

The `zref-check` package implementation*

Gustavo Barros[†]

2021-09-16

Contents

1	Initial setup	2
2	Dependencies	2
3	<code>zref</code> setup	2
4	Plumbing	3
4.1	Messages	3
4.2	Integer testing	4
4.3	Options	5
4.4	Position on page	9
4.5	Counter	11
4.6	Label formats	12
4.7	Property values	12
5	User interface	14
5.1	<code>\zcheck</code>	14
5.2	Targets	16
6	Checks	16
6.1	Single label checks	17
6.2	Setup	17
6.3	Running	18
6.4	Conditionals	21
6.4.1	This page	21
6.4.2	On page	22
6.4.3	Before / After	22
6.4.4	Pages	23
6.4.5	Close / Far	25
6.4.6	Chapter	26
6.4.7	Section	28
7	<code>zref-clever</code> integration	30

*This file describes v0.2.1, last revised 2021-09-16.

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

1 Initial setup

Start the DocStrip guards.

```
1 <*package>
   Identify the internal prefix (LATEX3 DocStrip convention).
```

```
2 <@@=zrefcheck>
```

For the `chapter` and `section` checks, `zref-check` uses the new hook system in `ltxcmds`, which was released with the 2021/06/01 L^AT_EX kernel.

```
3 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
4 \IfFormatAtLeastTF{2021-06-01}
5 {}
6 {%
7   \PackageError{zref-check}{LaTeX kernel too old}
8   {%
9     'zref-check' requires a LaTeX kernel newer than 2021-06-01.%
10    \MessageBreak Loading will abort!%
11   }%
12 \endinput
13 }%
   Identify the package.
14 \ProvidesExplPackage {zref-check} {2021-09-16} {0.2.1}
15 {Flexible cross-references with contextual checks based on zref}
```

2 Dependencies

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

3 zref setup

`\g__zrefcheck_abschap_int` `\g__zrefcheck_abssec_int` Provide absolute counters for section and chapter, and respective `zref` properties, so that we can make checks about relation of chapters/sections regardless of internal counters, since we don't get those for the unnumbered (starred) ones. 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 { zc@abschap } [0] { \int_use:N \g__zrefcheck_abschap_int }
27 \zref@addprop \ZREF@mainlist { zc@abschap }

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

```

These are the lists of properties to be used by zref-check, that is, the list of properties the references and targets store. This is the minimum set required, more properties may be added according to options.

```

32 \zref@newlist { zrefcheck-check }
33 \zref@addprops { zrefcheck-check }
34 {
35   page , % for messages
36   abspage ,
37   zc@abschap ,
38   zc@abssec
39 }
40 \zref@newlist { zrefcheck-target }
41 \zref@addprops { zrefcheck-target }
42 {
43   page , % so that \zpageref can refer to it
44   abspage ,
45   zc@abschap ,
46   zc@abssec
47 }
48 \zref@newlist { zrefcheck-end }
49 \zref@addprops { zrefcheck-end }
50 {
51   abspage ,
52   zc@abschap ,
53   zc@abssec
54 }

```

4 Plumbing

4.1 Messages

__zrefcheck_message:nnnn

__zrefcheck_message:nnnx

```

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

```

(End definition for __zrefcheck_message:nnnn.)

```

61 \msg_new:nnn { zref-check } { check-failed }
62 { Failed~check~'~#1'~for~label~'~#2'~on~page~#3~\msg_line_context:. }
63 \msg_new:nnn { zref-check } { double-check }
64 { Double-check~'~#1'~for~label~'~#2'~on~page~#3~\msg_line_context:. }

```

```

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

```

4.2 Integer testing

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

```

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

```

```

102     {
103       \tl_if_empty:OTF { \__zrefcheck_int_to_roman:w -0#1 }
104       { \prg_return_true: }
105       { \prg_return_false: }
106     }
107   }

```

(End definition for __zrefcheck_is_integer:n and __zrefcheck_int_to_roman:w.)

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

```

108 \prg_new_protected_conditional:Npnn \__zrefcheck_is_integer_rgx:n #1 { TF }
109 {
110   \regex_match:nnTF { \A\d+\Z } {#1}
111   { \prg_return_true: }
112   { \prg_return_false: }
113 }

```

(End definition for __zrefcheck_is_integer_rgx:n.)

4.3 Options

hyperref option

```

\l__zrefcheck_use_hyperref_bool
\l__zrefcheck_warn_hyperref_bool
114 \bool_new:N \l__zrefcheck_use_hyperref_bool
115 \bool_new:N \l__zrefcheck_warn_hyperref_bool
116 \keys_define:nn { zref-check }
117 {
118   hyperref .choice: ,
119   hyperref / auto .code:n =
120   {
121     \bool_set_true:N \l__zrefcheck_use_hyperref_bool
122     \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
123   } ,
124   hyperref / true .code:n =
125   {
126     \bool_set_true:N \l__zrefcheck_use_hyperref_bool
127     \bool_set_true:N \l__zrefcheck_warn_hyperref_bool
128   } ,
129   hyperref / false .code:n =
130   {
131     \bool_set_false:N \l__zrefcheck_use_hyperref_bool
132     \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
133   } ,
134   hyperref .initial:n = auto ,
135   hyperref .default:n = auto
136 }

```

(End definition for \l__zrefcheck_use_hyperref_bool and \l__zrefcheck_warn_hyperref_bool.)

```

137 \AddToHook { begindocument }

```

```

138 {
139   \@ifpackageloaded { hyperref }
140   {
141     \bool_if:NT \l__zrefcheck_use_hyperref_bool
142     {
143       \RequirePackage { zref-hyperref }
144       \zref@addprop { zrefcheck-target } { anchor }
145     }
146   }
147   {
148     \bool_if:NT \l__zrefcheck_warn_hyperref_bool
149     { \msg_warning:nn { zref-check } { missing-hyperref } }
150     \bool_set_false:N \l__zrefcheck_use_hyperref_bool
151   }
152   \keys_define:nn { zref-check }
153   {
154     hyperref .code:n =
155     { \msg_warning:nn { zref-check } { hyperref-preamble-only } }
156   }
157 }

```

msglevel option

`\l__zrefcheck_msglevel_tl`

```

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

```

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

```

182   ignore .code:n =
183   { \msg_warning:nn { zref-check } { ignore-document-only } } ,
184   ignore .value_forbidden:n = true
185 }

```

(End definition for \l__zrefcheck_msglevel_tl.)

