

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	Integer testing	4
4.3	Options	5
4.4	Position on page	9
4.5	Counter	11
4.6	Label formats	12
4.7	Property values	12
5	User interface	14
5.1	\zcheck	14
5.2	Targets	16
6	Checks	16
6.1	Running	17
6.2	Conditionals	19
6.2.1	This page	19
6.2.2	On page	20
6.2.3	Before / After	20
6.2.4	Pages	21
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 } { closerange-not-positive-integer }
85 {
86   Option~'closerange'~not~a~positive~integer \iow_newline:
87   on~input~line~\msg_line_number:.~Using~default~value.
88 }
89 \msg_new:nnn { zref-check } { labelcmd-undefined }
90 {
91   Control~sequence~named~'#1'~used~in~option~'labelcmd'~is~not~defined.~
92   Using~default~value.
93 }

```

4.2 Integer testing

`__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.

```

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 `\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.

```

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 }
111 \prg_generate_conditional_variant:Nnn \__zrefcheck_is_integer:rgx:n { x } { TF }

```

(End definition for `__zrefcheck_is_integer:rgx:n`.)

4.3 Options

hyperref option

```

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

```

```

132   hyperref .initial:n = auto ,
133   hyperref .default:n = auto
134 }

(End definition for \l__zrefcheck_use_hyperref_bool and \l__zrefcheck_warn_hyperref_bool.)

```

```

135 \AtBeginDocument
136 {
137   \ifpackageloaded { hyperref }
138   {
139     \bool_if:NT \l__zrefcheck_use_hyperref_bool
140     {
141       \RequirePackage { zref-hyperref }
142       \zref@addprop { zrefcheck } { anchor }
143     }
144   }
145   {
146     \bool_if:NT \l__zrefcheck_warn_hyperref_bool
147     { \msg_warning:nn { zref-check } { missing-hyperref } }
148     \bool_set_false:N \l__zrefcheck_use_hyperref_bool
149   }
150   \keys_define:nn { zref-check }
151   {
152     hyperref .code:n =
153     { \msg_warning:nn { zref-check } { hyperref-preamble-only } }
154   }
155 }

```

msglevel option

\l__zrefcheck_msglevel_tl

```

156 \tl_new:N \l__zrefcheck_msglevel_tl
157 \keys_define:nn { zref-check }
158 {
159   msglevel .choice: ,
160   msglevel / warn .code:n =
161   { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } } ,
162   msglevel / info .code:n =
163   { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } } ,
164   msglevel / none .code:n =
165   { \tl_set:Nn \l__zrefcheck_msglevel_tl { none } } ,
166   msglevel / obeydraft .code:n =
167   {
168     \ifdraft
169     { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
170     { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
171   } ,
172   msglevel / obeyfinal .code:n =
173   {
174     \ifoptionfinal
175     { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
176     { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
177   } ,
178   msglevel .value_required:n = true ,
179   msglevel .initial:n = warn ,

```

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

```

180   ignore .code:n =
181     { \msg_warning:nn { zref-check } { ignore-document-only } } ,
182   ignore .value_forbidden:n = true
183 }

```

(End definition for \l__zrefcheck_msglevel_tl.)

```

184 \AtBeginDocument
185 {
186   \keys_define:nn { zref-check }
187     { ignore .meta:n = { msglevel = none } }
188 }

```

`onpage` option

`\l__zrefcheck_msgonpage_bool`

```

189 \bool_new:N \l__zrefcheck_msgonpage_bool
190 \keys_define:nn { zref-check }
191 {
192   onpage .choice: ,
193   onpage / labelseq .code:n =
194     {
195       \bool_set_false:N \l__zrefcheck_msgonpage_bool
196     } ,
197   onpage / msg .code:n =
198     {
199       \bool_set_true:N \l__zrefcheck_msgonpage_bool
200     } ,
201   onpage / obeydraft .code:n =
202     {
203       \ifdraft
204         { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
205         { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
206     } ,
207   onpage / obeyfinal .code:n =
208     {
209       \ifoptionfinal
210         { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
211         { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
212     } ,
213   onpage .value_required:n = true ,
214   onpage .initial:n = labelseq
215 }

```

(End definition for \l__zrefcheck_msgonpage_bool.)

`closerange` option

`\l__zrefcheck_close_range_int`

```

216 \int_new:N \l__zrefcheck_close_range_int
217 \keys_define:nn { zref-check }
218 {
219   closerange .code:n =
220     {
221       \__zrefcheck_is_integer_rgx:nTF {#1}

```

```

222     { \int_set:Nn \l__zrefcheck_close_range_int { \int_eval:n {#1} } }
223     {
224       \msg_warning:nn { zref-check } { closerange-not-positive-integer }
225       \int_set:Nn \l__zrefcheck_close_range_int { 5 }
226     }
227   } ,
228   closerange .value_required:n = true ,
229   closerange .initial:n = 5
230 }

```

(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>.

```

231 \tl_new:N \l__zrefcheck_target_label_tl
232 \bool_new:N \l__zrefcheck_target_label_bool
233 \keys_define:nn { zref-check }
234 {
235   labelcmd .code:n =
236   {
237     \tl_set:Nn \l__zrefcheck_target_label_tl {#1}
238     \bool_set_true:N \l__zrefcheck_target_label_bool
239   } ,
240   labelcmd .value_required:n = true ,
241 }

