

The zref-check package implementation*

Gustavo Barros[†]

2023-06-14

Contents

1	Initial setup	2
2	Dependencies	2
3	zref setup	2
4	Plumbing	3
4.1	Messages	3
4.2	Integer testing	4
4.3	Options	6
4.4	Position on page	10
4.5	Counter	12
4.6	Label formats	12
4.7	Property values	13
5	User interface	15
5.1	\zcheck	15
5.2	Targets	17
6	Checks	17
6.1	Single label checks	18
6.2	Setup	18
6.3	Running	19
6.4	Conditionals	22
6.4.1	This page	22
6.4.2	On page	23
6.4.3	Before / After	24
6.4.4	Pages	24
6.4.5	Close / Far	26
6.4.6	Chapter	27
6.4.7	Section	29
7	zref-clever integration	31

*This file describes v0.3.3, released 2023-06-14.

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

1 Initial setup

Start the DocStrip guards.

```
1 <*package>
```

Identify the internal prefix (L^AT_EX3 DocStrip convention).

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

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

```
3 \NeedsTeXFormat{LaTeX2e}
4 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
5 \IfFormatAtLeastTF{2021-06-01}
6 {}
7 {%
8   \PackageError{zref-check}{LaTeX kernel too old}
9   {%
10     'zref-check' requires a LaTeX kernel newer than 2021-06-01.%
11     \MessageBreak Loading will abort!%
12   }%
13   \endinput
14 }%
```

Identify the package.

```
15 \ProvidesExplPackage {zref-check} {2023-06-14} {0.3.3}
16 {Flexible cross-references with contextual checks based on zref}
```

2 Dependencies

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

3 zref setup

Provide absolute counters for section and chapter, and respective `zref` properties, so that we can make checks about relation of chapters/sections regardless of internal counters, since we don't get those for the unnumbered (starred) ones. Thanks Ulrike Fischer for suggestions at TeX.SX about the proper place to make the hooks for this purpose.

```
20 \newcounter { zc@abschap }
21 \newcounter { zc@abssec } [ zc@abschap ]
```

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.

```
22 \AddToHook { cmd / chapter / before }
23 { \stepcounter { zc@abschap } }
24 \zref@newprop { zc@abschap } [0] { \int_use:N \c@zc@abschap }
25 \zref@addprop \ZREF@mainlist { zc@abschap }
```

```

26 \AddToHook { cmd / section / before }
27 { \stepcounter { zc@abssec } }
28 \zref@newprop { zc@abssec } [0] { \int_use:N \c@zc@abssec }
29 \zref@addprop \ZREF@mainlist { zc@abssec }

```

These are the lists of properties to be used by zref-check, that is, the list of properties the references and targets store. This is the minimum set required, more properties may be added according to options. For user facing labels, we must use the `main` property list, so that zref-clever can also retrieve the properties it needs to refer to them.

```

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

```

For zref-vario we only need page information, since we only perform above and below checks there.

```

45 \zref@newlist { zrefcheck-zrefvario }
46 \zref@addprops { zrefcheck-zrefvario }
47 {
48   page , % for messages
49   abspage ,
50 }

```

4 Plumbing

4.1 Messages

```

\__zrefcheck_message:nnnn
\__zrefcheck_message:nnnx
51 \cs_new_protected:Npn \__zrefcheck_message:nnnn #1#2#3#4
52 {
53   \use:c { msg_ \l__zrefcheck_msglevel_tl :nnnnn }
54   { zref-check } {#1} {#2} {#3} {#4}
55 }
56 \cs_generate_variant:Nn \__zrefcheck_message:nnnn { nnnx }

(End of definition for \__zrefcheck_message:nnnn.)

57 \msg_new:nnn { zref-check } { check-failed }
58 {
59   Check~failed~\msg_line_context:~
60   Failed-check~'#1'~for~label~'#2'~on~page~#3.
61 }
62 \msg_new:nnn { zref-check } { double-check }

