

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	7
4.4	Counter	9
4.5	Label formats	9
4.6	Property values	10
5	User interface	12
5.1	\zrcheck	12
5.2	Targets	14
6	Checks	15
6.1	Running	15
6.2	Conditionals	17
6.2.1	This page	17
6.2.2	On page	18
6.2.3	Before / After	19
6.2.4	Pages	19
6.2.5	Close / Far	21
6.2.6	Chapter	22
6.2.7	Section	24

*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\zrcheck'~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 }

```

4.2 Options

hyperref option

```

\l_zrefcheck_use_hyperref_bool
\l_zrefcheck_warn_hyperref_bool

```

```

79 \bool_new:N \l__zrefcheck_use_hyperref_bool
80 \bool_new:N \l__zrefcheck_warn_hyperref_bool
81 \keys_define:nn { zref-check }
82 {
83   hyperref .choice: ,
84   hyperref / auto .code:n =
85   {
86     \bool_set_true:N \l__zrefcheck_use_hyperref_bool
87     \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
88   } ,
89   hyperref / true .code:n =
90   {
91     \bool_set_true:N \l__zrefcheck_use_hyperref_bool
92     \bool_set_true:N \l__zrefcheck_warn_hyperref_bool
93   } ,
94   hyperref / false .code:n =
95   {
96     \bool_set_false:N \l__zrefcheck_use_hyperref_bool
97     \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
98   } ,
99   hyperref .default:n = auto
100 }

```

(End definition for \l__zrefcheck_use_hyperref_bool and \l__zrefcheck_warn_hyperref_bool.)

```

101 \AtBeginDocument
102 {
103   \@ifpackageloaded { hyperref }
104   {
105     \bool_if:NT \l__zrefcheck_use_hyperref_bool
106     {
107       \RequirePackage { zref-hyperref }
108       \zref@addprop { zrefcheck } { anchor }
109     }
110   }
111   {
112     \bool_if:NT \l__zrefcheck_warn_hyperref_bool
113     { \msg_warning:nn { zref-check } { missing-hyperref } }
114     \bool_set_false:N \l__zrefcheck_use_hyperref_bool
115   }
116   \keys_define:nn { zref-check }
117   {
118     hyperref .code:n =
119     { \msg_warning:nn { zref-check } { hyperref-preamble-only } }
120   }
121 }

```

msglevel option

\l__zrefcheck_msglevel_tl

```

122 \tl_new:N \l__zrefcheck_msglevel_tl
123 \keys_define:nn { zref-check }
124 {
125   msglevel .choice: ,
126   msglevel / warn .code:n =
127   { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } } ,
128   msglevel / info .code:n =
129   { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } } ,
130   msglevel / none .code:n =
131   { \tl_set:Nn \l__zrefcheck_msglevel_tl { none } } ,
132   msglevel / obeydraft .code:n =
133   {
134     \ifdraft
135     { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
136     { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
137   } ,
138   msglevel / obeyfinal .code:n =
139   {
140     \ifoptionfinal
141     { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
142     { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
143   } ,
144   ignore: alias for msglevel=none
145   ignore .code:n =
146   { \msg_warning:nn { zref-check } { ignore-document-only } }
147 }

```

(End definition for \l__zrefcheck_msglevel_tl.)

```

147 \AtBeginDocument

```

```

148 {
149   \keys_define:nn { zref-check }
150   {
151     ignore .meta:n =
152       { msglevel = none }
153   }
154 }

```

onpage option

\l__zrefcheck_msgonpage_bool

```

155 \bool_new:N \l__zrefcheck_msgonpage_bool
156 \keys_define:nn { zref-check }
157 {
158   onpage .choice: ,
159   onpage / labelseq .code:n =
160     {
161       \bool_set_false:N \l__zrefcheck_msgonpage_bool
162     } ,
163   onpage / msg .code:n =
164     {
165       \bool_set_true:N \l__zrefcheck_msgonpage_bool
166     } ,
167   onpage / obeydraft .code:n =
168     {
169       \ifdraft
170         { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
171         { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
172       } ,
173   onpage / obeyfinal .code:n =
174     {
175       \ifoptionfinal
176         { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
177         { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
178       }
179 }

```

(End definition for \l__zrefcheck_msgonpage_bool.)

closerange option

\l__zrefcheck_close_range_int

```

180 \int_new:N \l__zrefcheck_close_range_int
181 \keys_define:nn { zref-check }
182 {
183   closerange .int_set:N = \l__zrefcheck_close_range_int ,
184 }

```

(End definition for \l__zrefcheck_close_range_int.)

Set load-time default values

```

185 \keys_set:nn { zref-check }
186 {
187   hyperref = auto ,
188   msglevel = warn ,
189   onpage = labelseq ,
190   closerange = 5
191 }

```

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

```
192 \RequirePackage { l3keys2e }  
193 \ProcessKeysOptions { zref-check }
```

`\zrchecksetup` Provide `\zrchecksetup`.