```

(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.

```

242 \cs_new:Npn \__zrefcheck_target_label:n #1
243 { \zref@labelbylist {#1} { zrefcheck } }

```

(End definition for __zrefcheck_target_label:n.)

```

244 \AtBeginDocument
245 {
246   \bool_if:NT \l__zrefcheck_target_label_bool
247   {
248     \tl_if_blank:VT \l__zrefcheck_target_label_tl
249     { \tl_clear:N \l__zrefcheck_target_label_tl }
250     \cs_if_exist:cTF { \l__zrefcheck_target_label_tl }
251     {
252       \cs_set:Npx \__zrefcheck_target_label:n #1
253       {
254         \exp_not:o
255         { \cs:w \l__zrefcheck_target_label_tl \cs_end: }
256         {#1}
257       }
258     }
259     {
260       \exp_args:NnnV \msg_warning:nnn { zref-check }
261       { labelcmd-undefined } { \l__zrefcheck_target_label_tl }
262     }
263   }
264 }

```



```

263     }
264     \keys_define:nn { zref-check }
265     {
266         labelcmd .code:n =
267         {
268             \msg_warning:nnn { zref-check }
269             { option-preamble-only } { labelcmd }
270         }
271     }
272 }

```

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

```

273 \RequirePackage { l3keys2e }
274 \ProcessKeysOptions { zref-check }

```

`\zrefchecksetup` Provide `\zrefchecksetup`.

```

275 \NewDocumentCommand \zrefchecksetup { m }
276 { \keys_set:nn { zref-check } {#1} }

```

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

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 `ltxcmdhooks` 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`

```
277 \prop_new:N \g_zrefcheck_auxfile_lblseq_prop
```

(End definition for `\g_zrefcheck_auxfile_lblseq_prop`.)

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

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

```
280 {
```

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.

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

```
282 \group_begin:
```

```
283 \int_zero:N \l_tmpa_int
```

```
284 \tl_clear:N \l_tmpa_tl
```

```
285 \tl_clear:N \l_tmpb_tl
```

```
286 \bool_set_false:N \l_tmpa_bool
```

```
287 \ior_map_variable:NNn \g_tmpa_ior \l_tmpa_tl
```

```

288     {
289         \tl_map_variable:Nn \l_tmpa_tl \l_tmpb_tl
290         {
291             \tl_if_eq:NnTF \l_tmpb_tl { \zref@newlabel }
292             {
Found a \zref@label, signal it.
293                 \bool_set_true:N \l_tmpa_bool
294             }
295             {
296                 \bool_if:NTF \l_tmpa_bool
297                 {
298                     \bool_set_false:N \l_tmpa_bool
299                     \int_incr:N \l_tmpa_int
300                     \prop_gput:Nxx \g__zrefcheck_auxfile_lblseq_prop
301                     { \l_tmpb_tl } { \int_use:N \l_tmpa_int }
302                 }
303             {
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.
304                 \tl_map_break:
305             }
306         }
307     }
308 }
309 \group_end:
310 \ior_close:N \g_tmpa_ior
311 }

```

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.

4.5 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-abspace.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.

```

312 \begingroup
313   \let \@addtoreset \ltx@gobbletwo
314   \newcounter { zrefcheck }
315 \endgroup
316 \setcounter { zrefcheck } { 0 }

```

4.6 Label formats

```

\__zrefcheck_check_lblfmt:n      \__zrefcheck_check_lblfmt:n {<check id int>}
317 \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>}
318 \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. 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>}

319 \cs_new:Npn \zrefcheck_get_astl:nnn #1#2#3
320 {
321   \tl_clear:N #3
322   \tl_if_eq:nnTF {#2} { lblseq }
323   {
324     \prop_get:NnNF \g__zrefcheck_auxfile_lblseq_prop {#1} #3
325     {
326       \msg_warning:nnnn { zref-check }
327       { property-not-in-label } {#1} {#2}
328     }
329   }
330 {

```

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.

```

331     \zref@ifrefundefined {#1}
332     {}
333     {
334         \zref@ifpropundefined {#2}
335         { \msg_warning:nnnn { zref-check } { property-undefined } {#2} }
336         {
337             \zref@ifrefcontainsprop {#1} {#2}
338             {
339                 \tl_set:Nx #3
340                 { \zref@extractdefault {#1} {#2} { \c_empty_tl } }
341             }
342             {
343                 \msg_warning:nnnn
344                 { zref-check } { property-not-in-label } {#1} {#2}
345             }
346         }
347     }
348 }
349 }
```

(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).

```

350 \bool_new:N \l__zrefcheck_integer_bool
```

(End definition for `\l__zrefcheck_integer_bool`.)

