcheck_check_above:nn #1#2 { F , TF }
529 {
530     \group_begin:
531     \__zrefcheck_check_thispage:nnTF {#1} {#2}
532     {
533         \bool_set_true:N \l__zrefcheck_integer_bool

```

```

534 \zrefcheck_get_asint:nnn {#1} { lblseq } { \l__zrefcheck_lbl_int }
535 \zrefcheck_get_asint:nnn {#2} { lblseq } { \l__zrefcheck_ref_int }
536 \bool_lazy_and:nnTF
537 { \l__zrefcheck_integer_bool }
538 {
539   \int_compare_p:nNn
540     { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
541     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
542     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
543   }
544   { \group_insert_after:N \prg_return_true: }
545   { \group_insert_after:N \prg_return_false: }
546 }
547 { \group_insert_after:N \prg_return_false: }
548 \group_end:
549 }
550 \prg_new_conditional:Npnn \__zrefcheck_check_below:nn #1#2 { F , TF }
551 {
552   \__zrefcheck_check_thispage:nnTF {#1} {#2}
553   {
554     \__zrefcheck_check_above:nnTF {#1} {#2}
555     { \prg_return_false: }
556     { \prg_return_true: }
557   }
558   { \prg_return_false: }
559 }

```

(End definition for __zrefcheck_check_above:nn and __zrefcheck_check_below:nn.)

6.2.3 Before / After

```

\__zrefcheck_check_before:nn
\__zrefcheck_check_after:nn
560 \prg_new_conditional:Npnn \__zrefcheck_check_before:nn #1#2 { F }
561 {
562   \__zrefcheck_check_pagesbefore:nnTF {#1} {#2}
563   { \prg_return_true: }
564   {
565     \__zrefcheck_check_above:nnTF {#1} {#2}
566     { \prg_return_true: }
567     { \prg_return_false: }
568   }
569 }
570 \prg_new_conditional:Npnn \__zrefcheck_check_after:nn #1#2 { F }
571 {
572   \__zrefcheck_check_pagesafter:nnTF {#1} {#2}
573   { \prg_return_true: }
574   {
575     \__zrefcheck_check_below:nnTF {#1} {#2}
576     { \prg_return_true: }
577     { \prg_return_false: }
578   }
579 }

```

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

6.2.4 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_facing:nn

580 \prg_new_conditional:Npnn \__zrefcheck_check_nextpage:nn #1#2 { F }
581 {
582   \group_begin:
583     \bool_set_true:N \l__zrefcheck_integer_bool
584     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
585     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
586     \bool_lazy_and:nnTF
587       { \l__zrefcheck_integer_bool }
588       {
589         \int_compare_p:nNn
590           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
591           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
592           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
593       }
594     { \group_insert_after:N \prg_return_true: }
595     { \group_insert_after:N \prg_return_false: }
596   \group_end:
597 }
598 \prg_new_conditional:Npnn \__zrefcheck_check_prevpage:nn #1#2 { F }
599 {
600   \group_begin:
601     \bool_set_true:N \l__zrefcheck_integer_bool
602     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
603     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
604     \bool_lazy_and:nnTF
605       { \l__zrefcheck_integer_bool }
606       {
607         \int_compare_p:nNn
608           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
609           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
610           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
611       }
612     { \group_insert_after:N \prg_return_true: }
613     { \group_insert_after:N \prg_return_false: }
614   \group_end:
615 }
616 \prg_new_conditional:Npnn \__zrefcheck_check_pagesbefore:nn #1#2 { F , TF }
617 {
618   \group_begin:
619     \bool_set_true:N \l__zrefcheck_integer_bool
620     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
621     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
622     \bool_lazy_and:nnTF
623       { \l__zrefcheck_integer_bool }
624       {
625         \int_compare_p:nNn
626           { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
627           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
628           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
629       }
630     { \group_insert_after:N \prg_return_true: }