```
194 \NewDocumentCommand \zrchecksetup { m }  
195 { \keys_set:nn { zref-check } {#1} }
```

(End definition for `\zrchecksetup`. 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 `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`

```
196 \prop_new:N \g_zrefcheck_auxfile_lblseq_prop
```

(End definition for `\g_zrefcheck_auxfile_lblseq_prop`.)

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

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

```
199 {
```

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.

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

```
201 \group_begin:
```

```
202 \int_zero:N \l_tmpa_int
```

```
203 \tl_clear:N \l_tmpa_tl
```

```
204 \tl_clear:N \l_tmpb_tl
```

```
205 \bool_set_false:N \l_tmpa_bool
```

```
206 \ior_map_variable:NNn \g_tmpa_ior \l_tmpa_tl
```

```
207 {
```

```
208 \tl_map_variable:NNn \l_tmpa_tl \l_tmpb_tl
```

```
209 {
```

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

```
211 {
```

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

```
212 \bool_set_true:N \l_tmpa_bool
```

```
213 }
```

```
214 {
```

```
215 \bool_if:NTF \l_tmpa_bool
```



```

216         {
217             \bool_set_false:N \l_tmpa_bool
218             \int_incr:N \l_tmpa_int
219             \prop_gput:Nxx \g__zrefcheck_auxfile_lblseq_prop
220                 { \l_tmpb_tl } { \int_use:N \l_tmpa_int }
221         }
222     {

```

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.

```

223         \tl_map_break:
224     }
225 }
226 }
227 }
228 \group_end:
229 \ior_close:N \g_tmpa_ior
230 }

```

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-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.

```

231 \begingroup
232   \let \@addtoreset \ltx@gobbletwo
233   \newcounter { zrefcheck }
234 \endgroup
235 \setcounter { zrefcheck } { 0 }

```

4.5 Label formats

```

\__zrefcheck_check_lblfmt:n      \__zrefcheck_check_lblfmt:n {\check id int}}
236 \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}}
237 \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}}
238 \cs_new:Npn \zrefcheck_get_astl:nnn #1#2#3
239 {
240   \tl_clear:N #3
241   \tl_if_eq:nnTF {#2} { lblseq }
242   {
243     \prop_get:NnNF \g__zrefcheck_auxfile_lblseq_prop {#1} #3
244     {
245       \msg_warning:nnnn { zref-check }
246       { property-not-in-label } {#1} {#2}
247     }
248   }
249   {

```

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_zrcheck: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.

```

250     \zref@ifrefundefined {#1}
251     {}
252     {
253       \zref@ifpropundefined {#2}
254       { \msg_warning:nnnn { zref-check } { property-undefined } {#2} }
255       {
256         \zref@ifrefcontainsprop {#1} {#2}
257         {
258           \tl_set:Nx #3
259           { \zref@extractdefault {#1} {#2} { \c_empty_tl } }
260         }
261         {
262           \msg_warning:nnnn
263           { zref-check } { property-not-in-label } {#1} {#2}
264         }
265       }
266     }
267   }
268 }

```

(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).

```

269 \bool_new:N \l__zrefcheck_integer_bool

```

(End definition for \l__zrefcheck_integer_bool.)

\l__zrefcheck_propval_tl

```

270 \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>}
271 \cs_new:Npn \zrefcheck_get_asint:nnn #1#2#3
272 {
273   \zrefcheck_get_astl:nnn {#1} {#2} { \l__zrefcheck_propval_tl }
274   \__zrefcheck_is_integer:nTF { \l__zrefcheck_propval_tl }
275   {

```

Make it an integer data type.

```

276   \int_set:Nn #3 { \int_eval:n { \l__zrefcheck_propval_tl } }
277   }
278   {

```

```

279         \bool_set_false:N \l__zrefcheck_integer_bool
280         \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_zrcheck:nnnnn`.

```

281     { }
282     {
283         \msg_warning:nnnn { zref-check }
284         { property-not-integer } {#2} {#1}
285     }
286 }
287 }

```

(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.

```

288 \cs_new_eq:NN \__zrefcheck_int_to_roman:w \__int_to_roman:w
289 \prg_new_conditional:Npnn \__zrefcheck_is_integer:n #1 { p , T , F , TF }
290 {
291     \tl_if_empty:oTF {#1}
292     { \prg_return_false: }
293     {
294         \tl_if_empty:oTF { \__zrefcheck_int_to_roman:w -0#1 }
295         { \prg_return_true: }
296         { \prg_return_false: }
297     }
298 }

```

(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.

```

299 \prg_new_protected_conditional:Npnn \__zrefcheck_is_integer_rgx:n #1 { TF }
300 {
301     \regex_match:nnTF { \A\d+\Z } {#1}
302     { \prg_return_true: }
303     { \prg_return_false: }
304 }
305 \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 \zrcheck

`\zrcheck` The $\{\langle text \rangle\}$ argument of `\zrcheck` 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>.

`\zrcheck(*)[\langle options \rangle]{\langle labels \rangle}[\langle checks \rangle]{\langle text \rangle}`

```
306 \NewDocumentCommand \zrcheck
307   { s O { } > { \SplitList { , } } m > { \SplitList { , } } O { } m }
308   { \zref@wrapper@babel \__zrefcheck_zrcheck:nnnnn {#3} {#1} {#2} {#4} {#5} }
```

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

`\g__zrefcheck_id_int`

```
\l__zrefcheck_checkbeg_tl 309 \int_new:N \g__zrefcheck_id_int
\l__zrefcheck_checkend_tl 310 \tl_new:N \l__zrefcheck_checkbeg_tl
\l__zrefcheck_link_label_tl 311 \tl_new:N \l__zrefcheck_checkend_tl
\l__zrefcheck_link_anchor_tl 312 \tl_new:N \l__zrefcheck_link_label_tl
\l__zrefcheck_link_star_tl 313 \tl_new:N \l__zrefcheck_link_anchor_tl
314 \bool_new:N \l__zrefcheck_link_star_tl
```

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

`__zrefcheck_zrcheck:nnnnn` An intermediate internal function, which 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_zrcheck:nnnnn {\langle labels \rangle} {\langle * \rangle} {\langle options \rangle} {\langle checks \rangle} {\langle text \rangle}`

```
315 \cs_new:Npn \__zrefcheck_zrcheck:nnnnn #1#2#3#4#5
316   {
317     \group_begin:
```

Process local options.