`\l__zrefcheck_propval_tl`

```

351 \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)}
352 \cs_new:Npn \zrefcheck_get_asint:nnn #1#2#3
353 {
354   \zrefcheck_get_astl:nnn {#1} {#2} { \l__zrefcheck_propval_tl }
355   \__zrefcheck_is_integer:nTF { \l__zrefcheck_propval_tl }
356   {

```

Make it an integer data type.

```

357     \int_set:Nn #3 { \int_eval:n { \l__zrefcheck_propval_tl } }
358   }
359   {
360     \bool_set_false:N \l__zrefcheck_integer_bool
361     \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`.

```

362     { }
363   {
364     \msg_warning:nnnn { zref-check }
365     { property-not-integer } {#2} {#1}
366   }
367 }
368 }

```

(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{*}[\options][\labels][\checks]{\text}

```

```

369 \NewDocumentCommand \zcheck
370 { s O { } } > { \SplitList { , } } m > { \SplitList { , } } O { } m }
371 { \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 372 \int_new:N \g__zrefcheck_id_int
\l__zrefcheck_checkend_tl 373 \tl_new:N \l__zrefcheck_checkbeg_tl
\l__zrefcheck_link_label_tl 374 \tl_new:N \l__zrefcheck_checkend_tl
\l__zrefcheck_link_anchor_tl 375 \tl_new:N \l__zrefcheck_link_label_tl
\l__zrefcheck_link_star_tl 376 \tl_new:N \l__zrefcheck_link_anchor_tl
377 \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 `{\labels}` 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 {\labels} {\(*)} {\options} {\checks} {\text}
```

```
378 \cs_new:Npn \__zrefcheck_zcheck:nnnnn #1#2#3#4#5
```

```
379 {
```

```
380   \group_begin:
```

Process local options.

```
381   \keys_set:nn { zref-check } {#3}
```

Names of the labels for this zrefcheck call.

```
382   \int_gincr:N \g__zrefcheck_id_int
```

```
383   \tl_set:Nx \l__zrefcheck_checkbeg_tl
```

```
384     { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
```

```
385   \tl_set:Nx \l__zrefcheck_checkend_tl
```

```
386     { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
```

Set checkbeg label.

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

Typeset `{\text}`, 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.

```
388   \tl_set:Nn \l__zrefcheck_link_label_tl { \tl_head:n {#1} }
```

```
389   \bool_set:Nn \l__zrefcheck_link_star_tl {#2}
```

```
390   \zref@ifrefundefined { \l__zrefcheck_link_label_tl }
```

If the reference is undefined, just typeset.

```
391     {#5}
```

```
392   {
```

```
393     \bool_if:nTF
```

```
394       {
```

```
395         \l__zrefcheck_use_hyperref_bool &&
```

```
396         ! \l__zrefcheck_link_star_tl
```

```
397       }
```

```
398     {
```

```
399       \exp_args:Nx \zrefcheck_get_astl:nnn
```

```
400         { \l__zrefcheck_link_label_tl }
```

```
401         { anchor } { \l__zrefcheck_link_anchor_tl }
```

```
402         \hyperlink { \l__zrefcheck_link_anchor_tl } {#5}
```

```
403       }
```

```
404     {#5}
```

```
405   }
```

Set checkend label.

```
406   \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`.

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

Run the checks.

```
408   \__zrefcheck_run_checks:nnV {#4} {#1} { \l__zrefcheck_checkbeg_tl }
```

```
409   \group_end:
```

```
410 }
```

(End definition for `_zrefcheck_zcheck:nnnnn`.)

5.2 Targets

```
\zctarget      \zctarget{<label>}{<text>}
411 \NewDocumentCommand \zctarget { m +m }
412 {
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.
413   \group_begin:
414   \refstepcounter { zrefcheck }
415   \zref@wrapper@babel \_zrefcheck_target_label:n {#1}
416   #2
417   \zref@wrapper@babel
418   \zref@labelbylist { \_zrefcheck_end_lblfmt:n {#1} } { zrefcheck }
419   \group_end:
420 }
```

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

```
zcregion      \begin{zcregion}{<label>}
...
\end{zcregion}
421 \NewDocumentEnvironment {zcregion} { m }
422 {
423   \refstepcounter { zrefcheck }
424   \zref@wrapper@babel \_zrefcheck_target_label:n {#1}
425 }
426 {
427   \zref@wrapper@babel
428   \zref@labelbylist { \_zrefcheck_end_lblfmt:n {#1} } { zrefcheck }
429 }
```

(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:nnn {\<checks>} {\<labels>} {\<reference>}
\__zrefcheck_run_checks:nnV
430 \cs_new:Npn \__zrefcheck_run_checks:nnn #1#2#3
431 {
432   \group_begin:
433     \tl_map_inline:nn {#2}
434     {
435       \tl_map_inline:nn {#1}
436       { \__zrefcheck_do_check:nnn {####1} {##1} {#3} }
437     }
438   \group_end:
439 }
440 \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
441 \bool_new:N \l__zrefcheck_passedcheck_bool
442 \bool_new:N \l__zrefcheck_onpage_bool
443 \seq_new:N \c__zrefcheck_onpage_checks_seq
444 \seq_set_from_clist:Nn \c__zrefcheck_onpage_checks_seq
445 { 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.
446 \cs_generate_variant:Nn \exp_args:Nnno { Nnoo }

\__zrefcheck_do_check:nnn      \__zrefcheck_do_check:nnn {\<check>} {\<label beg>} {\<reference beg>}
447 \cs_new:Npn \__zrefcheck_do_check:nnn #1#2#3
448 {
449   \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.

450   \zref@ifrefundefined {#2}
451   {}
452   {
453     \tl_if_empty:nTF {#1}
454     {}
455     {