```

```

631     { \group_insert_after:N \prg_return_false: }
632   \group_end:
633 }
634 \cs_new_eq:NN \__zrefcheck_check_ppbefore:nnF \__zrefcheck_check_pagesbefore:nnF
635 \prg_new_conditional:Npnn \__zrefcheck_check_pagesafter:nn #1#2 { F , TF }
636 {
637   \group_begin:
638     \bool_set_true:N \l__zrefcheck_integer_bool
639     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
640     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
641     \bool_lazy_and:nnTF
642       { \l__zrefcheck_integer_bool }
643       {
644         \int_compare_p:nNn
645           { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
646           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
647           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
648       }
649     { \group_insert_after:N \prg_return_true: }
650     { \group_insert_after:N \prg_return_false: }
651   \group_end:
652 }
653 \cs_new_eq:NN \__zrefcheck_check_ppafter:nnF \__zrefcheck_check_pagesafter:nnF
654 \prg_new_conditional:Npnn \__zrefcheck_check_facing:nn #1#2 { F }
655 {
656   \group_begin:
657     \bool_set_true:N \l__zrefcheck_integer_bool
658     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
659     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
660     \bool_lazy_and:nnTF
661       { \l__zrefcheck_integer_bool }
662       {

```

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

```

663     \legacy_if_p:n { @twoside } &&

```

Now we test “facing”.

```

664     (
665       (
666         \int_if_odd_p:n { \l__zrefcheck_ref_int } &&
667         \int_compare_p:nNn
668           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 }
669       ) ||
670       (
671         \int_if_even_p:n { \l__zrefcheck_ref_int } &&
672         \int_compare_p:nNn
673           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 }
674       )
675     ) &&
676     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
677     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
678   }
679   { \group_insert_after:N \prg_return_true: }
680   { \group_insert_after:N \prg_return_false: }
681 \group_end:

```

682 }

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

6.2.5 Close / Far

```

\_zrefcheck\_check\_close:nn
\_zrefcheck\_check\_far:nn
683 \prg_new_conditional:Npnn \_zrefcheck\_check\_close:nn #1#2 { F , TF }
684 {
685   \group_begin:
686     \bool_set_true:N \l\_zrefcheck\_integer\_bool
687     \zrefcheck\_get\_asint:nnn {#1} { abspage } { \l\_zrefcheck\_lbl\_int }
688     \zrefcheck\_get\_asint:nnn {#2} { abspage } { \l\_zrefcheck\_ref\_int }
689     \bool_lazy_and:nnTF
690       { \l\_zrefcheck\_integer\_bool }
691       {
692         \int_compare_p:nNn
693           { \int_abs:n { \l\_zrefcheck\_lbl\_int - \l\_zrefcheck\_ref\_int } }
694           <
695           { \l\_zrefcheck\_close\_range\_int + 1 } &&
696           ! \int_compare_p:nNn { \l\_zrefcheck\_lbl\_int } = { 0 } &&
697           ! \int_compare_p:nNn { \l\_zrefcheck\_ref\_int } = { 0 }
698       }
699       { \group_insert_after:N \prg_return_true: }
700       { \group_insert_after:N \prg_return_false: }
701   \group_end:
702 }
703 \prg_new_conditional:Npnn \_zrefcheck\_check\_far:nn #1#2 { F }
704 {
705   \_zrefcheck\_check\_close:nnTF {#1} {#2}
706   { \prg_return_false: }
707   { \prg_return_true: }
708 }

```

(End definition for `_zrefcheck_check_close:nn` and `_zrefcheck_check_far:nn`.)

6.2.6 Chapter

```

\_zrefcheck\_check\_thischap:nn
\_zrefcheck\_check\_nextchap:nn
\_zrefcheck\_check\_prevchap:nn
\_zrefcheck\_check\_chapsafter:nn
\_zrefcheck\_check\_chapsbefore:nn
709 \prg_new_conditional:Npnn \_zrefcheck\_check\_thischap:nn #1#2 { F }
710 {
711   \group_begin:
712     \bool_set_true:N \l\_zrefcheck\_integer\_bool
713     \zrefcheck\_get\_asint:nnn {#1} { abschap } { \l\_zrefcheck\_lbl\_int }
714     \zrefcheck\_get\_asint:nnn {#2} { abschap } { \l\_zrefcheck\_ref\_int }
715     \bool_lazy_and:nnTF
716       { \l\_zrefcheck\_integer\_bool }
717       {
718         \int_compare_p:nNn
719           { \l\_zrefcheck\_lbl\_int } = { \l\_zrefcheck\_ref\_int } &&