```
318     \keys_set:nn { zrcheck } {#3}
```

Names of the labels for this `zrefcheck` call.

```
319     \int_gincr:N \g__zrefcheck_id_int
320     \tl_set:Nx \l__zrefcheck_checkbeg_tl
321       { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
322     \tl_set:Nx \l__zrefcheck_checkend_tl
323       { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
```

Set `checkbeg` label.

```
324     \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrcheck }
```

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.

```
325     \tl_set:Nn \l__zrefcheck_link_label_tl { \tl_head:n {#1} }
326     \bool_set:Nn \l__zrefcheck_link_star_tl {#2}
327     \zref@ifrefundefined { \l__zrefcheck_link_label_tl }
```

If the reference is undefined, just typeset.

```

328     {#5}
329     {
330         \bool_if:nTF
331         {
332             \l__zrefcheck_use_hyperref_bool &&
333             ! \l__zrefcheck_link_star_tl
334         }
335         {
336             \exp_args:Nx \zrefcheck_get_astl:nnn
337             { \l__zrefcheck_link_label_tl }
338             { anchor } { \l__zrefcheck_link_anchor_tl }
339             \hyperlink { \l__zrefcheck_link_anchor_tl } {#5}
340         }
341     } {#5}
342 }

```

Set checkend label.

```

343     \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.

```

344     \tl_map_function:nN {#1} \zref@refused

```

Run the checks.

```

345     \__zrefcheck_run_checks:nnV {#4} {#1} { \l__zrefcheck_checkbeg_tl }
346     \group_end:
347 }

```

(End definition for __zrefcheck_zrcheck:nnnnn.)

5.2 Targets

```

\zrctarget      \zrctarget{<label>}{<text>}

348 \NewDocumentCommand \zrctarget { m +m }
349 {
350     \refstepcounter { zrefcheck }
351     \zref@wrapper@babel \zref@labelbylist {#1} { zrefcheck }
352     #2
353     \zref@wrapper@babel
354     \zref@labelbylist { \__zrefcheck_end_lblfmt:n {#1} } { zrefcheck }
355 }

```

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

```

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

356 \NewDocumentEnvironment {zrcregion} { m }
357 {
358     \refstepcounter { zrefcheck }
359     \zref@wrapper@babel \zref@labelbylist {#1} { zrefcheck }
360 }
361 {

```

```

362 \zref@wrapper@babel
363 \zref@labelbylist { \__zrefcheck_end_lblfmt:n {#1} } { zrefcheck }
364 }

```

(End definition for `zrcregion`. 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 `\zrcheck`. 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
365 \cs_new:Npn \__zrefcheck_run_checks:nnn #1#2#3
366 {
367   \group_begin:
368     \tl_map_inline:nn {#2}
369     {
370       \tl_map_inline:nn {#1}
371       { \__zrefcheck_do_check:nnn {####1} {##1} {#3} }
372     }
373   \group_end:
374 }
375 \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
376 \bool_new:N \l_zrefcheck_passedcheck_bool
377 \bool_new:N \l__zrefcheck_onpage_bool
378 \seq_new:N \c__zrefcheck_onpage_checks_seq
379 \seq_set_from_clist:Nn \c__zrefcheck_onpage_checks_seq
380 { 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.

```
381 \cs_generate_variant:Nn \exp_args:Nnno { Nnoo }
```

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

```
382 \cs_new:Npn \__zrefcheck_do_check:nnn #1#2#3
```

```
383 {
```

```
384   \group_begin:
```

`<label beg>` may be defined or not, it is arbitrary user input. Whether this is the case is checked in `__zrefcheck_zrcheck: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.

```
385   \zref@ifrefundefined {#2}
```

```
386   {}
```

```
387   {
```

```
388     \bool_set_true:N \l__zrefcheck_passedcheck_bool
```

```
389     \bool_set_false:N \l__zrefcheck_onpage_bool
```

```
390     \cs_if_exist:cTF { __zrefcheck_check_ #1 :nnF }
```

```
391     {
```

“label beg” vs “reference beg”.

```
392       \use:c { __zrefcheck_check_ #1 :nnF }
```

```
393       {#2} {#3}
```

```
394       { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
```

“label beg” vs “reference end”.

```
395       \exp_args:Nnno \use:c { __zrefcheck_check_ #1 :nnF }
```

```
396       {#2} { \__zrefcheck_end_lblfmt:n {#3} }
```

```
397       { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
```

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

```
398       \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
```

```
399       {}
```

```
400       {
```