```

```

456         \bool_set_true:N \l__zrefcheck_passedcheck_bool
457         \bool_set_false:N \l__zrefcheck_onpage_bool
458         \cs_if_exist:cTF { __zrefcheck_check_ #1 :nnF }
459         {

```

“label beg” vs “reference beg”.

```

460         \use:c { __zrefcheck_check_ #1 :nnF }
461         {#2} {#3}
462         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }

```

“label beg” vs “reference end”.

```

463         \exp_args:Nnno \use:c { __zrefcheck_check_ #1 :nnF }
464         {#2} { \__zrefcheck_end_lblfmt:n {#3} }
465         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }

```

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

```

466         \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
467         {}
468         {

```

“label end” vs “reference beg”.

```

469         \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
470         { \__zrefcheck_end_lblfmt:n {#2} } {#3}
471         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }

```

“label end” vs “reference end”.

```

472         \exp_args:Nnoo \use:c { __zrefcheck_check_ #1 :nnF }
473         { \__zrefcheck_end_lblfmt:n {#2} } }
474         { \__zrefcheck_end_lblfmt:n {#3} } }
475         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
476     }

```

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.

```

477         \seq_if_in:NnT \c__zrefcheck_onpage_checks_seq {#1}
478         {
479             \__zrefcheck_check_thispage:nnT
480             {#2} {#3}
481             { \bool_set_true:N \l__zrefcheck_onpage_bool }
482             \__zrefcheck_check_thispage:nnT
483             {#2} { \__zrefcheck_end_lblfmt:n {#3} }
484             { \bool_set_true:N \l__zrefcheck_onpage_bool }
485             \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
486             {}
487             {
488                 \__zrefcheck_check_thispage:nnT
489                 { \__zrefcheck_end_lblfmt:n {#2} } {#3}
490                 { \bool_set_true:N \l__zrefcheck_onpage_bool }
491                 \__zrefcheck_check_thispage:nnT
492                 { \__zrefcheck_end_lblfmt:n {#2} }
493                 { \__zrefcheck_end_lblfmt:n {#3} }
494                 { \bool_set_true:N \l__zrefcheck_onpage_bool }
495             }
496         }
497         \bool_if:NTF \l__zrefcheck_passedcheck_bool

```

```

498         {
499             \bool_if:nT
500             {
501                 \l__zrefcheck_msgonpage_bool &&
502                 \l__zrefcheck_onpage_bool
503             }
504             {
505                 \__zrefcheck_message:nnnx { double-check } {#1} {#2}
506                 { \zref@extractdefault {#3} {page} {'unknown'} }
507             }
508         }
509         {
510             \__zrefcheck_message:nnnx { check-failed } {#1} {#2}
511             { \zref@extractdefault {#3} {page} {'unknown'} }
512         }
513     }
514     { \msg_warning:nnn { zref-check } { check-missing } {#1} }
515 }
516 }
517 \group_end:
518 }

```

(End definition for __zrefcheck_do_check:nnn.)

6.2 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.

```

519 \int_new:N \l__zrefcheck_lbl_int
520 \int_new:N \l__zrefcheck_ref_int
521 \int_new:N \l__zrefcheck_lbl_b_int
522 \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

```

```

523 \prg_new_conditional:Npnn \__zrefcheck_check_thispage:nn #1#2 { T , F , TF }
524 {
525     \group_begin:
526     \bool_set_true:N \l__zrefcheck_integer_bool
527     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
528     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
529     \bool_lazy_and:nnTF
530     { \l__zrefcheck_integer_bool }
531     {
532         \int_compare_p:nNn
533         { \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.

```

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

```

```

535         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
536     }
537     { \group_insert_after:N \prg_return_true: }
538     { \group_insert_after:N \prg_return_false: }
539 \group_end:
540 }

```

(End definition for __zrefcheck_check_thispage:nn.)

6.2.2 On page

__zrefcheck_check_above:nn

__zrefcheck_check_below:nn

```

541 \prg_new_conditional:Npnn \__zrefcheck_check_above:nn #1#2 { F , TF }
542 {
543     \group_begin:
544     \__zrefcheck_check_thispage:nnTF {#1} {#2}
545     {
546         \bool_set_true:N \l__zrefcheck_integer_bool
547         \zrefcheck_get_asint:nnn {#1} { lblseq } { \l__zrefcheck_lbl_int }
548         \zrefcheck_get_asint:nnn {#2} { lblseq } { \l__zrefcheck_ref_int }
549         \bool_lazy_and:nnTF
550         { \l__zrefcheck_integer_bool }
551         {
552             \int_compare_p:nNn
553             { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
554             ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
555             ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
556         }
557         { \group_insert_after:N \prg_return_true: }
558         { \group_insert_after:N \prg_return_false: }
559     }
560     { \group_insert_after:N \prg_return_false: }
561 \group_end:
562 }
563 \prg_new_conditional:Npnn \__zrefcheck_check_below:nn #1#2 { F , TF }
564 {
565     \__zrefcheck_check_thispage:nnTF {#1} {#2}
566     {
567         \__zrefcheck_check_above:nnTF {#1} {#2}
568         { \prg_return_false: }
569         { \prg_return_true: }
570     }
571     { \prg_return_false: }
572 }

```

(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