```

‘0’ is the default value of `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.

```

720         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
721         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
722     }
723     { \group_insert_after:N \prg_return_true: }
724     { \group_insert_after:N \prg_return_false: }
725 \group_end:
726 }
727 \prg_new_conditional:Npnn \__zrefcheck_check_nextchap:nn #1#2 { F }
728 {
729     \group_begin:
730     \bool_set_true:N \l__zrefcheck_integer_bool
731     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
732     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
733     \bool_lazy_and:nnTF
734     { \l__zrefcheck_integer_bool }
735     {
736         \int_compare_p:nNn
737         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
738         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
739     }
740     { \group_insert_after:N \prg_return_true: }
741     { \group_insert_after:N \prg_return_false: }
742 \group_end:
743 }
744 \prg_new_conditional:Npnn \__zrefcheck_check_prevchap:nn #1#2 { F }
745 {
746     \group_begin:
747     \bool_set_true:N \l__zrefcheck_integer_bool
748     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
749     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
750     \bool_lazy_and:nnTF
751     { \l__zrefcheck_integer_bool }
752     {
753         \int_compare_p:nNn
754         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
755         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
756         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
757     }
758     { \group_insert_after:N \prg_return_true: }
759     { \group_insert_after:N \prg_return_false: }
760 \group_end:
761 }
762 \prg_new_conditional:Npnn \__zrefcheck_check_chapsafter:nn #1#2 { F }
763 {
764     \group_begin:
765     \bool_set_true:N \l__zrefcheck_integer_bool
766     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
767     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
768     \bool_lazy_and:nnTF
769     { \l__zrefcheck_integer_bool }
770     {
771         \int_compare_p:nNn

```



```

772         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
773         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
774     }
775     { \group_insert_after:N \prg_return_true: }
776     { \group_insert_after:N \prg_return_false: }
777 \group_end:
778 }
779 \prg_new_conditional:Npnn \__zrefcheck_check_chapsbefore:nn #1#2 { F }
780 {
781     \group_begin:
782     \bool_set_true:N \l__zrefcheck_integer_bool
783     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
784     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
785     \bool_lazy_and:nnTF
786     { \l__zrefcheck_integer_bool }
787     {
788         \int_compare_p:nNn
789         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
790         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
791         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
792     }
793     { \group_insert_after:N \prg_return_true: }
794     { \group_insert_after:N \prg_return_false: }
795 \group_end:
796 }

```

(End definition for __zrefcheck_check_thischap:nn and others.)

6.2.7 Section

```

\__zrefcheck_check_thissec:nn
\__zrefcheck_check_nextsec:nn
\__zrefcheck_check_prevsec:nn
\__zrefcheck_check_secsoafter:nn
\__zrefcheck_check_secsobefore:nn
797 \prg_new_conditional:Npnn \__zrefcheck_check_thissec:nn #1#2 { F }
798 {
799     \group_begin:
800     \bool_set_true:N \l__zrefcheck_integer_bool
801     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
802     \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
803     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
804     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
805     \bool_lazy_and:nnTF
806     { \l__zrefcheck_integer_bool }
807     {
808         \int_compare_p:nNn
809         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
810         \int_compare_p:nNn
811         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int } &&

```

‘0’ is the default value of `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.

```

812         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
813         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }

```

```

814     }
815     { \group_insert_after:N \prg_return_true: }
816     { \group_insert_after:N \prg_return_false: }
817 \group_end:
818 }
819 \prg_new_conditional:Npnn \__zrefcheck_check_nextsec:nn #1#2 { F }
820 {
821   \group_begin:
822     \bool_set_true:N \l__zrefcheck_integer_bool
823     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
824     \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
825     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
826     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
827     \bool_lazy_and:nnTF
828       { \l__zrefcheck_integer_bool }
829       {
830         \int_compare_p:nNn
831           { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
832         \int_compare_p:nNn
833           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
834         ! \int_compare_p:nNn { \l__zrefcheck_lbl_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_conditional:Npnn \__zrefcheck_check_prevsec:nn #1#2 { F }
841 {
842   \group_begin:
843     \bool_set_true:N \l__zrefcheck_integer_bool
844     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
845     \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
846     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
847     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
848     \bool_lazy_and:nnTF
849       { \l__zrefcheck_integer_bool }
850       {
851         \int_compare_p:nNn
852           { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
853         \int_compare_p:nNn
854           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
855         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
856         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
857       }
858     { \group_insert_after:N \prg_return_true: }
859     { \group_insert_after:N \prg_return_false: }
860   \group_end:
861 }
862 \prg_new_conditional:Npnn \__zrefcheck_check_secsafter:nn #1#2 { F }
863 {
864   \group_begin:
865     \bool_set_true:N \l__zrefcheck_integer_bool
866     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
867     \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }

```

```

868 \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
869 \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
870 \bool_lazy_and:nnTF
871 { \l__zrefcheck_integer_bool }
872 {
873   \int_compare_p:nNn
874   { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
875   \int_compare_p:nNn
876   { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
877   ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
878 }
879 { \group_insert_after:N \prg_return_true: }
880 { \group_insert_after:N \prg_return_false: }
881 \group_end:
882 }
883 \prg_new_conditional:Npnn \__zrefcheck_check_secsbefore:nn #1#2 { F }
884 {
885   \group_begin:
886   \bool_set_true:N \l__zrefcheck_integer_bool
887   \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
888   \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
889   \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
890   \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
891   \bool_lazy_and:nnTF
892   { \l__zrefcheck_integer_bool }
893   {
894     \int_compare_p:nNn
895     { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
896     \int_compare_p:nNn
897     { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
898     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
899     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
900   }
901   { \group_insert_after:N \prg_return_true: }
902   { \group_insert_after:N \prg_return_false: }
903   \group_end:
904 }

```

(End definition for __zrefcheck_check_thissec:nn and others.)

```

905 \endpackage

```

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.

A		B
\A	351	\begingroup 281
\AddToHook	21, 28	bool commands:
\AtBeginDocument	112, 161, 213	\bool_if:NTF 116, 123, 215, 265, 484
		\bool_if:nTF 380, 486

<code>\bool_lazy_and:nnTF</code>	12, 516, 536, 586, 604, 622, 641, 660, 689, 715, 733, 750, 768, 785, 805, 827, 848, 870, 891
<code>\bool_new:N</code>	89, 90, 166, 201, 319, 364, 428, 429
<code>\bool_set:Nn</code>	376
<code>\bool_set_false:N</code>	97, 106, 107, 125, 172, 181, 188, 255, 267, 329, 444, 449, 452, 458, 462
<code>\bool_set_true:N</code>	96, 101, 102, 176, 182, 187, 207, 262, 443, 468, 471, 477, 481, 513, 533, 583, 601, 619, 638, 657, 686, 712, 730, 747, 765, 782, 800, 822, 843, 865, 886
<code>\l_tmpa_bool</code> ...	10, 255, 262, 265, 267
C	
<code>\catcode</code>	10
<code>\chapter</code>	2, 23
cs commands:	
<code>\cs:w</code>	224
<code>\cs_end:</code>	224
<code>\cs_generate_variant:Nn</code> .	45, 427, 433
<code>\cs_if_exist:NTF</code>	219, 445
<code>\cs_new:Npn</code>	40, 211, 286, 287, 288, 321, 365, 417, 434
<code>\cs_new_eq:NN</code>	338, 634, 653
<code>\cs_set:Npx</code>	221
D	
<code>\d</code>	351
E	
<code>\endgroup</code>	284
<code>\endinput</code>	12
exp commands:	
<code>\exp_args:Nnno</code>	433, 450
<code>\exp_args:NnnV</code>	229
<code>\exp_args:Nno</code>	456
<code>\exp_args:Nnoo</code>	459
<code>\exp_args:Nx</code>	386
<code>\exp_not:n</code>	223
F	
file commands:	
<code>\file_if_exist:nTF</code>	248
<code>\fmtversion</code>	3
G	
group commands:	
<code>\group_begin:</code>	251, 367, 400, 419, 436, 512, 530, 582, 600, 618, 637, 656, 685, 711, 729, 746, 764, 781, 799, 821, 842, 864, 885
<code>\group_end:</code>	278, 396, 406, 425, 504, 526, 548, 596, 614, 632, 651, 681, 701, 725, 742, 760, 777, 795, 817, 838, 860, 881, 903
<code>\group_insert_after:N</code>	524, 525, 544, 545, 547, 594, 595, 612, 613, 630, 631, 649, 650, 679, 680, 699, 700, 723, 724, 740, 741, 758, 759, 775, 776, 793, 794, 815, 816, 836, 837, 858, 859, 879, 880, 901, 902
H	
<code>\hyperlink</code>	14, 389
I	
<code>\ifdraft</code>	145, 180
<code>\IfFormatAtLeastTF</code>	3, 4
<code>\ifoptionfinal</code>	151, 186
int commands:	
<code>\int_abs:n</code>	693
<code>\int_compare_p:nNn</code>	519, 521, 522, 539, 541, 542, 589, 591, 592, 607, 609, 610, 625, 627, 628, 644, 646, 647, 667, 672, 676, 677, 692, 696, 697, 718, 720, 721, 736, 738, 753, 755, 756, 771, 773, 788, 790, 791, 808, 810, 812, 813, 830, 832, 834, 851, 853, 855, 856, 873, 875, 877, 894, 896, 898, 899
<code>\int_eval:n</code>	326
<code>\int_gincr:N</code>	23, 29, 369
<code>\int_if_even_p:n</code>	671
<code>\int_if_odd_p:n</code>	666
<code>\int_incr:N</code>	268
<code>\int_new:N</code>	19, 20, 193, 359, 506, 507, 508, 509
<code>\int_set:Nn</code>	326
<code>\int_use:N</code>	26, 30, 270, 286
<code>\int_zero:N</code>	24, 252
<code>\l_tmpa_int</code>	252, 268, 270
int internal commands:	
<code>__int_to_roman:w</code>	13, 338
ior commands:	
<code>\ior_close:N</code>	279
<code>\ior_map_variable:NNn</code>	256
<code>\ior_open:Nn</code>	250
<code>\g_tmpa_ior</code>	250, 256, 279
iow commands:	
<code>\iow_newline:</code>	48, 53, 64, 69, 73, 76, 81
K	
keys commands:	
<code>\keys_define:nn</code>	91, 127, 134, 163, 167, 194, 202, 233