```
186 \AddToHook { begindocument }
187 {
188   \keys_define:nn { zref-check }
189     { ignore .meta:n = { msglevel = none } }
190 }
```

onpage option

\l__zrefcheck_msgonpage_bool

```
191 \bool_new:N \l__zrefcheck_msgonpage_bool
192 \keys_define:nn { zref-check }
193 {
194   onpage .choice: ,
195   onpage / labelseq .code:n =
196     {
197       \bool_set_false:N \l__zrefcheck_msgonpage_bool
198     } ,
199   onpage / msg .code:n =
200     {
201       \bool_set_true:N \l__zrefcheck_msgonpage_bool
202     } ,
203   onpage / obeydraft .code:n =
204     {
205       \ifdraft
206         { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
207         { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
208       } ,
209   onpage / obeyfinal .code:n =
210     {
211       \ifoptionfinal
212         { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
213         { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
214       } ,
215   onpage .value_required:n = true ,
216   onpage .initial:n = labelseq
217 }
```

(End definition for \l__zrefcheck_msgonpage_bool.)

closerange option

\l__zrefcheck_close_range_int

```
218 \int_new:N \l__zrefcheck_close_range_int
219 \keys_define:nn { zref-check }
220 {
221   closerange .code:n =
222     {
223       \__zrefcheck_is_integer_rgx:nTF {#1}
224       { \int_set:Nn \l__zrefcheck_close_range_int { \int_eval:n {#1} } }
225       {
226         \msg_warning:nn { zref-check } { closerange-not-positive-integer }
227         \int_set:Nn \l__zrefcheck_close_range_int { 5 }
228       }
229     }
```

```

228     }
229   } ,
230   closerange .value_required:n = true ,
231   closerange .initial:n = 5
232 }

```

(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>.

```

233 \tl_new:N \l__zrefcheck_target_label_tl
234 \bool_new:N \l__zrefcheck_target_label_bool
235 \keys_define:nn { zref-check }
236 {
237   labelcmd .code:n =
238   {
239     \tl_set:Nn \l__zrefcheck_target_label_tl {#1}
240     \bool_set_true:N \l__zrefcheck_target_label_bool
241   } ,
242   labelcmd .value_required:n = true ,
243 }

```

(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.

```

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

```

(End definition for __zrefcheck_target_label:n.)

```

246 \AddToHook { begindocument }
247 {
248   \bool_if:NT \l__zrefcheck_target_label_bool
249   {
250     \tl_if_blank:VT \l__zrefcheck_target_label_tl
251     { \tl_clear:N \l__zrefcheck_target_label_tl }
252     \cs_if_exist:CTF { \l__zrefcheck_target_label_tl }
253     {
254       \cs_set_protected:Npx \__zrefcheck_target_label:n #1
255       {
256         \exp_not:o
257         { \cs:w \l__zrefcheck_target_label_tl \cs_end: }
258         {#1}
259       }
260     }
261     {
262       \exp_args:Nnno \msg_warning:nnn { zref-check }
263       { labelcmd-undefined } { \l__zrefcheck_target_label_tl }
264     }
265   }
266   \keys_define:nn { zref-check }
267   {

```



```

268     labelcmd .code:n =
269     {
270         \msg_warning:nnn { zref-check }
271         { option-preamble-only } { labelcmd }
272     }
273 }
274 }

```

Package options

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

```

275 \RequirePackage { l3keys2e }
276 \ProcessKeysOptions { zref-check }

```

`\zrefchecksetup` Provide `\zrefchecksetup`.

```

277 \NewDocumentCommand \zrefchecksetup { m }
278 { \keys_set:nn { zref-check } {#1} }

```

(End definition for \zrefchecksetup.)

4.4 Position on page

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

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

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

Patching this macro for this is not possible. First, `\zref@newlabel` is one of those “commands that look ahead” mentioned in `ltxcmdhooks` documentation. Indeed, `\@newl@bel` receives 3 arguments, and `\zref@newlabel` just passes the first, the following two will be scanned ahead. Second, the `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` and how the strings work is a caveat (see https://tex.stackexchange.com/q/446123#comment1516961_446381, thanks David Carlisle). Still... let’s stick to tradition as long as it works, `zref` already does a great job in this regard anyway.

`\g_zrefcheck_auxfile_lblseq_prop`

```
279 \prop_new:N \g_zrefcheck_auxfile_lblseq_prop
```

(End definition for `\g_zrefcheck_auxfile_lblseq_prop`.)

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

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

```
282 {
```

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.

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

```
284 \group_begin:
```

```
285 \int_zero:N \l_tmpa_int
```

```
286 \tl_clear:N \l_tmpa_tl
```

```
287 \tl_clear:N \l_tmpb_tl
```

```
288 \bool_set_false:N \l_tmpa_bool
```

```
289 \ior_map_variable:NNn \g_tmpa_ior \l_tmpa_tl
```

```
290 {
```

```

291         \tl_map_variable:NNn \l_tmpa_tl \l_tmpb_tl
292         {
293             \tl_if_eq:NnTF \l_tmpb_tl { \zref@newlabel }
294             {

```

Found a \zref@label, signal it.

```

295             \bool_set_true:N \l_tmpa_bool
296         }
297         {
298             \bool_if:NNTF \l_tmpa_bool
299             {
300                 \bool_set_false:N \l_tmpa_bool
301                 \int_incr:N \l_tmpa_int
302                 \prop_gput:Nxx \g__zrefcheck_auxfile_lblseq_prop
303                 { \l_tmpb_tl } { \int_use:N \l_tmpa_int }
304             }
305             {

```

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.

```

306             \tl_map_break:
307         }
308     }
309 }
310 }
311 \group_end:
312 \ior_close:N \g_tmpa_ior
313 }

```

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

4.5 Counter

We need a dedicated counter for the labels generated by the checks and targets. The value of the counter is not relevant, we just need it to be able to set proper anchors with \refstepcounter. And, since I couldn’t find a \refstepcounter equivalent in L3, we use a standard 2e counter here. I’m also using the technique to ensure the counter is

never reset that is used by `zref-abspace.sty` and `\zref@require@unique`. Indeed, the requirements are the same, we need numbers ensured to be *unique* in the counter.

```

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

```

4.6 Label formats

```

\__zrefcheck_check_lblfmt:n \__zrefcheck_check_lblfmt:n {<check id int>}
319 \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>}
320 \cs_new:Npn \__zrefcheck_end_lblfmt:n #1 { #1 @zrefcheck }
(End definition for \__zrefcheck_end_lblfmt:n.)

```

4.7 Property values

`\zrefcheck_get_astl:nnn` A convenience function to retrieve property values from labels. Uses `\g__zrefcheck_auxfile_lblseq_prop` for `lblseq`, and calls `\zref@extractdefault` for everything else.

We cannot use the “return value” of `__zrefcheck_get_astl:nnn` or `__zrefcheck_get_asint:nnn` directly, because we need to use the retrieved property values as arguments in the checks, however we use here a number of non-expandable operations. Hence, we receive a local `tl/int` variable as third argument and set that, so that it is available (and expandable) at the place of use, and also make these functions ‘protected’ (see egreg’s <https://tex.stackexchange.com/a/572903>: “a function that performs assignments should be protected”). For this reason, we do not group here, because we are passing a local variable around, but it is expected this function will be called within a group.

We’re returning `\c_empty_tl` in case of failure to find the intended property value (explicitly in `\zref@extractdefault`, but that is also what `\tl_clear:N` does).