```

573 \prg_new_conditional:Npnn \__zrefcheck_check_before:nn #1#2 { F }
574 {
575     \__zrefcheck_check_pagesbefore:nnTF {#1} {#2}
576     { \prg_return_true: }

```

```

577     {
578       \__zrefcheck_check_above:nnTF {#1} {#2}
579       { \prg_return_true:  }
580       { \prg_return_false: }
581     }
582   }
583 \prg_new_conditional:Npnn \__zrefcheck_check_after:nn #1#2 { F }
584 {
585   \__zrefcheck_check_pagesafter:nnTF {#1} {#2}
586   { \prg_return_true:  }
587   {
588     \__zrefcheck_check_below:nnTF {#1} {#2}
589     { \prg_return_true:  }
590     { \prg_return_false: }
591   }
592 }

```

(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
593 \prg_new_conditional:Npnn \__zrefcheck_check_nextpage:nn #1#2 { F }
594 {
595   \group_begin:
596     \bool_set_true:N \l__zrefcheck_integer_bool
597     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
598     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
599     \bool_lazy_and:nnTF
600     { \l__zrefcheck_integer_bool }
601     {
602       \int_compare_p:nNn
603       { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
604       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
605       ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
606     }
607     { \group_insert_after:N \prg_return_true:  }
608     { \group_insert_after:N \prg_return_false: }
609   \group_end:
610 }
611 \prg_new_conditional:Npnn \__zrefcheck_check_prevpage:nn #1#2 { F }
612 {
613   \group_begin:
614     \bool_set_true:N \l__zrefcheck_integer_bool
615     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
616     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
617     \bool_lazy_and:nnTF
618     { \l__zrefcheck_integer_bool }
619     {
620       \int_compare_p:nNn
621       { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
622       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
623       ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
624     }
625   \group_end:
626 }

```

```

625         { \group_insert_after:N \prg_return_true: }
626         { \group_insert_after:N \prg_return_false: }
627     \group_end:
628 }
629 \prg_new_conditional:Npnn \__zrefcheck_check_pagesbefore:nn #1#2 { 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 } &&
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 }
647 \cs_new_eq:NN \__zrefcheck_check_ppbefore:nnF \__zrefcheck_check_pagesbefore:nnF
648 \prg_new_conditional:Npnn \__zrefcheck_check_pagesafter:nn #1#2 { F , TF }
649 {
650     \group_begin:
651     \bool_set_true:N \l__zrefcheck_integer_bool
652     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
653     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
654     \bool_lazy_and:nnTF
655     { \l__zrefcheck_integer_bool }
656     {
657         \int_compare_p:nNn
658         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
659         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
660         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
661     }
662     { \group_insert_after:N \prg_return_true: }
663     { \group_insert_after:N \prg_return_false: }
664     \group_end:
665 }
666 \cs_new_eq:NN \__zrefcheck_check_ppafter:nnF \__zrefcheck_check_pagesafter:nnF
667 \prg_new_conditional:Npnn \__zrefcheck_check_facing:nn #1#2 { F }
668 {
669     \group_begin:
670     \bool_set_true:N \l__zrefcheck_integer_bool
671     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
672     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
673     \bool_lazy_and:nnTF
674     { \l__zrefcheck_integer_bool }
675     {

```

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

```

676         \legacy_if_p:n { @twoside } &&

```

Now we test “facing”.

```

677      (
678      (
679          \int_if_odd_p:n { \l__zrefcheck_ref_int } &&
680          \int_compare_p:nNn
681              { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 }
682      ) ||
683      (
684          \int_if_even_p:n { \l__zrefcheck_ref_int } &&
685          \int_compare_p:nNn
686              { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 }
687      )
688      ) &&
689      ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
690      ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
691      }
692      { \group_insert_after:N \prg_return_true: }
693      { \group_insert_after:N \prg_return_false: }
694  \group_end:
695  }

```

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

6.2.5 Close / Far

```

\__zrefcheck_check_close:nn
\__zrefcheck_check_far:nn
696 \prg_new_conditional:Npnn \__zrefcheck_check_close:nn #1#2 { F , TF }
697 {
698     \group_begin:
699     \bool_set_true:N \l__zrefcheck_integer_bool
700     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
701     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
702     \bool_lazy_and:nnTF
703     { \l__zrefcheck_integer_bool }
704     {
705         \int_compare_p:nNn
706         { \int_abs:n { \l__zrefcheck_lbl_int - \l__zrefcheck_ref_int } }
707         <
708         { \l__zrefcheck_close_range_int + 1 } &&
709         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
710         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
711     }
712     { \group_insert_after:N \prg_return_true: }
713     { \group_insert_after:N \prg_return_false: }
714 \group_end:
715 }
716 \prg_new_conditional:Npnn \__zrefcheck_check_far:nn #1#2 { F }
717 {
718     \__zrefcheck_check_close:nnTF {#1} {#2}
719     { \prg_return_false: }
720     { \prg_return_true: }
721 }

```

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

```

‘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.