\keys_set:nn	245, 368		
\l_keys_value_tl	206		
L			
legacy commands:			
\legacy_if_p:n	663		
\let	282		
M			
\MessageBreak	10		
msg commands:			
\msg_line_number:	49, 54, 57, 59, 61, 65, 82		
\msg_new:nnn	46, 51, 56, 58, 60, 62, 67, 72, 74, 79, 84		
\msg_warning:nn	124, 130, 158		
\msg_warning:nnn	229, 237, 501		
\msg_warning:nnnn	295, 304, 312, 333		
N			
\newcounter	283		
\NewDocumentCommand	244, 356, 398		
\NewDocumentEnvironment	408		
\noexpand	70		
P			
\PackageError	7		
prg commands:			
\prg_generate_conditional_variant:Nnn	355		
\prg_new_conditional:Nppn	16, 339, 510, 528, 550, 560, 570, 580, 598, 616, 635, 654, 683, 703, 709, 727, 744, 762, 779, 797, 819, 840, 862, 883		
\prg_new_protected_conditional:Nppn	349		
\prg_return_false:	16, 342, 346, 353, 525, 545, 547, 555, 558, 567, 577, 595, 613, 631, 650, 680, 700, 706, 724, 741, 759, 776, 794, 816, 837, 859, 880, 902		
\prg_return_true:	16, 345, 352, 524, 544, 556, 563, 566, 573, 576, 594, 612, 630, 649, 679, 699, 707, 723, 740, 758, 775, 793, 815, 836, 858, 879, 901		
\ProcessKeysOptions	243		
prop commands:			
\prop_get:NnNTF	293		
\prop_gput:Nnn	269		
\prop_new:N	246		
\providecommand	3		
\ProvidesExplPackage	14		
R			
\refstepcounter	10, 15, 401, 410		
regex commands:			
\regex_match:nnTF	351		
\RequirePackage	16, 17, 18, 118, 242		
\romannumeral	13		
S			
\section	25		
seq commands:			
\seq_if_in:NnTF	464		
\seq_new:N	430		
\seq_set_from_clist:Nn	431		
\setcounter	285		
\SplitList	357		
sys commands:			
\c_sys_jobname_str	247		
T			
TeX and L ^A T _E X 2 _ε commands:			
\@addtoreset	282		
\@currentlabel	15		
\@ifl@t@r	3		
\@ifpackageloaded	114		
\@newl@bel	8		
\ltx@gobbles	282		
\zref@addprop	16, 27, 31, 119		
\zref@addprops	33		
\zref@extractdefault	11, 309, 493, 498		
\zref@ifpropundefined	303		
\zref@ifrefcontainsprop	306		
\zref@ifrefundefined	11, 300, 330, 377, 437, 453, 472		
\ZREF@label	10		
\zref@label	9, 10		
\zref@labelbylist	212, 374, 393, 405, 415		
\ZREF@mainlist	27, 31		
\zref@newlabel	8–10, 260		
\zref@newlist	32		
\zref@newprop	16, 26, 30		
\zref@refused	11, 15, 394		
\zref@require@unique	10		
\zref@wrapper@babel	8, 14, 15, 358, 402, 404, 411, 414		
tex commands:			
\tex_romannumeral:D	13		
tl commands:			
\c_empty_tl	11, 309		
\tl_clear:N	11, 218, 253, 254, 290		
\tl_head:n	375		
\tl_if_blank:nTF	13, 217		
\tl_if_empty:nTF	13, 341, 344, 440		
\tl_if_eq:NnTF	260		