“label end” vs “reference beg”.

```
401         \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
```

```
402         { \__zrefcheck_end_lblfmt:n {#2} } {#3}
```

```
403         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
```

“label end” vs “reference end”.

```
404         \exp_args:Nnoo \use:c { __zrefcheck_check_ #1 :nnF }
```

```
405         { \__zrefcheck_end_lblfmt:n {#2} }
```

```
406         { \__zrefcheck_end_lblfmt:n {#3} }
```

```
407         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
```

```
408       }
```

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.

```
409       \seq_if_in:NnT \c__zrefcheck_onpage_checks_seq {#1}
```

```
410       {
```

```
411         \__zrefcheck_check_thispage:nnT
```



```

412         {#2} {#3}
413         { \bool_set_true:N \l__zrefcheck_onpage_bool }
414     \__zrefcheck_check_thispage:nnT
415     {#2} { \__zrefcheck_end_lblfmt:n {#3} }
416     { \bool_set_true:N \l__zrefcheck_onpage_bool }
417     \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
418     {}
419     {
420         \__zrefcheck_check_thispage:nnT
421         { \__zrefcheck_end_lblfmt:n {#2} } {#3}
422         { \bool_set_true:N \l__zrefcheck_onpage_bool }
423         \__zrefcheck_check_thispage:nnT
424         { \__zrefcheck_end_lblfmt:n {#2} }
425         { \__zrefcheck_end_lblfmt:n {#3} }
426         { \bool_set_true:N \l__zrefcheck_onpage_bool }
427     }
428 }
429 \bool_if:NTF \l__zrefcheck_passedcheck_bool
430 {
431     \bool_if:nT
432     {
433         \l__zrefcheck_msgonpage_bool &&
434         \l__zrefcheck_onpage_bool
435     }
436     {
437         \__zrefcheck_message:nnnx { double-check } {#1} {#2}
438         { \zref@extractdefault {#3} {page} {'unknown'} }
439     }
440 }
441 {
442     \__zrefcheck_message:nnnx { check-failed } {#1} {#2}
443     { \zref@extractdefault {#3} {page} {'unknown'} }
444 }
445 }
446 { \msg_warning:nnn { zref-check } { check-missing } {#1} }
447 }
448 \group_end:
449 }

```

(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
450 \int_new:N \l__zrefcheck_lbl_int
451 \int_new:N \l__zrefcheck_ref_int
452 \int_new:N \l__zrefcheck_lbl_b_int
453 \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

```

```

454 \prg_new_conditional:Npnn \__zrefcheck_check_thispage:nn #1#2 { T , F , TF }
455 {
456   \group_begin:
457     \bool_set_true:N \l__zrefcheck_integer_bool
458     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
459     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
460     \bool_lazy_and:nnTF
461       { \l__zrefcheck_integer_bool }
462       {
463         \int_compare_p:nNn
464           { \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.
465           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
466           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
467       }
468       { \group_insert_after:N \prg_return_true: }
469       { \group_insert_after:N \prg_return_false: }
470   \group_end:
471 }

```

(End definition for __zrefcheck_check_thispage:nn.)

6.2.2 On page

```

\__zrefcheck_check_above:nn
\__zrefcheck_check_below:nn
472 \prg_new_conditional:Npnn \__zrefcheck_check_above:nn #1#2 { F , TF }
473 {
474   \group_begin:
475     \__zrefcheck_check_thispage:nnTF {#1} {#2}
476     {
477       \bool_set_true:N \l__zrefcheck_integer_bool
478       \zrefcheck_get_asint:nnn {#1} { lblseq } { \l__zrefcheck_lbl_int }
479       \zrefcheck_get_asint:nnn {#2} { lblseq } { \l__zrefcheck_ref_int }
480       \bool_lazy_and:nnTF
481         { \l__zrefcheck_integer_bool }
482         {
483           \int_compare_p:nNn
484             { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
485             ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
486             ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
487         }
488         { \group_insert_after:N \prg_return_true: }
489         { \group_insert_after:N \prg_return_false: }
490     }
491     { \group_insert_after:N \prg_return_false: }
492   \group_end:
493 }
494 \prg_new_conditional:Npnn \__zrefcheck_check_below:nn #1#2 { F , TF }
495 {
496   \__zrefcheck_check_thispage:nnTF {#1} {#2}

```

```

497     {
498       \_zrefcheck_check_above:nnTF {#1} {#2}
499       { \prg_return_false: }
500       { \prg_return_true:  }
501     }
502     { \prg_return_false: }
503   }

```

(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
504 \prg_new_conditional:Npnn \_zrefcheck_check_before:nn #1#2 { F }
505 {
506   \_zrefcheck_check_pagesbefore:nnTF {#1} {#2}
507   { \prg_return_true: }
508   {
509     \_zrefcheck_check_above:nnTF {#1} {#2}
510     { \prg_return_true:  }
511     { \prg_return_false: }
512   }
513 }
514 \prg_new_conditional:Npnn \_zrefcheck_check_after:nn #1#2 { F }
515 {
516   \_zrefcheck_check_pagesafter:nnTF {#1} {#2}
517   { \prg_return_true: }
518   {
519     \_zrefcheck_check_below:nnTF {#1} {#2}
520     { \prg_return_true:  }
521     { \prg_return_false: }
522   }
523 }