```

\zrefcheck_get_astl:nnn {<label>} {<prop>} {<tl var>}

321 \cs_new_protected:Npn \zrefcheck_get_astl:nnn #1#2#3
322 {
323   \tl_clear:N #3
324   \tl_if_eq:nnTF {#2} { lblseq }
325   {
326     \prop_get:NnNF \g__zrefcheck_auxfile_lblseq_prop {#1} #3
327     {
328       \msg_warning:nnnn { zref-check }
329       { property-not-in-label } {#1} {#2}
330     }
331   }
332   {

```

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.

```

333     \zref@ifrefundefined {#1}
334     {}
335     {
336         \zref@ifpropundefined {#2}
337         { \msg_warning:nnnn { zref-check } { property-undefined } {#2} }
338         {
339             \zref@ifrefcontainsprop {#1} {#2}
340             {
341                 \tl_set:Nx #3
342                 { \zref@extractdefault {#1} {#2} { \c_empty_tl } }
343             }
344             {
345                 \msg_warning:nnnn
346                 { zref-check } { property-not-in-label } {#1} {#2}
347             }
348         }
349     }
350 }
351 }

```

(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).

```

352 \bool_new:N \l__zrefcheck_integer_bool

```

(End definition for `\l__zrefcheck_integer_bool`.)

`\l__zrefcheck_propval_tl`

```

353 \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}
354 \cs_new_protected:Npn \zrefcheck_get_asint:nnn #1#2#3
355 {
356   \zrefcheck_get_astl:nnn {#1} {#2} { \l__zrefcheck_propval_tl }
357   \__zrefcheck_is_integer:nTF { \l__zrefcheck_propval_tl }
358   {

```

Make it an integer data type.

```

359       \int_set:Nn #3 { \int_eval:n { \l__zrefcheck_propval_tl } }
360   }
361   {
362     \bool_set_false:N \l__zrefcheck_integer_bool
363     \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:nnnn`.

```

364       { }
365       {
366         \msg_warning:nnnn { zref-check }
367         { property-not-integer } {#2} {#1}
368       }
369     }
370   }

```

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

5 User interface

5.1 `\zcheck`

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

```

\zcheck{*}[\checks/options]{\labels}{\text}

```

```

371 \NewDocumentCommand \zcheck { s O { } m m }
372 { \zref@wrapper@babel \__zrefcheck_zcheck:nnnn {#3} {#1} {#2} {#4} }

```

(End definition for `\zcheck`.)

```

\l__zrefcheck_zcheck_labels_seq
\g__zrefcheck_id_int
\l__zrefcheck_checkbeg_tl
\l__zrefcheck_link_label_tl
\l__zrefcheck_link_anchor_tl
\l__zrefcheck_link_star_bool
373 \seq_new:N \l__zrefcheck_zcheck_labels_seq
374 \int_new:N \g__zrefcheck_id_int
375 \tl_new:N \l__zrefcheck_checkbeg_tl
376 \tl_new:N \l__zrefcheck_link_label_tl
377 \tl_new:N \l__zrefcheck_link_anchor_tl
378 \bool_new:N \l__zrefcheck_link_star_bool

```

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

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

```
\__zrefcheck_zcheck:nnnn {\labels} {\langle*\rangle} {\langlechecks/options\rangle} {\langletext\rangle}
```

```
379 \cs_new_protected:Npn \__zrefcheck_zcheck:nnnn #1#2#3#4
380 {
381   \group_begin:
```

Process local options and checks.

```
382   \keys_set:nn { zref-check / zcheck } {#3}
383   \seq_set_from_clist:Nn \l__zrefcheck_zcheck_labels_seq {#1}
```

Names of the labels for this `zcheck` call.

```
384   \int_gincr:N \g__zrefcheck_id_int
385   \tl_set:Nx \l__zrefcheck_checkbeg_tl
386     { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
```

Set `checkbeg` label.

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

Typeset `{\langletext\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.

```
388   \seq_get:NN \l__zrefcheck_zcheck_labels_seq \l__zrefcheck_link_label_tl
389   \bool_set:Nn \l__zrefcheck_link_star_bool {#2}
390   \zref@ifrefundefined { \l__zrefcheck_link_label_tl }
```

If the reference is undefined, just typeset.

```
391     {#4}
392     {
393       \bool_if:nTF
394         {
395           \l__zrefcheck_use_hyperref_bool &&
396           ! \l__zrefcheck_link_star_bool
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 } {#4}
403         }
404     } {#4}
405 }
```

Set `checkend` label.

```
406   \bool_if:NT \l__zrefcheck_zcheck_end_label_bool
407   {
408     \zref@labelbylist
409       { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
410       { zrefcheck-end }
411   }
```

Check if `\langlelabels\rangle` are defined.

```
412   \seq_map_function:NN \l__zrefcheck_zcheck_labels_seq \zref@refused
```

Run the checks.

```

413     \_zrefcheck_run_checks:nmx { \l_zrefcheck_zcheck_checks_seq }
414     { \l_zrefcheck_zcheck_labels_seq } { \l_zrefcheck_checkbeg_tl }
415     \group_end:
416 }

```

(End definition for _zrefcheck_zcheck:nnnn.)

5.2 Targets

```

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

417 \NewDocumentCommand \zctarget { m +m }
418 {

```

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.

```

419     \group_begin:
420     \refstepcounter { zrefcheck }
421     \zref@wrapper@babel \_zrefcheck_target_label:n {#1}
422     #2
423     \zref@wrapper@babel
424     \zref@labelbylist { \_zrefcheck_end_lblfmt:n {#1} } { zrefcheck-end }
425     \group_end:
426 }

```

(End definition for \zctarget.)

```

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

427 \NewDocumentEnvironment {zcregion} { m }
428 {
429     \refstepcounter { zrefcheck }
430     \zref@wrapper@babel \_zrefcheck_target_label:n {#1}
431 }
432 {
433     \zref@wrapper@babel
434     \zref@labelbylist { \_zrefcheck_end_lblfmt:n {#1} } { zrefcheck-end }
435 }

```

(End definition for zcregion.)

6 Checks

What is needed define a zref-check check?