<code>\tl_if_eq:nnTF</code>	291	<code>__zrefcheck_check_far:nn</code>	683
<code>\tl_map_break:</code>	273	<code>__zrefcheck_check_lblfmt:n</code>	
<code>\tl_map_function:nN</code>	394	10, 286, 371
<code>\tl_map_inline:nn</code>	420, 422	<code>__zrefcheck_check_nextchap:nn</code> .	709
<code>\tl_map_variable:NNn</code>	258	<code>__zrefcheck_check_nextpage:nn</code> .	580
<code>\tl_new:N</code>		<code>__zrefcheck_check_nextsec:nn</code> ..	797
.....	133, 200, 320, 360, 361, 362, 363	<code>__zrefcheck_check_pagesafter:nn</code>	580
<code>\tl_set:Nn</code> .	138, 140, 142, 146, 147,	<code>__zrefcheck_check_pagesafter:nnTF</code>	
	152, 153, 206, 247, 308, 370, 372, 375	572, 653
<code>\g_tmpa_tl</code>	247, 248, 250	<code>__zrefcheck_check_pagesbefore:nn</code>	
<code>\l_tmpa_tl</code>	253, 256, 258	580
<code>\l_tmpb_tl</code>	254, 258, 260, 270	<code>__zrefcheck_check_pagesbefore:nnTF</code>	
		562, 634
		<code>__zrefcheck_check_ppafter:nn</code> ..	580
		<code>__zrefcheck_check_ppafter:nnTF</code>	653
		<code>__zrefcheck_check_ppbefore:nn</code> .	580
		<code>__zrefcheck_check_ppbefore:nnTF</code>	634
		<code>__zrefcheck_check_prevchap:nn</code> .	709
		<code>__zrefcheck_check_prevpage:nn</code> .	580
		<code>__zrefcheck_check_prevsec:nn</code> ..	797
		<code>__zrefcheck_check_secsafter:nn</code>	797
		<code>__zrefcheck_check_secsbefore:nn</code>	797
		<code>__zrefcheck_check_thischap:nn</code> .	709
		<code>__zrefcheck_check_thispage:nn</code> .	510
		<code>__zrefcheck_check_thispage:nnTF</code>	
		466, 469, 475, 478, 531, 552
		<code>__zrefcheck_check_thissec:nn</code> ..	797
		<code>\l__zrefcheck_checkbeg_tl</code>	
		359, 370, 373, 374, 395
		<code>\l__zrefcheck_checkend_tl</code>	
		359, 372, 393
		<code>\l__zrefcheck_close_range_int</code> ...	
		193, 695
		<code>__zrefcheck_do_check:nnn</code> 17, 423, 434	
		<code>__zrefcheck_end_lblfmt:n</code>	
		. 11, 287, 373, 405, 415, 451, 453,	
		457, 460, 461, 470, 472, 476, 479, 480	
		<code>__zrefcheck_get_asint:nnn</code>	11
		<code>__zrefcheck_get_astl:nnn</code>	11
		<code>\g__zrefcheck_id_int</code> ..	359, 369, 371
		<code>__zrefcheck_int_to_roman:w</code> ...	338
		<code>\l__zrefcheck_integer_bool</code>	
		12, 319, 329,
		513, 517, 533, 537, 583, 587, 601,	
		605, 619, 623, 638, 642, 657, 661,	
		686, 690, 712, 716, 730, 734, 747,	
		751, 765, 769, 782, 786, 800, 806,	
		822, 828, 843, 849, 865, 871, 886, 892	
		<code>__zrefcheck_is_integer:n</code> ...	13, 338
		<code>__zrefcheck_is_integer:nnTF</code> ...	324
		<code>__zrefcheck_is_integer_rgx:n</code> 13, 349	
		<code>\l__zrefcheck_lbl_b_int</code>	
		506, 803, 809,
		825, 831, 846, 852, 868, 874, 889, 895	