```

(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
524 \prg_new_conditional:Npnn \_zrefcheck_check_nextpage:nn #1#2 { F }
525 {
526   \group_begin:
527   \bool_set_true:N \l_zrefcheck_integer_bool
528   \zrefcheck_get_asint:nnn {#1} { abspage } { \l_zrefcheck_lbl_int }
529   \zrefcheck_get_asint:nnn {#2} { abspage } { \l_zrefcheck_ref_int }
530   \bool_lazy_and:nnTF
531   { \l_zrefcheck_integer_bool }
532   {
533     \int_compare_p:nNn
534     { \l_zrefcheck_lbl_int } = { \l_zrefcheck_ref_int + 1 } &&
535     ! \int_compare_p:nNn { \l_zrefcheck_lbl_int } = { 0 } &&
536     ! \int_compare_p:nNn { \l_zrefcheck_ref_int } = { 0 }
537   }
538   { \group_insert_after:N \prg_return_true: }

```

```

539         { \group_insert_after:N \prg_return_false: }
540     \group_end:
541 }
542 \prg_new_conditional:Npnn \__zrefcheck_check_prevpage:nn #1#2 { F }
543 {
544     \group_begin:
545     \bool_set_true:N \l__zrefcheck_integer_bool
546     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
547     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
548     \bool_lazy_and:nnTF
549     { \l__zrefcheck_integer_bool }
550     {
551         \int_compare_p:nNn
552         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
553         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
554         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
555     }
556     { \group_insert_after:N \prg_return_true: }
557     { \group_insert_after:N \prg_return_false: }
558 \group_end:
559 }
560 \prg_new_conditional:Npnn \__zrefcheck_check_pagesbefore:nn #1#2 { F , TF }
561 {
562     \group_begin:
563     \bool_set_true:N \l__zrefcheck_integer_bool
564     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
565     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
566     \bool_lazy_and:nnTF
567     { \l__zrefcheck_integer_bool }
568     {
569         \int_compare_p:nNn
570         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
571         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
572         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
573     }
574     { \group_insert_after:N \prg_return_true: }
575     { \group_insert_after:N \prg_return_false: }
576 \group_end:
577 }
578 \cs_new_eq:NN \__zrefcheck_check_ppbefore:nnF \__zrefcheck_check_pagesbefore:nnF
579 \prg_new_conditional:Npnn \__zrefcheck_check_pagesafter:nn #1#2 { F , TF }
580 {
581     \group_begin:
582     \bool_set_true:N \l__zrefcheck_integer_bool
583     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
584     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
585     \bool_lazy_and:nnTF
586     { \l__zrefcheck_integer_bool }
587     {
588         \int_compare_p:nNn
589         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
590         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
591         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
592     }

```

```

593     { \group_insert_after:N \prg_return_true: }
594     { \group_insert_after:N \prg_return_false: }
595   \group_end:
596 }
597 \cs_new_eq:NN \__zrefcheck_check_ppafter:nnF \__zrefcheck_check_pagesafter:nnF
598 \prg_new_conditional:Npnn \__zrefcheck_check_facing: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       {

```

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

```

607     \legacy_if_p:n { @twoside } &&

```

Now we test “facing”.

```

608   (
609     (
610       \int_if_odd_p:n { \l__zrefcheck_ref_int } &&
611       \int_compare_p:nNn
612         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 }
613     ) ||
614     (
615       \int_if_even_p:n { \l__zrefcheck_ref_int } &&
616       \int_compare_p:nNn
617         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 }
618     )
619   ) &&
620   ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
621   ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
622 }
623 { \group_insert_after:N \prg_return_true: }
624 { \group_insert_after:N \prg_return_false: }
625 \group_end:
626 }

```

(End definition for __zrefcheck_check_nextpage:nn and others.)

6.2.5 Close / Far

```

\__zrefcheck_check_close:nn
\__zrefcheck_check_far:nn
627 \prg_new_conditional:Npnn \__zrefcheck_check_close:nn #1#2 { F , TF }
628 {
629   \group_begin:
630     \bool_set_true:N \l__zrefcheck_integer_bool
631     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
632     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
633     \bool_lazy_and:nnTF
634       { \l__zrefcheck_integer_bool }
635       {
636         \int_compare_p:nNn
637           { \int_abs:n { \l__zrefcheck_lbl_int - \l__zrefcheck_ref_int } }

```

```

638         <
639         { \l__zrefcheck_close_range_int + 1 } &&
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 \prg_new_conditional:Npnn \__zrefcheck_check_far:nn #1#2 { F }
648 {
649     \__zrefcheck_check_close:nnTF {#1} {#2}
650     { \prg_return_false: }
651     { \prg_return_true: }
652 }

```

(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
653 \prg_new_conditional:Npnn \__zrefcheck_check_thischap:nn #1#2 { F }
654 {
655     \group_begin:
656     \bool_set_true:N \l__zrefcheck_integer_bool
657     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
658     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
659     \bool_lazy_and:nnTF
660     { \l__zrefcheck_integer_bool }
661     {
662         \int_compare_p:nNn
663         { \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.