```

767         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
768         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
769         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
770     }
771     { \group_insert_after:N \prg_return_true: }
772     { \group_insert_after:N \prg_return_false: }
773 \group_end:
774 }
775 \prg_new_conditional:Npnn \__zrefcheck_check_chapsafter:nn #1#2 { F }
776 {
777     \group_begin:
778     \bool_set_true:N \l__zrefcheck_integer_bool
779     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
780     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
781     \bool_lazy_and:nnTF
782     { \l__zrefcheck_integer_bool }
783     {
784         \int_compare_p:nNn
785         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
786         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
787     }
788     { \group_insert_after:N \prg_return_true: }
789     { \group_insert_after:N \prg_return_false: }
790 \group_end:
791 }
792 \prg_new_conditional:Npnn \__zrefcheck_check_chapsbefore:nn #1#2 { F }
793 {
794     \group_begin:
795     \bool_set_true:N \l__zrefcheck_integer_bool
796     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
797     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
798     \bool_lazy_and:nnTF
799     { \l__zrefcheck_integer_bool }
800     {
801         \int_compare_p:nNn
802         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
803         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
804         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
805     }
806     { \group_insert_after:N \prg_return_true: }
807     { \group_insert_after:N \prg_return_false: }
808 \group_end:
809 }

```

(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_secsover:nn
\__zrefcheck_check_secsover:nn
810 \prg_new_conditional:Npnn \__zrefcheck_check_thissec:nn #1#2 { F }
811 {
812     \group_begin:
813     \bool_set_true:N \l__zrefcheck_integer_bool
814     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }

```

```

815 \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
816 \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
817 \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
818 \bool_lazy_and:nnTF
819 { \l__zrefcheck_integer_bool }
820 {
821   \int_compare_p:nNn
822     { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
823   \int_compare_p:nNn
824     { \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.

```

825   ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
826   ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
827 }
828 { \group_insert_after:N \prg_return_true: }
829 { \group_insert_after:N \prg_return_false: }
830 \group_end:
831 }
832 \prg_new_conditional:Npnn \__zrefcheck_check_nextsec:nn #1#2 { F }
833 {
834   \group_begin:
835     \bool_set_true:N \l__zrefcheck_integer_bool
836     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
837     \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
838     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
839     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
840     \bool_lazy_and:nnTF
841     { \l__zrefcheck_integer_bool }
842     {
843       \int_compare_p:nNn
844         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
845       \int_compare_p:nNn
846         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
847       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
848     }
849     { \group_insert_after:N \prg_return_true: }
850     { \group_insert_after:N \prg_return_false: }
851   \group_end:
852 }
853 \prg_new_conditional:Npnn \__zrefcheck_check_prevsec:nn #1#2 { F }
854 {
855   \group_begin:
856     \bool_set_true:N \l__zrefcheck_integer_bool
857     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
858     \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
859     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
860     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
861     \bool_lazy_and:nnTF
862     { \l__zrefcheck_integer_bool }

```

```

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

```

```

917 }
(End definition for \_zrefcheck\_check\_thissec:nn and others.)
918 \end{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.

A		E	
\A	107	\endgroup	315
\AddToHook	21, 28	\endinput	12
\AtBeginDocument	135, 184, 244	exp commands:	
B		\exp_args:Nnno	446, 463
\begingroup	312	\exp_args:NnnV	260
bool commands:		\exp_args:Nno	469
\bool_if:NTF	139, 146, 246, 296, 497	\exp_args:Nnoo	472
\bool_if:nTF	393, 499	\exp_args:Nx	399
\bool_lazy_and:nnTF		\exp_not:n	254
	13, 529, 549, 599,		
	617, 635, 654, 673, 702, 728, 746,		
	763, 781, 798, 818, 840, 861, 883, 904		
\bool_new:N			
	112, 113, 189, 232, 350, 377, 441, 442		
\bool_set:Nn	389		
\bool_set_false:N			
	120, 129, 130, 148, 195, 204, 211,		
	286, 298, 360, 457, 462, 465, 471, 475		
\bool_set_true:N	119, 124,		
	125, 199, 205, 210, 238, 293, 456,		
	481, 484, 490, 494, 526, 546, 596,		
	614, 632, 651, 670, 699, 725, 743,		
	760, 778, 795, 813, 835, 856, 878, 899		
\l_tmpa_bool	11, 286, 293, 296, 298		
C		F	
\catcode	11	file commands:	
\chapter	2, 24	\file_if_exist:nTF	279
cs commands:		\fmtversion	3
\cs:w	255		
\cs_end:	255		
\cs_generate_variant:Nn	45, 440, 446		
\cs_if_exist:NTF	250, 458		
\cs_new:Npn	40,		
	242, 317, 318, 319, 352, 378, 430, 447		
\cs_new_eq:NN	94, 647, 666		
\cs_set:Npx	252		
D		G	
\d	107	group commands:	
		\group_begin:	282,
			380, 413, 432, 449, 525, 543, 595,
			613, 631, 650, 669, 698, 724, 742,
			759, 777, 794, 812, 834, 855, 877, 898
		\group_end:	309,
			409, 419, 438, 517, 539, 561, 609,
			627, 645, 664, 694, 714, 738, 755,
			773, 790, 808, 830, 851, 873, 894, 916
		\group_insert_after:N	537,
			538, 557, 558, 560, 607, 608, 625,
			626, 643, 644, 662, 663, 692, 693,
			712, 713, 736, 737, 753, 754, 771,
			772, 788, 789, 806, 807, 828, 829,
			849, 850, 871, 872, 892, 893, 914, 915
		H	
		\hyperlink	14, 402
		I	
		\ifdraft	168, 203
		\IfFormatAtLeastTF	3, 4
		\ifoptionfinal	174, 209
		int commands:	
		\int_abs:n	706