U

use commands:

`\use:N`

Z

`\Z`

`\zcheck`

`zcregion`

`\zctarget`

`\zref`

`\zref-check`

zrefcheck commands:

`\zrefcheck_get_asint:nnn`

.....

`\zrefcheck_get_astl:nnn`

.....

zrefcheck internal commands:

`\g__zrefcheck_abschap_int` .

`\g__zrefcheck_abssec_int` .

`\g__zrefcheck_auxfile_lblseq_prop`

`__zrefcheck_check_(check):nn` ...

`__zrefcheck_check_above:nn` ...

`__zrefcheck_check_above:nnTF` ...

`__zrefcheck_check_after:nn` ...

`__zrefcheck_check_before:nn` ...

`__zrefcheck_check_below:nn` ...

`__zrefcheck_check_below:nnTF` ...

`__zrefcheck_check_chapsafter:nn` ...

`__zrefcheck_check_chapsbefore:nn` ...

`__zrefcheck_check_close:nn` ...

`__zrefcheck_check_close:nnTF` ...

`__zrefcheck_check_facing:nn` ...

\l_zrefcheck_lbl_int	\l_zrefcheck_ref_b_int
..... 506, 514, 520, 521, 534, 540, 506, 804, 809,
541, 584, 590, 591, 602, 608, 609,	826, 831, 847, 852, 869, 874, 890, 895
620, 626, 627, 639, 645, 646, 658,	\l_zrefcheck_ref_int
668, 673, 676, 687, 693, 696, 713, 506, 515, 520, 522,
719, 720, 731, 737, 738, 748, 754,	535, 540, 542, 585, 590, 592, 603,
755, 766, 772, 773, 783, 789, 790,	608, 610, 621, 626, 628, 640, 645,
801, 811, 812, 823, 833, 834, 844,	647, 659, 666, 668, 671, 673, 677,
854, 855, 866, 876, 877, 887, 897, 898	688, 693, 697, 714, 719, 721, 732,
\l_zrefcheck_link_anchor_tl ...	737, 749, 754, 756, 767, 772, 784,
..... 359, 388, 389	789, 791, 802, 811, 813, 824, 833,
\l_zrefcheck_link_label_tl	845, 854, 856, 867, 876, 888, 897, 899
..... 359, 375, 377, 387	_zrefcheck_run_checks:nnn
\l_zrefcheck_link_star_tl 16, 395, 417
..... 359, 376, 383	_zrefcheck_target_label:n
_zrefcheck_message:nnnn 40, 492, 497 211, 221, 402, 411
\l_zrefcheck_msglevel_tl ... 42, 133	\l_zrefcheck_target_label_bool .
\l_zrefcheck_msgonpage_bool 166, 488 201, 207, 215
\l_zrefcheck_onpage_bool	\l_zrefcheck_target_label_tl ...
..... 428, 444, 468, 471, 477, 481, 489 200, 217, 218, 219, 224, 230
\c_zrefcheck_onpage_checks_seq .	\l_zrefcheck_use_hyperref_bool .
..... 428, 464 89, 116, 125, 382
\l_zrefcheck_passedcheck_bool ..	\l_zrefcheck_warn_hyperref_bool
..... 428, 443, 449, 452, 458, 462, 484 89, 123
\l_zrefcheck_propval_tl	_zrefcheck_zcheck:nnnnn
..... 320, 323, 324, 326 11, 13, 14, 17, 358, 365
	\zrefchecksetup
	8, 244