First, a conditional function defined with:

```
\prg_new_protected_conditional:Npnn \_zrefcheck_check_<check>:nn #1#2 { F }
```

where <check> is the name of the check, the first argument is the <label> and the second the <reference>. The existence of the check is verified by the existence of the function with this name-scheme (and signatures). As usual, this function must return either \prg_return_true: or \prg_return_false:. Of course, you can define other variants

if you need them internally, it is just that what the package does expect and verifies is the existence of the `:nnF` variant.

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

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

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

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

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

6.1 Single label checks

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

`\c__zrefcheck_single_label_checks_seq`

```

436 \seq_new:N \c__zrefcheck_single_label_checks_seq
437 \seq_set_from_clist:Nn \c__zrefcheck_single_label_checks_seq
438 {
439   thischap ,
440   prevchap ,
441   nextchap ,
442   chapsbefore ,
443   chapsafter ,
444   thissec ,
445   prevsec ,
446   nextsec ,
447   secsbefore ,
448   secsafter ,
449 }
```

(End definition for `\c__zrefcheck_single_label_checks_seq`.)

6.2 Setup

`\l__zrefcheck_zcheck_checks_seq`
`\l__zrefcheck_end_label_required_bool`

```

450 \seq_new:N \l__zrefcheck_zcheck_checks_seq
451 \bool_new:N \l__zrefcheck_zcheck_end_label_bool
```

(End definition for `\l__zrefcheck_zcheck_checks_seq` and `\l__zrefcheck_end_label_required_bool`.)

First, we inherit all the main options into the keys of `zref-check / zcheck`.

```

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

Then we add the checks to it.

```

453 \clist_map_inline:nn
454 {
455     thispage ,
456     prevpage ,
457     nextpage ,
458     facing ,
459     pagegap ,
460     above ,
461     below ,
462     pagesbefore ,
463     ppbefore ,
464     pagesafter ,
465     ppafter ,
466     before ,
467     after ,
468     thischap ,
469     prevchap ,
470     nextchap ,
471     chapsbefore ,
472     chapsafter ,
473     thissec ,
474     prevsec ,
475     nextsec ,
476     secsbefore ,
477     secsafter ,
478     close ,
479     far ,
480 }
481 {
482     \keys_define:nn { zref-check / zcheck }
483     {
484         #1 .code:n =
485         {
486             \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq {#1}
487             \seq_if_in:NnF \c__zrefcheck_single_label_checks_seq {#1}
488             { \bool_set_true:N \l__zrefcheck_zcheck_end_label_bool }
489         } ,
490         #1 .value_forbidden:n = true ,
491     }
492 }

```

6.3 Running

```

\__zrefcheck_run_checks:nnn
    \__zrefcheck_run_checks:nnn {<checks>} {<labels>} {<reference>}
    <checks> are expected to be received as a sequence variable.
493 \cs_new_protected:Npn \__zrefcheck_run_checks:nnn #1#2#3
494 {
495     \group_begin:
496     \seq_map_inline:Nn #2
497     {
498         \seq_map_inline:Nn #1
499         { \__zrefcheck_do_check:nnn {####1} {##1} {#3} }

```

```

500     }
501     \group_end:
502 }
503 \cs_generate_variant:Nn \__zrefcheck_run_checks:nnn { nnn }

```

(End definition for __zrefcheck_run_checks:nnn.)

```

\l__zrefcheck_passedcheck_bool
\l__zrefcheck_onpage_bool
\c__zrefcheck_onpage_checks_seq
504 \bool_new:N \l__zrefcheck_passedcheck_bool
505 \bool_new:N \l__zrefcheck_onpage_bool
506 \seq_new:N \c__zrefcheck_onpage_checks_seq
507 \seq_set_from_clist:Nn \c__zrefcheck_onpage_checks_seq
508 { 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.

```

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

```

```

\__zrefcheck_do_check:nnn
\__zrefcheck_do_check:nnn {<check>} {<label beg>} {<reference beg>}
510 \cs_new_protected:Npn \__zrefcheck_do_check:nnn #1#2#3
511 {
512     \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.

```

513     \zref@ifrefundefined {#2}
514     {}
515     {
516         \tl_if_empty:nF {#1}
517         {
518             \bool_set_true:N \l__zrefcheck_passedcheck_bool
519             \bool_set_false:N \l__zrefcheck_onpage_bool
520             \cs_if_exist:cTF { __zrefcheck_check_ #1 :nnF }
521             {
522                 % ‘‘label beg’’ vs ‘‘reference beg’’.
523                 \use:c { __zrefcheck_check_ #1 :nnF }
524                 {#2} {#3}
525                 { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
526                 % ‘‘reference end’’ \emph{may} exist or not depending on the
527                 % checks.
528                 \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#3} }
529                 {
530                     % ‘‘label end’’ \emph{may} have been created by the
531                     % target commands.
532                     \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
533                     {}
534                     {
535                         % ‘‘label end’’ vs ‘‘reference beg’’.
536                         \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
537                         { \__zrefcheck_end_lblfmt:n {#2} } {#3}
538                         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }

```

```

539     }
540   }
541   {
542     % ‘‘label beg’’ vs ‘‘reference end’’.
543     \exp_args:Nnno \use:c { __zrefcheck_check_ #1 :nnF }
544       {#2} { \__zrefcheck_end_lblfmt:n {#3} }
545       { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
546     % ‘‘label end’’ \emph{may} have been created by the
547     % target commands.
548     \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
549       {}
550     {
551       % ‘‘label end’’ vs ‘‘reference beg’’.
552       \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
553         { \__zrefcheck_end_lblfmt:n {#2} } {#3}
554         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
555       % ‘‘label end’’ vs ‘‘reference end’’.
556       \exp_args:Nnoo \use:c { __zrefcheck_check_ #1 :nnF }
557         { \__zrefcheck_end_lblfmt:n {#2} }
558         { \__zrefcheck_end_lblfmt:n {#3} }
559         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
560     }
561   }

```

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.

```

562     \seq_if_in:NnT \c__zrefcheck_onpage_checks_seq {#1}
563     {
564       \__zrefcheck_check_thispage:nnT
565         {#2} {#3}
566       { \bool_set_true:N \l__zrefcheck_onpage_bool }
567       \__zrefcheck_check_thispage:nnT
568         {#2} { \__zrefcheck_end_lblfmt:n {#3} }
569       { \bool_set_true:N \l__zrefcheck_onpage_bool }
570       \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
571         {}
572       {
573         \__zrefcheck_check_thispage:nnT
574           { \__zrefcheck_end_lblfmt:n {#2} } {#3}
575         { \bool_set_true:N \l__zrefcheck_onpage_bool }
576         \__zrefcheck_check_thispage:nnT
577           { \__zrefcheck_end_lblfmt:n {#2} }
578           { \__zrefcheck_end_lblfmt:n {#3} }
579         { \bool_set_true:N \l__zrefcheck_onpage_bool }
580       }
581     }
582   \bool_if:NTF \l__zrefcheck_passedcheck_bool
583   {
584     \bool_if:nT
585     {
586       \l__zrefcheck_msgonpage_bool &&
587       \l__zrefcheck_onpage_bool
588     }

```

```

589         {
590             \__zrefcheck_message:nnx { double-check } {#1} {#2}
591             { \zref@extractdefault {#3} {page} {'unknown'} }
592         }
593     }
594     {
595         \__zrefcheck_message:nnx { check-failed } {#1} {#2}
596         { \zref@extractdefault {#3} {page} {'unknown'} }
597     }
598 }
599 { \msg_warning:nnn { zref-check } { check-missing } {#1} }
600 }
601 }
602 \group_end:
603 }

```

(End definition for __zrefcheck_do_check:nnn.)

6.4 Conditionals

```

\l__zrefcheck_lbl_int
\l__zrefcheck_ref_int
\l__zrefcheck_lbl_b_int
\l__zrefcheck_ref_b_int

```

More readable scratch variables for the tests.