<code>\@ifl@t@r</code>	3	<code>\zref</code>	14
<code>\@ifpackageloaded</code>	137	<code>\zref-check</code>	2
<code>\@newl@bel</code>	9	zrefcheck commands:	
<code>\ltx@gobbletwo</code>	313	<code>\zrefcheck_get_asint:nnn</code>	
<code>\zref@addprop</code>	17, 27, 31, 142	13, 16, 352, 527, 528, 547,
<code>\zref@addprops</code>	33	548, 597, 598, 615, 616, 633, 634,
<code>\zref@extractdefault</code>	12, 340, 506, 511	652, 653, 671, 672, 700, 701, 726,
<code>\zref@ifpropundefined</code>	334	727, 744, 745, 761, 762, 779, 780,
<code>\zref@ifrefcontainsprop</code>	337	796, 797, 814, 815, 816, 817, 836,
<code>\zref@ifrefundefined</code>	837, 838, 839, 857, 858, 859, 860,
.....	12, 331, 361, 390, 450, 466, 485	879, 880, 881, 882, 900, 901, 902, 903
<code>\ZREF@label</code>	11	<code>\zrefcheck_get_astl:nnn</code>	
<code>\zref@label</code>	10, 11	12, 13, 16, 319, 354, 399
<code>\zref@labelbylist</code>		zrefcheck internal commands:	
.....	243, 387, 406, 418, 428	<code>\g_zrefcheck_abschap_int</code> ..	19, 23, 26
<code>\ZREF@mainlist</code>	27, 31	<code>\g_zrefcheck_abssec_int</code> ..	19, 24, 29, 30
<code>\ZREF@newlabel</code>	9–11, 291	<code>\g_zrefcheck_auxfile_lblseq-</code>	
<code>\zref@newlist</code>	32	prop	12, 17, 277, 300, 324
<code>\zref@newprop</code>	17, 26, 30	<code>__zrefcheck_check_<check>:nn</code> ..	16
<code>\zref@refused</code>	12, 15, 407	<code>__zrefcheck_check_above:nn</code> ..	541
<code>\zref@require@unique</code>	11	<code>__zrefcheck_check_above:nnTF</code> ..	
<code>\zref@wrapper@babel</code>	567, 578
...	9, 14, 15, 371, 415, 417, 424, 427	<code>__zrefcheck_check_after:nn</code> ..	573
tex commands:		<code>__zrefcheck_check_before:nn</code> ..	573
<code>\tex_romannumeral:D</code>	5	<code>__zrefcheck_check_below:nn</code> ..	541
tl commands:		<code>__zrefcheck_check_below:nnTF</code> ..	588
<code>\c_empty_tl</code>	12, 340	<code>__zrefcheck_check_chapsafter:nn</code>	722
<code>\tl_clear:N</code>	12, 249, 284, 285, 321	<code>__zrefcheck_check_chapsbefore:nn</code>	
<code>\tl_head:n</code>	388	722
<code>\tl_if_blank:nTF</code>	4, 248	<code>__zrefcheck_check_close:nn</code> ..	696
<code>\tl_if_empty:nTF</code>	4, 97, 100, 453	<code>__zrefcheck_check_close:nnTF</code> ..	718
<code>\tl_if_eq:NnTF</code>	291	<code>__zrefcheck_check_facing:nn</code> ..	593
<code>\tl_if_eq:nnTF</code>	322	<code>__zrefcheck_check_far:nn</code>	696
<code>\tl_map_break:</code>	304	<code>__zrefcheck_check_lblfmt:n</code>	
<code>\tl_map_function:nN</code>	407	12, 317, 384
<code>\tl_map_inline:nn</code>	433, 435	<code>__zrefcheck_check_nextchap:nn</code> ..	722
<code>\tl_map_variable:NNn</code>	289	<code>__zrefcheck_check_nextpage:nn</code> ..	593
<code>\tl_new:N</code>		<code>__zrefcheck_check_nextsec:nn</code> ..	810
.....	156, 231, 351, 373, 374, 375, 376	<code>__zrefcheck_check_pagesafter:nn</code>	593
<code>\tl_set:Nn</code> ..	161, 163, 165, 169, 170,	<code>__zrefcheck_check_pagesafter:nnTF</code>	
.....	175, 176, 237, 278, 339, 383, 385, 388	585, 666
<code>\g_tmpa_tl</code>	278, 279, 281	<code>__zrefcheck_check_pagesbefore:nn</code>	
<code>\l_tmpa_tl</code>	284, 287, 289	593
<code>\l_tmpb_tl</code>	285, 289, 291, 301	<code>__zrefcheck_check_pagesbefore:nnTF</code>	
U		575, 647
use commands:		<code>__zrefcheck_check_ppafter:nn</code> ..	593
<code>\use:N</code>	42, 460, 463, 469, 472	<code>__zrefcheck_check_ppafter:nnTF</code>	666
Z		<code>__zrefcheck_check_ppbefore:nn</code> ..	593
<code>\Z</code>	107	<code>__zrefcheck_check_ppbefore:nnTF</code>	647
<code>\zcheck</code>	14, 16, 70, 369	<code>__zrefcheck_check_prevchap:nn</code> ..	722
<code>zcregion</code>	421	<code>__zrefcheck_check_prevpage:nn</code> ..	593
<code>\zctarget</code>	8, 16, 411	<code>__zrefcheck_check_prevsec:nn</code> ..	810
		<code>__zrefcheck_check_secsafter:nn</code>	810
		<code>__zrefcheck_check_secsbefore:nn</code>	810