```

```

63 {
64   Same-page-check~\msg_line_context:..~
65   Double-check~'#1'~for-label~'#2'~on-page~#3.
66 }
67 \msg_new:nnn { zref-check } { check-missing }
68 { Check~'#1'~not-defined~\msg_line_context:.. }
69 \msg_new:nnn { zref-check } { property-undefined }
70 { Property~'#1'~not-defined~\msg_line_context:.. }
71 \msg_new:nnn { zref-check } { property-not-in-label }
72 { Label~'#1'~has-no-property~'#2'~\msg_line_context:.. }
73 \msg_new:nnn { zref-check } { property-not-integer }
74 { Property~'#1'~for-label~'#2'~not-an-integer~\msg_line_context:.. }
75 \msg_new:nnn { zref-check } { hyperref-preamble-only }
76 {
77   Option~'hyperref'~only-available-in-the-preamble. \iow_newline:
78   Use-the-starred-version-of~'\iow_char:N\zcheck'~instead.
79 }
80 \msg_new:nnn { zref-check } { missing-hyperref }
81 { Missing~'hyperref'~package. \iow_newline: Setting~'hyperref=false'. }
82 \msg_new:nnn { zref-check } { ignore-document-only }
83 {
84   Option~'ignore'~only-available-in-the-document. \iow_newline:
85   Use-option~'msglevel'~instead.
86 }
87 \msg_new:nnn { zref-check } { option-preamble-only }
88 { Option~'#1'~is-preamble-only~\msg_line_context:.. }
89 \msg_new:nnn { zref-check } { closerange-not-positive-integer }
90 {
91   Option~'closerange'~not-a-positive-integer~\msg_line_context:..~
92   Using-default-value.
93 }
94 \msg_new:nnn { zref-check } { labelcmd-undefined }
95 {
96   Control-sequence-named~'#1'~used-in-option~'labelcmd'~is-not-defined.~
97   Using-default-value.
98 }
99 \msg_new:nnn { zref-check } { option-deprecated-with-alternative }
100 {
101   Option~'#1'~has-been-deprecated~\msg_line_context:.\iow_newline:
102   Use~'#2'~instead.
103 }
104 \msg_new:nnn { zref-check } { option-deprecated }
105 { Option~'#1'~has-been-deprecated~\msg_line_context:.. }
106 \msg_new:nnn { zref-check } { load-time-options }
107 {
108   'zref-check'~does-not-accept-load-time-options.~
109   To-configure-package-options,~use~'\iow_char:N\zrefchecksetup'.
110 }

```

4.2 Integer testing

`__zrefcheck_is_integer:n` From <https://tex.stackexchange.com/a/244405> (thanks Enrico Gregorio, aka 'egreg'),
`__zrefcheck_int_to_roman:w` 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.

```

111 \cs_new_eq:NN \__zrefcheck_int_to_roman:w \__int_to_roman:w
112 \prg_new_conditional:Npnn \__zrefcheck_is_integer:n #1 { p, T , F , TF }
113 {
114   \tl_if_empty:oTF {#1}
115   { \prg_return_false: }
116   {
117     \tl_if_empty:oTF { \__zrefcheck_int_to_roman:w -0#1 }
118     { \prg_return_true: }
119     { \prg_return_false: }
120   }
121 }
```

(End of 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.

```

122 \prg_new_protected_conditional:Npnn \__zrefcheck_is_integer_rgx:n #1 { TF }
123 {
124   \regex_match:nnTF { \A\d+\Z } {#1}
125   { \prg_return_true: }
126   { \prg_return_false: }
127 }
```

(End of definition for `__zrefcheck_is_integer_rgx:n`.)