```

604 \int_new:N \l__zrefcheck_lbl_int
605 \int_new:N \l__zrefcheck_ref_int
606 \int_new:N \l__zrefcheck_lbl_b_int
607 \int_new:N \l__zrefcheck_ref_b_int

```

(End definition for \l__zrefcheck_lbl_int and others.)

6.4.1 This page

```

\__zrefcheck_check_thispage:nn

```

```

608 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thispage:nn #1#2 { T , F , TF }
609 {
610     \group_begin:
611     \bool_set_true:N \l__zrefcheck_integer_bool
612     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
613     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
614     \bool_lazy_and:nnTF
615     { \l__zrefcheck_integer_bool }
616     {
617         \int_compare_p:nNn
618         { \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.

```

619         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
620         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
621     }
622     { \group_insert_after:N \prg_return_true: }
623     { \group_insert_after:N \prg_return_false: }
624 \group_end:
625 }

```

(End definition for __zrefcheck_check_thispage:nn.)

6.4.2 On page

```

\__zrefcheck_check_above:nn
\__zrefcheck_check_below:nn
626 \prg_new_protected_conditional:Npnn \__zrefcheck_check_above:nn #1#2 { F , TF }
627 {
628   \group_begin:
629   \__zrefcheck_check_thispage:nnTF {#1} {#2}
630   {
631     \bool_set_true:N \l__zrefcheck_integer_bool
632     \zrefcheck_get_asint:nnn {#1} { lblseq } { \l__zrefcheck_lbl_int }
633     \zrefcheck_get_asint:nnn {#2} { lblseq } { \l__zrefcheck_ref_int }
634     \bool_lazy_and:nnTF
635     { \l__zrefcheck_integer_bool }
636     {
637       \int_compare_p:nNn
638       { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
639       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
640       ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
641     }
642     { \group_insert_after:N \prg_return_true: }
643     { \group_insert_after:N \prg_return_false: }
644   }
645   { \group_insert_after:N \prg_return_false: }
646   \group_end:
647 }
648 \prg_new_protected_conditional:Npnn \__zrefcheck_check_below:nn #1#2 { F , TF }
649 {
650   \__zrefcheck_check_thispage:nnTF {#1} {#2}
651   {
652     \__zrefcheck_check_above:nnTF {#1} {#2}
653     { \prg_return_false: }
654     { \prg_return_true: }
655   }
656   { \prg_return_false: }
657 }

```

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

6.4.3 Before / After

```

\__zrefcheck_check_before:nn
\__zrefcheck_check_after:nn
658 \prg_new_protected_conditional:Npnn \__zrefcheck_check_before:nn #1#2 { F }
659 {
660   \__zrefcheck_check_pagesbefore:nnTF {#1} {#2}
661   { \prg_return_true: }
662   {
663     \__zrefcheck_check_above:nnTF {#1} {#2}
664     { \prg_return_true: }
665     { \prg_return_false: }
666   }
667 }
668 \prg_new_protected_conditional:Npnn \__zrefcheck_check_after:nn #1#2 { F }
669 {
670   \__zrefcheck_check_pagesafter:nnTF {#1} {#2}

```

```

671     { \prg_return_true: }
672   {
673     \__zrefcheck_check_below:nnTF {#1} {#2}
674     { \prg_return_true: }
675     { \prg_return_false: }
676   }
677 }

```

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

6.4.4 Pages

```

\__zrefcheck_check_nextpage:nn
\__zrefcheck_check_prevpage:nn
\__zrefcheck_check_pagesbefore:nn
\__zrefcheck_check_ppbefore:nn
\__zrefcheck_check_pagesafter:nn
\__zrefcheck_check_ppafter:nn
\__zrefcheck_check_pagegap:nn
\__zrefcheck_check_facing:nn
678 \prg_new_protected_conditional:Npnn \__zrefcheck_check_nextpage:nn #1#2 { F }
679 {
680   \group_begin:
681   \bool_set_true:N \l__zrefcheck_integer_bool
682   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
683   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
684   \bool_lazy_and:nnTF
685     { \l__zrefcheck_integer_bool }
686     {
687       \int_compare_p:nNn
688         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
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 }
696 \prg_new_protected_conditional:Npnn \__zrefcheck_check_prevpage:nn #1#2 { F }
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         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
707         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
708         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
709     }
710     { \group_insert_after:N \prg_return_true: }
711     { \group_insert_after:N \prg_return_false: }
712   \group_end:
713 }
714 \prg_new_protected_conditional:Npnn \__zrefcheck_check_pagesbefore:nn #1#2 { F , TF }
715 {
716   \group_begin:
717   \bool_set_true:N \l__zrefcheck_integer_bool
718   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }

```

```

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

```



```

773 \bool_set_true:N \l__zrefcheck_integer_bool
774 \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
775 \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
776 \bool_lazy_and:nnTF
777 { \l__zrefcheck_integer_bool }
778 {

```

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

```

779 \legacy_if_p:n { @twoside } &&

```

Now we test “facing”.

```

780 (
781 (
782 \int_if_odd_p:n { \l__zrefcheck_ref_int } &&
783 \int_compare_p:nNn
784 { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 }
785 ) ||
786 (
787 \int_if_even_p:n { \l__zrefcheck_ref_int } &&
788 \int_compare_p:nNn
789 { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 }
790 )
791 ) &&
792 ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
793 ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
794 }
795 { \group_insert_after:N \prg_return_true: }
796 { \group_insert_after:N \prg_return_false: }
797 \group_end:
798 }

```

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

6.4.5 Close / Far

```

\__zrefcheck_check_close:nn
\__zrefcheck_check_far:nn
799 \prg_new_protected_conditional:Npnn \__zrefcheck_check_close:nn #1#2 { F , TF }
800 {
801 \group_begin:
802 \bool_set_true:N \l__zrefcheck_integer_bool
803 \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
804 \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
805 \bool_lazy_and:nnTF
806 { \l__zrefcheck_integer_bool }
807 {
808 \int_compare_p:nNn
809 { \int_abs:n { \l__zrefcheck_lbl_int - \l__zrefcheck_ref_int } }
810 <
811 { \l__zrefcheck_close_range_int + 1 } &&
812 ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
813 ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
814 }
815 { \group_insert_after:N \prg_return_true: }
816 { \group_insert_after:N \prg_return_false: }
817 \group_end:

```

```

818 }
819 \prg_new_protected_conditional:Npnn \__zrefcheck_check_far:nn #1#2 { F }
820 {
821   \__zrefcheck_check_close:nnTF {#1} {#2}
822   { \prg_return_false: }
823   { \prg_return_true: }
824 }

```

(End definition for __zrefcheck_check_close:nn and __zrefcheck_check_far:nn.)