_zrefcheck_check_thischap:nn	722	_l_zrefcheck_link_label_tl	372, 388, 390, 400
_zrefcheck_check_thispage:nn	523	_l_zrefcheck_link_star_tl	372, 389, 396
_zrefcheck_check_thispage:nnTF	479, 482, 488, 491, 544, 565	_zrefcheck_message:nnnn	40, 505, 510
_zrefcheck_check_thissec:nn	810	_l_zrefcheck_msglevel_tl	42, 156
_l_zrefcheck_checkbeg_tl	372, 383, 386, 387, 408	_l_zrefcheck_msgonpage_bool	189, 501
_l_zrefcheck_checkend_tl	372, 385, 406	_l_zrefcheck_onpage_bool	441, 457, 481, 484, 490, 494, 502
_l_zrefcheck_close_range_int	216, 708	_c_zrefcheck_onpage_checks_seq	441, 477
_zrefcheck_do_check:nnn	17, 436, 447	_l_zrefcheck_passedcheck_bool	441, 456, 462, 465, 471, 475, 497
_zrefcheck_end_lblfmt:n	12, 318, 386, 418, 428, 464, 466, 470, 473, 474, 483, 485, 489, 492, 493	_l_zrefcheck_propval_tl	351, 354, 355, 357
_zrefcheck_get_asint:nnn	12	_l_zrefcheck_ref_b_int	519, 817, 822, 839, 844, 860, 865, 882, 887, 903, 908
_zrefcheck_get_astl:nnn	12	_l_zrefcheck_ref_int	519, 528, 533, 535, 548, 553, 555, 598, 603, 605, 616, 621, 623, 634, 639, 641, 653, 658, 660, 672, 679, 681, 684, 686, 690, 701, 706, 710, 727, 732, 734, 745, 750, 762, 767, 769, 780, 785, 797, 802, 804, 815, 824, 826, 837, 846, 858, 867, 869, 880, 889, 901, 910, 912
_g_zrefcheck_id_int	372, 382, 384	_zrefcheck_run_checks:nnn	17, 408, 430
_zrefcheck_int_to_roman:w	94	_zrefcheck_target_label:n	242, 252, 415, 424
_l_zrefcheck_integer_bool	13, 350, 360, 526, 530, 546, 550, 596, 600, 614, 618, 632, 636, 651, 655, 670, 674, 699, 703, 725, 729, 743, 747, 760, 764, 778, 782, 795, 799, 813, 819, 835, 841, 856, 862, 878, 884, 899, 905	_l_zrefcheck_target_label_bool	232, 238, 246
_zrefcheck_is_integer:n	5, 94	_l_zrefcheck_target_label_tl	231, 248, 249, 250, 255, 261
_zrefcheck_is_integer:nTF	355	_l_zrefcheck_use_hyperref_bool	112, 139, 148, 395
_zrefcheck_is_integer_rgx:n	5, 105	_l_zrefcheck_warn_hyperref_bool	112, 146
_zrefcheck_is_integer_rgx:nTF	221	_zrefcheck_zcheck:nnnnn	12, 14, 17, 371, 378
_l_zrefcheck_lbl_b_int	519, 816, 822, 838, 844, 859, 865, 881, 887, 902, 908	_zrefchecksetup	9, 275
_l_zrefcheck_lbl_int	519, 527, 533, 534, 547, 553, 554, 597, 603, 604, 615, 621, 622, 633, 639, 640, 652, 658, 659, 671, 681, 686, 689, 700, 706, 709, 726, 732, 733, 744, 750, 751, 761, 767, 768, 779, 785, 786, 796, 802, 803, 814, 824, 825, 836, 846, 847, 857, 867, 868, 879, 889, 890, 900, 910, 911		
_l_zrefcheck_link_anchor_tl	372, 401, 402		