```

664         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
665         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
666     }
667     { \group_insert_after:N \prg_return_true: }
668     { \group_insert_after:N \prg_return_false: }
669 \group_end:
670 }
671 \prg_new_conditional:Npnn \__zrefcheck_check_nextchap:nn #1#2 { F }
672 {
673     \group_begin:
674     \bool_set_true:N \l__zrefcheck_integer_bool
675     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
676     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
677     \bool_lazy_and:nnTF
678     { \l__zrefcheck_integer_bool }
679     {

```

```

680         \int_compare_p:nNn
681         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
682         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
683     }
684     { \group_insert_after:N \prg_return_true: }
685     { \group_insert_after:N \prg_return_false: }
686 \group_end:
687 }
688 \prg_new_conditional:Npnn \__zrefcheck_check_prevchap:nn #1#2 { F }
689 {
690     \group_begin:
691     \bool_set_true:N \l__zrefcheck_integer_bool
692     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
693     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
694     \bool_lazy_and:nnTF
695     { \l__zrefcheck_integer_bool }
696     {
697         \int_compare_p:nNn
698         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
699         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
700         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
701     }
702     { \group_insert_after:N \prg_return_true: }
703     { \group_insert_after:N \prg_return_false: }
704 \group_end:
705 }
706 \prg_new_conditional:Npnn \__zrefcheck_check_chapsafter:nn #1#2 { F }
707 {
708     \group_begin:
709     \bool_set_true:N \l__zrefcheck_integer_bool
710     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
711     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
712     \bool_lazy_and:nnTF
713     { \l__zrefcheck_integer_bool }
714     {
715         \int_compare_p:nNn
716         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
717         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
718     }
719     { \group_insert_after:N \prg_return_true: }
720     { \group_insert_after:N \prg_return_false: }
721 \group_end:
722 }
723 \prg_new_conditional:Npnn \__zrefcheck_check_chapsbefore:nn #1#2 { F }
724 {
725     \group_begin:
726     \bool_set_true:N \l__zrefcheck_integer_bool
727     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
728     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
729     \bool_lazy_and:nnTF
730     { \l__zrefcheck_integer_bool }
731     {
732         \int_compare_p:nNn
733         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&

```

```

734         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
735         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
736     }
737     { \group_insert_after:N \prg_return_true: }
738     { \group_insert_after:N \prg_return_false: }
739 \group_end:
740 }

```

(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_secsafter:nn
\__zrefcheck_check_secsbefore:nn
741 \prg_new_conditional:Npnn \__zrefcheck_check_thissec:nn #1#2 { F }
742 {
743     \group_begin:
744     \bool_set_true:N \l__zrefcheck_integer_bool
745     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
746     \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
747     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
748     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
749     \bool_lazy_and:nnTF
750     { \l__zrefcheck_integer_bool }
751     {
752         \int_compare_p:nNn
753         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
754         \int_compare_p:nNn
755         { \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.

```

756         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
757         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
758     }
759     { \group_insert_after:N \prg_return_true: }
760     { \group_insert_after:N \prg_return_false: }
761 \group_end:
762 }
763 \prg_new_conditional:Npnn \__zrefcheck_check_nextsec:nn #1#2 { F }
764 {
765     \group_begin:
766     \bool_set_true:N \l__zrefcheck_integer_bool
767     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
768     \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
769     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
770     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
771     \bool_lazy_and:nnTF
772     { \l__zrefcheck_integer_bool }
773     {
774         \int_compare_p:nNn
775         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&

```



```

776         \int_compare_p:nNn
777         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
778         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
779     }
780     { \group_insert_after:N \prg_return_true: }
781     { \group_insert_after:N \prg_return_false: }
782 \group_end:
783 }
784 \prg_new_conditional:Npnn \__zrefcheck_check_prevsec:nn #1#2 { F }
785 {
786     \group_begin:
787     \bool_set_true:N \l__zrefcheck_integer_bool
788     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
789     \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
790     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
791     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
792     \bool_lazy_and:nnTF
793     { \l__zrefcheck_integer_bool }
794     {
795         \int_compare_p:nNn
796         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
797         \int_compare_p:nNn
798         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
799         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
800         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
801     }
802     { \group_insert_after:N \prg_return_true: }
803     { \group_insert_after:N \prg_return_false: }
804 \group_end:
805 }
806 \prg_new_conditional:Npnn \__zrefcheck_check_secsafter:nn #1#2 { F }
807 {
808     \group_begin:
809     \bool_set_true:N \l__zrefcheck_integer_bool
810     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
811     \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
812     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
813     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
814     \bool_lazy_and:nnTF
815     { \l__zrefcheck_integer_bool }
816     {
817         \int_compare_p:nNn
818         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
819         \int_compare_p:nNn
820         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
821         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
822     }
823     { \group_insert_after:N \prg_return_true: }
824     { \group_insert_after:N \prg_return_false: }
825 \group_end:
826 }
827 \prg_new_conditional:Npnn \__zrefcheck_check_secsbefore:nn #1#2 { F }
828 {
829     \group_begin:

```

```

830 \bool_set_true:N \l__zrefcheck_integer_bool
831 \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
832 \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
833 \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
834 \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
835 \bool_lazy_and:nnTF
836 { \l__zrefcheck_integer_bool }
837 {
838   \int_compare_p:nNn
839   { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
840   \int_compare_p:nNn
841   { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
842   ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
843   ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
844 }
845 { \group_insert_after:N \prg_return_true: }
846 { \group_insert_after:N \prg_return_false: }
847 \group_end:
848 }

(End definition for \__zrefcheck_check_thissec:nn and others.)