6.4.6 Chapter

```

\__zrefcheck_check_thischap:nn
\__zrefcheck_check_nextchap:nn
\__zrefcheck_check_prevchap:nn
\__zrefcheck_check_chapsafter:nn
\__zrefcheck_check_chapsbefore:nn
825 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thischap:nn #1#2 { F }
826 {
827   \group_begin:
828   \bool_set_true:N \l__zrefcheck_integer_bool
829   \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
830   \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
831   \bool_lazy_and:nnTF
832   { \l__zrefcheck_integer_bool }
833   {
834     \int_compare_p:nNn
835     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int } &&

```

‘0’ is the default value of `zc@abschap` property, and means here no `\chapter` has yet been issued, therefore it cannot be “this chapter”, nor “the next chapter”, nor “the previous chapter”, it is just “no chapter”. Note, however, that a statement about a “future” chapter does not require the “current” one to exist. This comment extends to all chapter checks.

```

836     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
837     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
838   }
839   { \group_insert_after:N \prg_return_true: }
840   { \group_insert_after:N \prg_return_false: }
841 \group_end:
842 }
843 \prg_new_protected_conditional:Npnn \__zrefcheck_check_nextchap:nn #1#2 { F }
844 {
845   \group_begin:
846   \bool_set_true:N \l__zrefcheck_integer_bool
847   \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
848   \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
849   \bool_lazy_and:nnTF
850   { \l__zrefcheck_integer_bool }
851   {
852     \int_compare_p:nNn
853     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
854     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
855   }
856   { \group_insert_after:N \prg_return_true: }
857   { \group_insert_after:N \prg_return_false: }
858 \group_end:
859 }

```

```

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

```

(End definition for __zrefcheck_check_thischap:nn and others.)

6.4.7 Section

```

\__zrefcheck_check_thissec:nn
\__zrefcheck_check_nextsec:nn
\__zrefcheck_check_prevsec:nn
\__zrefcheck_check_secsafter:nn
\__zrefcheck_check_secsbefore:nn
913 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thissec:nn #1#2 { F }
914 {
915   \group_begin:
916     \bool_set_true:N \l__zrefcheck_integer_bool
917     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
918     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
919     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
920     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
921     \bool_lazy_and:nnTF
922       { \l__zrefcheck_integer_bool }
923       {
924         \int_compare_p:nNn
925           { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
926         \int_compare_p:nNn
927           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int } &&
928           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
929           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
930       }
931       { \group_insert_after:N \prg_return_true: }
932       { \group_insert_after:N \prg_return_false: }
933   \group_end:
934 }
935 \prg_new_protected_conditional:Npnn \__zrefcheck_check_nextsec:nn #1#2 { F }
936 {
937   \group_begin:
938     \bool_set_true:N \l__zrefcheck_integer_bool
939     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
940     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
941     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
942     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
943     \bool_lazy_and:nnTF
944       { \l__zrefcheck_integer_bool }
945       {
946         \int_compare_p:nNn
947           { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
948         \int_compare_p:nNn
949           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
950         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
951       }
952       { \group_insert_after:N \prg_return_true: }
953       { \group_insert_after:N \prg_return_false: }
954   \group_end:
955 }
956 \prg_new_protected_conditional:Npnn \__zrefcheck_check_prevsec:nn #1#2 { F }
957 {

```

```

958 \group_begin:
959 \bool_set_true:N \l__zrefcheck_integer_bool
960 \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
961 \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
962 \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
963 \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
964 \bool_lazy_and:nnTF
965 { \l__zrefcheck_integer_bool }
966 {
967   \int_compare_p:nNn
968     { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
969   \int_compare_p:nNn
970     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
971   ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
972   ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
973 }
974 { \group_insert_after:N \prg_return_true: }
975 { \group_insert_after:N \prg_return_false: }
976 \group_end:
977 }
978 \prg_new_protected_conditional:Npnn \__zrefcheck_check_secsafter:nn #1#2 { F }
979 {
980   \group_begin:
981     \bool_set_true:N \l__zrefcheck_integer_bool
982     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
983     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
984     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
985     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
986     \bool_lazy_and:nnTF
987     { \l__zrefcheck_integer_bool }
988     {
989       \int_compare_p:nNn
990         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
991       \int_compare_p:nNn
992         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
993       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
994     }
995     { \group_insert_after:N \prg_return_true: }
996     { \group_insert_after:N \prg_return_false: }
997   \group_end:
998 }
999 \prg_new_protected_conditional:Npnn \__zrefcheck_check_secsbefore:nn #1#2 { F }
1000 {
1001   \group_begin:
1002     \bool_set_true:N \l__zrefcheck_integer_bool
1003     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
1004     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
1005     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
1006     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
1007     \bool_lazy_and:nnTF
1008     { \l__zrefcheck_integer_bool }
1009     {
1010       \int_compare_p:nNn
1011         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&

```

```

1012         \int_compare_p:nNn
1013         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
1014         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
1015         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
1016     }
1017     { \group_insert_after:N \prg_return_true: }
1018     { \group_insert_after:N \prg_return_false: }
1019 \group_end:
1020 }

```

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

7 zref-clever integration

There are four tasks `zref-clever` needs to do, in order to offer integration with `zref-check` from the options of `\zcref`: i) set the “beg label”; ii) set the checks options; iii) run the checks; iv) (possibly) set the “end label”. Since ‘ii)’ can be done directly by running `\keys_set:nn { zref-check / zcheck }` on the options received, we provide convenience functions for the other three tasks.

```

\zrefcheck_zcref_beg_label:
    \zrefcheck_zcref_end_label_maybe:
    \zrefcheck_zcref_run_checks_on_labels:n
1021 \cs_new_protected:Npn \zrefcheck_zcref_beg_label:
1022 {
1023     \int_gincr:N \g__zrefcheck_id_int
1024     \tl_set:Nx \l__zrefcheck_checkbeg_tl
1025     { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
1026     \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrefcheck-check }
1027 }
1028 \cs_new_protected:Npn \zrefcheck_zcref_end_label_maybe:
1029 {
1030     \bool_if:NT \l__zrefcheck_zcheck_end_label_bool
1031     {
1032         \zref@labelbylist
1033         { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
1034         { zrefcheck-end }
1035     }
1036 }
1037 \cs_new_protected:Npn \zrefcheck_zcref_run_checks_on_labels:n #1
1038 {
1039     \__zrefcheck_run_checks:nnx
1040     { \l__zrefcheck_zcheck_checks_seq } {#1} { \l__zrefcheck_checkbeg_tl }
1041 }

```

(End definition for `\zrefcheck_zcref_beg_label:`, `\zrefcheck_zcref_end_label_maybe:`, and `\zrefcheck_zcref_run_checks_on_labels:n`. These functions are documented on page ??.)

```

1042 </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	
<code>\A</code>	110
<code>\AddToHook</code>	21, 28, 137, 186, 246
B	
<code>\beginngroup</code>	314
bool commands:	
<code>\bool_if:NTF</code>
...	141, 148, 248, 298, 406, 582, 1030
<code>\bool_if:nTF</code>	393, 584
<code>\bool_lazy_and:nnTF</code>
...	13, 614, 634, 684, 702,
...	720, 739, 758, 776, 805, 831, 849,
...	866, 884, 901, 921, 943, 964, 986, 1007
<code>\bool_new:N</code>	114,
...	115, 191, 234, 352, 378, 451, 504, 505
<code>\bool_set:Nn</code>	389
<code>\bool_set_false:N</code>	122,
...	131, 132, 150, 197, 206, 213, 288,
...	300, 362, 519, 525, 538, 545, 554, 559
<code>\bool_set_true:N</code>	121, 126, 127, 201,
...	207, 212, 240, 295, 488, 518, 566,
...	569, 575, 579, 611, 631, 681, 699,
...	717, 736, 755, 773, 802, 828, 846,
...	863, 881, 898, 916, 938, 959, 981, 1002
<code>\l_tmpa_bool</code> ...	11, 288, 295, 298, 300
C	
<code>\catcode</code>	11
<code>\chapter</code>	2, 26
clist commands:	
<code>\clist_map_inline:nn</code>	453
cs commands:	
<code>\cs:w</code>	257
<code>\cs_end:</code>	257
<code>\cs_generate_variant:Nn</code> ..	60, 503, 509
<code>\cs_if_exist:NTF</code>	252, 520
<code>\cs_new:Npn</code>	319, 320
<code>\cs_new_eq:NN</code>	97, 732, 751
<code>\cs_new_protected:Npn</code>	55, 244, 321,
...	354, 379, 493, 510, 1021, 1028, 1037
<code>\cs_set_protected:Npx</code>	254
D	
<code>\d</code>	110
E	
<code>\emph</code>	526, 530, 546
<code>\endgroup</code>	317
<code>\endinput</code>	12
exp commands:	
<code>\exp_args:Nnno</code>	262, 509, 543
<code>\exp_args:Nno</code>	536, 552
<code>\exp_args:Nnoo</code>	556
<code>\exp_args:Nx</code>	399
<code>\exp_not:n</code>	256
F	
file commands:	
<code>\file_if_exist:nTF</code>	281
<code>\fmtversion</code>	3
G	
group commands:	
<code>\group_begin:</code>	284, 381,
...	419, 495, 512, 610, 628, 680, 698,
...	716, 735, 754, 772, 801, 827, 845,
...	862, 880, 897, 915, 937, 958, 980, 1001
<code>\group_end:</code>	311, 415,
...	425, 501, 602, 624, 646, 694, 712,
...	730, 749, 768, 797, 817, 841, 858,
...	876, 893, 911, 933, 954, 976, 997, 1019
<code>\group_insert_after:N</code>	622, 623, 642,
...	643, 645, 692, 693, 710, 711, 728,
...	729, 747, 748, 766, 767, 795, 796,
...	815, 816, 839, 840, 856, 857, 874,
...	875, 891, 892, 909, 910, 931, 932,
...	952, 953, 974, 975, 995, 996, 1017, 1018
H	
<code>\hyperlink</code>	14, 402
I	
<code>\ifdraft</code>	170, 205
<code>\IfFormatAtLeastTF</code>	3, 4
<code>\ifoptionfinal</code>	176, 211
int commands:	
<code>\int_abs:n</code>	762, 809
<code>\int_compare_p:nNn</code> ..	617, 619, 620,
...	637, 639, 640, 687, 689, 690, 705,
...	707, 708, 723, 725, 726, 742, 744,
...	745, 761, 763, 764, 783, 788, 792,
...	793, 808, 812, 813, 834, 836, 837,
...	852, 854, 869, 871, 872, 887, 889,
...	904, 906, 907, 924, 926, 928, 929,
...	946, 948, 950, 967, 969, 971, 972,
...	989, 991, 993, 1010, 1012, 1014, 1015
<code>\int_eval:n</code>	224, 359

<code>\int_gincr:N</code>	23, 29, 384, 1023	<code>\prg_new_protected_conditional:Npnn</code>	
<code>\int_if_even_p:n</code>	787	16, 108, 608,
<code>\int_if_odd_p:n</code>	782	626, 648, 658, 668, 678, 696, 714,	
<code>\int_incr:N</code>	301	733, 752, 770, 799, 819, 825, 843,	
<code>\int_new:N</code>		860, 878, 895, 913, 935, 956, 978, 999	
..	19, 20, 218, 374, 604, 605, 606, 607	<code>\prg_return_false:</code>	
<code>\int_set:Nn</code>	224, 227, 359	..	16, 101, 105, 112, 623, 643, 645,
<code>\int_use:N</code>	26, 30, 303, 319	653, 656, 665, 675, 693, 711, 729,	
<code>\int_zero:N</code>	24, 285	748, 767, 796, 816, 822, 840, 857,	
<code>\l_tmpa_int</code>	285, 301, 303	875, 892, 910, 932, 953, 975, 996, 1018	
int internal commands:		<code>\prg_return_true:</code>	
<code>__int_to_roman:w</code>	4, 5, 97	16, 104, 111, 622, 642, 654,
ior commands:		661, 664, 671, 674, 692, 710, 728,	
<code>\ior_close:N</code>	312	747, 766, 795, 815, 823, 839, 856,	
<code>\ior_map_variable:NNn</code>	289	874, 891, 909, 931, 952, 974, 995, 1017	
<code>\ior_open:Nn</code>	283	<code>\ProcessKeysOptions</code>	276
<code>\g_tmpa_ior</code>	283, 289, 312	prop commands:	
iow commands:		<code>\prop_get:NnNTF</code>	326
<code>\iow_newline:</code>	75, 79, 82	<code>\prop_gput:Nnn</code>	302
		<code>\prop_new:N</code>	279