849 </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		545, 563, 582, 601, 630, 656, 674, 691, 709, 726, 744, 766, 787, 809, 830	
\A	301	\l_tmpa_bool 8, 205, 212, 215, 217	
\AddToHook	21, 28		
\AtBeginDocument	101, 147		
B		C	
\begingroup	231	\catcode	8
bool commands:		\chapter	2, 22
		cs commands:	
\bool_if:NTF	105, 112, 215, 429	\cs_generate_variant:Nn .	45, 375, 381
\bool_if:nTF	330, 431	\cs_if_exist:NTF	390
\bool_lazy_and:nnTF		\cs_new:Npn	
.....	11, 460, 480, 530, 548, 566, 585, 604, 633, 659, 677, 694, 712, 729, 749, 771, 792, 814, 835	. 40, 236, 237, 238, 271, 315, 365, 382	
\bool_new:N		\cs_new_eq:NN	288, 578, 597
.....	79, 80, 155, 269, 314, 376, 377	D	
\bool_set:Nn	326	\d	301
\bool_set_false:N		E	
... 87, 96, 97, 114, 161, 170, 177, 205, 217, 279, 389, 394, 397, 403, 407		\endgroup	234
\bool_set_true:N	86, 91, 92, 165, 171, 176, 212, 388, 413, 416, 422, 426, 457, 477, 527,	\endinput	12
		exp commands:	
		\exp_args:Nnno	381, 395
		\exp_args:Nno	401
		\exp_args:Nnoo	404

\exp_args:Nx	336	ior commands:	
F		\ior_close:N	229
file commands:		\ior_map_variable:NNn	206
\file_if_exist:nTF	198	\ior_open:Nn	200
\fmtversion	3	\g_tmpa_ior	200, 206, 229
G		iow commands:	
group commands:		\iow_newline: ...	48, 53, 64, 69, 73, 76
\group_begin:		K	
. 201, 317, 367, 384, 456, 474, 526,		keys commands:	
544, 562, 581, 600, 629, 655, 673,		\keys_define:nn	
690, 708, 725, 743, 765, 786, 808, 829		81, 116, 123, 149, 156, 181
\group_end:		\keys_set:nn	185, 195, 318
. 228, 346, 373, 448, 470, 492, 540,		L	
558, 576, 595, 625, 645, 669, 686,		legacy commands:	
704, 721, 739, 761, 782, 804, 825, 847		\legacy_if_p:n	607
\group_insert_after:N	468,	\let	232
469, 488, 489, 491, 538, 539, 556,		M	
557, 574, 575, 593, 594, 623, 624,		\MessageBreak	10
643, 644, 667, 668, 684, 685, 702,		msg commands:	
703, 719, 720, 737, 738, 759, 760,		\msg_line_number:	49, 54, 57, 59, 61, 65
780, 781, 802, 803, 823, 824, 845, 846		\msg_new:nnn	
H		46, 51, 56, 58, 60, 62, 67, 72, 74
\hyperlink	12, 339	\msg_warning:nn	113, 119, 145
I		\msg_warning:nnn	446
\ifdraft	134, 169	\msg_warning:nnnn	245, 254, 262, 283
\IfFormatAtLeastTF	3, 4	N	
\ifoptionfinal	140, 175	\newcounter	233
int commands:		\NewDocumentCommand	194, 306, 348
\int_abs:n	637	\NewDocumentEnvironment	356
\int_compare_p:nNn		\noexpand	70
.....	463, 465, 466, 483, 485, 486,	P	
533, 535, 536, 551, 553, 554, 569,		\PackageError	7
571, 572, 588, 590, 591, 611, 616,		prg commands:	
620, 621, 636, 640, 641, 662, 664,		\prg_generate_conditional_-	
665, 680, 682, 697, 699, 700, 715,		variant:Nnn	305
717, 732, 734, 735, 752, 754, 756,		\prg_new_conditional:Npnn	15, 289,
757, 774, 776, 778, 795, 797, 799,		454, 472, 494, 504, 514, 524, 542,	
800, 817, 819, 821, 838, 840, 842, 843		560, 579, 598, 627, 647, 653, 671,	
\int_eval:n	276	688, 706, 723, 741, 763, 784, 806, 827	
\int_gincr:N	23, 29, 319	\prg_new_protected_conditional:Npnn	
\int_if_even_p:n	615	299
\int_if_odd_p:n	610	\prg_return_false:	
\int_incr:N	218	15, 292, 296, 303, 469, 489,
\int_new:N		491, 499, 502, 511, 521, 539, 557,	
.. 19, 20, 180, 309, 450, 451, 452, 453		575, 594, 624, 644, 650, 668, 685,	
\int_set:Nn	276	703, 720, 738, 760, 781, 803, 824, 846	
\int_use:N	26, 30, 220, 236	\prg_return_true:	
\int_zero:N	24, 202	15, 295, 302, 468, 488,
\l_tmpa_int	202, 218, 220	500, 507, 510, 517, 520, 538, 556,	
int internal commands:			
__int_to_roman:w	12, 288		

_zrefcheck_check_before:nn	..	504	695, 709, 713, 726, 730, 744, 750,
_zrefcheck_check_below:nn	...	472	766, 772, 787, 793, 809, 815, 830, 836
_zrefcheck_check_below:nnTF	..	519	_zrefcheck_is_integer:n ... 12 , 288
_zrefcheck_check_chapsafter:nn		653	_zrefcheck_is_integer:nTF ... 274
_zrefcheck_check_chapsbefore:nn		653	_zrefcheck_is_integer_rgx:n 12 , 299
	653	\l_zrefcheck_lbl_b_int
_zrefcheck_check_close:nn	...	627 450 , 747 , 753 ,
_zrefcheck_check_close:nnTF	..	649	769 , 775 , 790 , 796 , 812 , 818 , 833 , 839
_zrefcheck_check_facing:nn	..	524	\l_zrefcheck_lbl_int
_zrefcheck_check_far:nn	627 450 , 458 , 464 , 465 , 478 , 484 ,
_zrefcheck_check_lblfmt:n	9 , 236 , 321	485 , 528 , 534 , 535 , 546 , 552 , 553 ,
_zrefcheck_check_nextchap:nn	..	653	564 , 570 , 571 , 583 , 589 , 590 , 602 ,
_zrefcheck_check_nextpage:nn	..	524	612 , 617 , 620 , 631 , 637 , 640 , 657 ,
_zrefcheck_check_nextsec:nn	..	741	663 , 664 , 675 , 681 , 682 , 692 , 698 ,
_zrefcheck_check_pagesafter:nn		524	699 , 710 , 716 , 717 , 727 , 733 , 734 ,
_zrefcheck_check_pagesafter:nnTF		516 , 597	745 , 755 , 756 , 767 , 777 , 778 , 788 ,
_zrefcheck_check_pagesbefore:nn		524	798 , 799 , 810 , 820 , 821 , 831 , 841 , 842
_zrefcheck_check_pagesbefore:nnTF		506 , 578	\l_zrefcheck_link_anchor_tl ...
_zrefcheck_check_ppafter:nn	..	524 309 , 338 , 339
_zrefcheck_check_ppafter:nnTF		597	\l_zrefcheck_link_label_tl
_zrefcheck_check_ppbefore:nn	..	524 309 , 325 , 327 , 337
_zrefcheck_check_ppbefore:nnTF		578	\l_zrefcheck_link_star_tl
_zrefcheck_check_prevchap:nn	..	653 309 , 326 , 333
_zrefcheck_check_prevpage:nn	..	524	_zrefcheck_message:nnnn 40 , 437 , 442
_zrefcheck_check_prevsec:nn	..	741	\l_zrefcheck_msglevel_tl ... 42 , 122
_zrefcheck_check_secsafter:nn		741	\l_zrefcheck_msgonpage_bool 155 , 433
_zrefcheck_check_secsbefore:nn		741	\l_zrefcheck_onpage_bool
_zrefcheck_check_thischap:nn	..	653 376 , 389 , 413 , 416 , 422 , 426 , 434
_zrefcheck_check_thispage:nn	..	454	\c_zrefcheck_onpage_checks_seq ..
_zrefcheck_check_thispage:nnTF		411 , 414 , 420 , 423 , 475 , 496 376 , 409
_zrefcheck_check_thissec:nn	..	741	\l_zrefcheck_passedcheck_bool ..
\l_zrefcheck_checkbeg_tl	309 , 320 , 323 , 324 , 345 376 , 388 , 394 , 397 , 403 , 407 , 429
\l_zrefcheck_checkend_tl	309 , 322 , 343	\l_zrefcheck_propval_tl
\l_zrefcheck_close_range_int	...	180 , 639 270 , 273 , 274 , 276
_zrefcheck_do_check:nnn	16 , 371 , 382		\l_zrefcheck_ref_b_int
_zrefcheck_end_lblfmt:n	9 , 237 , 323 , 354 , 363 , 396 , 398 , 450 , 748 , 753 ,
		402 , 405 , 406 , 415 , 417 , 421 , 424 , 425	770 , 775 , 791 , 796 , 813 , 818 , 834 , 839
_zrefcheck_get_asint:nnn	10	\l_zrefcheck_ref_int
_zrefcheck_get_astl:nnn	10 450 , 459 , 464 , 466 ,
\g_zrefcheck_int_to_roman:w	...	309 , 319 , 321	479 , 484 , 486 , 529 , 534 , 536 , 547 ,
\l_zrefcheck_integer_bool	11 , 269 , 279 ,	552 , 554 , 565 , 570 , 572 , 584 , 589 ,
		457 , 461 , 477 , 481 , 527 , 531 , 545 ,	591 , 603 , 610 , 612 , 615 , 617 , 621 ,
		549 , 563 , 567 , 582 , 586 , 601 , 605 ,	632 , 637 , 641 , 658 , 663 , 665 , 676 ,
		630 , 634 , 656 , 660 , 674 , 678 , 691 ,	681 , 693 , 698 , 700 , 711 , 716 , 728 ,
			733 , 735 , 746 , 755 , 757 , 768 , 777 ,
			789 , 798 , 800 , 811 , 820 , 832 , 841 , 843
			_zrefcheck_run_checks:nnn
		 15 , 345 , 365
			\l_zrefcheck_use_hyperref_bool ..
		 79 , 105 , 114 , 332
			\l_zrefcheck_warn_hyperref_bool
		 79 , 112
			_zrefcheck_zrcheck:nnnnn
		 10 , 11 , 13 , 16 , 308 , 315