K		<code>\providecommand</code>	3
keys commands:		<code>\ProvidesExplPackage</code>	14
<code>\keys_define:nn</code>	116, 152,		
159, 188, 192, 219, 235, 266, 452, 482		R	
<code>\keys_set:nn</code>	278, 382	<code>\refstepcounter</code>	11, 16, 420, 429
		regex commands:	
L		<code>\regex_match:nnTF</code>	110
<code>\label</code>	11	<code>\RequirePackage</code>	16, 17, 18, 143, 275
legacy commands:		<code>\romannumeral</code>	4
<code>\legacy_if_p:n</code>	779		
<code>\let</code>	315	S	
		<code>\section</code>	28
M		seq commands:	
<code>\MessageBreak</code>	10	<code>\seq_get:NN</code>	388
msg commands:		<code>\seq_if_in:NnTF</code>	487, 562
<code>\msg_line_context:</code>		<code>\seq_map_function:NN</code>	412
.....	62, 64, 66, 68, 70, 72, 86, 89	<code>\seq_map_inline:Nn</code>	496, 498
<code>\msg_new:nnn</code>	61,	<code>\seq_new:N</code>	373, 436, 450, 506
63, 65, 67, 69, 71, 73, 78, 80, 85, 87, 92		<code>\seq_put_right:Nn</code>	486
<code>\msg_warning:nn</code> ...	149, 155, 183, 226	<code>\seq_set_from_clist:Nn</code> ..	383, 437, 507
<code>\msg_warning:nnn</code>	262, 270, 599	<code>\setcounter</code>	318
<code>\msg_warning:nnnn</code> ..	328, 337, 345, 366	sys commands:	
		<code>\c_sys_jobname_str</code>	280
N			
<code>\newcounter</code>	316	T	
<code>\NewDocumentCommand</code>	277, 371, 417	TeX and L ^A T _E X 2 _ε commands:	
<code>\NewDocumentEnvironment</code>	427	<code>\@addtoreset</code>	315
<code>\noexpand</code>	76	<code>\@currentlabel</code>	16
		<code>\@ifl@t@r</code>	3
P		<code>\@ifpackageloaded</code>	139
<code>\PackageError</code>	7	<code>\@newl@bel</code>	9
<code>\pageref</code>	11	<code>\ltx@gobbletwo</code>	315
prg commands:		<code>\zref@addprop</code>	17, 27, 31, 144
<code>\prg_new_conditional:Npnn</code>	98	<code>\zref@addprops</code>	33, 41, 49

\l__zrefcheck_checkbeg_tl	373, 385, 387, 409, 414, 1024, 1026, 1033, 1040	\l__zrefcheck_msglevel_tl	57, 158
\l__zrefcheck_close_range_int	218, 811	\l__zrefcheck_msgonpage_bool	191, 586
__zrefcheck_do_check:nnn	19, 499, 510	\l__zrefcheck_onpage_bool	504, 519, 566, 569, 575, 579, 587
\l__zrefcheck_end_label_required_bool	450	\c__zrefcheck_onpage_checks_seq	504, 562
__zrefcheck_end_lblfmt:n	12, 320, 409, 424, 434, 528, 532, 537, 544, 548, 553, 557, 558, 568, 570, 574, 577, 578, 1033	\l__zrefcheck_passedcheck_bool	504, 518, 525, 538, 545, 554, 559, 582
__zrefcheck_get_asint:nnn	12	\l__zrefcheck_propval_tl	353, 356, 357, 359
__zrefcheck_get_astl:nnn	12	\l__zrefcheck_ref_b_int	604, 920, 925, 942, 947, 963, 968, 985, 990, 1006, 1011
\g__zrefcheck_id_int	373, 384, 386, 1023, 1025	\l__zrefcheck_ref_int	604, 613, 618, 620, 633, 638, 640, 683, 688, 690, 701, 706, 708, 719, 724, 726, 738, 743, 745, 757, 762, 764, 775, 782, 784, 787, 789, 793, 804, 809, 813, 830, 835, 837, 848, 853, 865, 870, 872, 883, 888, 900, 905, 907, 918, 927, 929, 940, 949, 961, 970, 972, 983, 992, 1004, 1013, 1015
_zrefcheck_int_to_roman:w	97	__zrefcheck_run_checks:nnn	18, 413, 493, 1039
\l__zrefcheck_integer_bool	13, 352, 362, 611, 615, 631, 635, 681, 685, 699, 703, 717, 721, 736, 740, 755, 759, 773, 777, 802, 806, 828, 832, 846, 850, 863, 867, 881, 885, 898, 902, 916, 922, 938, 944, 959, 965, 981, 987, 1002, 1008	\c__zrefcheck_single_label_checks_seq	17, 436, 487
__zrefcheck_is_integer:n	5, 97	_zrefcheck_target_label:n	244, 254, 421, 430
__zrefcheck_is_integer:nTF	357	\l__zrefcheck_target_label_bool	234, 240, 248
_zrefcheck_is_integer_rgx:n	5, 108	\l__zrefcheck_target_label_tl	233, 250, 251, 252, 257, 263
_zrefcheck_is_integer_rgx:nTF	223	\l__zrefcheck_use_hyperref_bool	114, 141, 150, 395
\l__zrefcheck_lbl_b_int	604, 919, 925, 941, 947, 962, 968, 984, 990, 1005, 1011	\l__zrefcheck_warn_hyperref_bool	114, 148
\l__zrefcheck_lbl_int	604, 612, 618, 619, 632, 638, 639, 682, 688, 689, 700, 706, 707, 718, 724, 725, 737, 743, 744, 756, 762, 763, 774, 784, 789, 792, 803, 809, 812, 829, 835, 836, 847, 853, 854, 864, 870, 871, 882, 888, 889, 899, 905, 906, 917, 927, 928, 939, 949, 950, 960, 970, 971, 982, 992, 993, 1003, 1013, 1014	__zrefcheck_zcheck:nnnn	15, 372, 379
\l__zrefcheck_link_anchor_tl	373, 401, 402	__zrefcheck_zcheck:nnnnn	13, 14, 19
\l__zrefcheck_link_label_tl	373, 388, 390, 400	\l__zrefcheck_zcheck_checks_seq	413, 450, 486, 1040
\l__zrefcheck_link_star_bool	373, 389, 396	\l__zrefcheck_zcheck_end_label_bool	406, 451, 488, 1030
__zrefcheck_message:nnnn	55, 590, 595	\l__zrefcheck_zcheck_labels_seq	373, 383, 388, 412, 414
		\zrefchecksetup	9, 277