4.3 Options

hyperref option

`\l__zrefcheck_use_hyperref_bool`
`\l__zrefcheck_warn_hyperref_bool`

```
128 \bool_new:N \l__zrefcheck_use_hyperref_bool
129 \bool_new:N \l__zrefcheck_warn_hyperref_bool
130 \keys_define:nn { zref-check }
131 {
132   hyperref .choice: ,
133   hyperref / auto .code:n =
134   {
135     \bool_set_true:N \l__zrefcheck_use_hyperref_bool
136     \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
137   } ,
138   hyperref / true .code:n =
139   {
140     \bool_set_true:N \l__zrefcheck_use_hyperref_bool
141     \bool_set_true:N \l__zrefcheck_warn_hyperref_bool
142   } ,
143   hyperref / false .code:n =
144   {
145     \bool_set_false:N \l__zrefcheck_use_hyperref_bool
146     \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
147   } ,
148   hyperref .initial:n = auto ,
149   hyperref .default:n = auto
150 }

151 \AddToHook { begindocument }
152 {
153   \@ifpackageloaded { hyperref }
154   {
155     \bool_if:NT \l__zrefcheck_use_hyperref_bool
156     { \RequirePackage { zref-hyperref } }
157   }
158   {
159     \bool_if:NT \l__zrefcheck_warn_hyperref_bool
160     { \msg_warning:nn { zref-check } { missing-hyperref } }
161     \bool_set_false:N \l__zrefcheck_use_hyperref_bool
162   }
163   \keys_define:nn { zref-check }
164   {
165     hyperref .code:n =
166     { \msg_warning:nn { zref-check } { hyperref-preamble-only } }
167   }
168 }
```

msglevel option

`\l__zrefcheck_msglevel_tl`

```

169 \tl_new:N \l__zrefcheck_msglevel_tl
170 \keys_define:nn { zref-check }
171 {
172   msglevel .choice: ,
173   msglevel / warn .code:n =
174     { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } } ,
175   msglevel / info .code:n =
176     { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } } ,
177   msglevel / none .code:n =
178     { \tl_set:Nn \l__zrefcheck_msglevel_tl { none } } ,
179   msglevel / infoifdraft .code:n =
180     {
181       \ifdraft
182         { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
183         { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
184     } ,
185   msglevel / warniffinal .code:n =
186     {
187       \ifoptionfinal
188         { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
189         { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
190     } ,
191   msglevel / obeydraft .code:n =
192     {
193       % NOTE Option value deprecated in 2021-12-07 for v0.2.2.
194       \msg_warning:nnnn { zref-check } { option-deprecated-with-alternative }
195       { msglevel=obeydraft } { msglevel=infoifdraft }
196     } ,
197   msglevel / obeyfinal .code:n =
198     {
199       % NOTE Option value deprecated in 2021-12-07 for v0.2.2.
200       \msg_warning:nnnn { zref-check } { option-deprecated-with-alternative }
201       { msglevel=obeyfinal } { msglevel=warniffinal }
202     } ,
203   msglevel .value_required:n = true ,
204   msglevel .initial:n = warn ,

```

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

```

205   ignore .code:n =
206     { \msg_warning:nn { zref-check } { ignore-document-only } } ,
207   ignore .value_forbidden:n = true
208 }

209 \AddToHook { begindocument }
210 {
211   \keys_define:nn { zref-check }
212     { ignore .meta:n = { msglevel = none } }
213 }

```

onpage option

\l__zrefcheck_msgonpage_bool

```
214 \bool_new:N \l__zrefcheck_msgonpage_bool
215 \keys_define:nn { zref-check }
216 {
217   onpage .choice: ,
218   onpage / labelseq .code:n =
219   {
220     \bool_set_false:N \l__zrefcheck_msgonpage_bool
221   } ,
222   onpage / msg .code:n =
223   {
224     \bool_set_true:N \l__zrefcheck_msgonpage_bool
225   } ,
226   onpage / labelseqifdraft .code:n =
227   {
228     \ifdraft
229     { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
230     { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
231   } ,
232   onpage / msgiffinal .code:n =
233   {
234     \ifoptionfinal
235     { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
236     { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
237   } ,
238   onpage / obeydraft .code:n =
239   {
240     % NOTE Option value deprecated in 2021-12-07 for v0.2.2.
241     \msg_warning:nnnn { zref-check }{ option-deprecated-with-alternative }
242     { onpage=obeydraft } { onpage=labelseqifdraft }
243   } ,
244   onpage / obeyfinal .code:n =
245   {
246     % NOTE Option value deprecated in 2021-12-07 for v0.2.2.
247     \msg_warning:nnnn { zref-check }{ option-deprecated-with-alternative }
248     { onpage=obeyfinal } { onpage=msgiffinal }
249   } ,
250   onpage .value_required:n = true ,
251   onpage .initial:n = labelseq
252 }
```


closerange option

\l__zrefcheck_close_range_int

```
253 \int_new:N \l__zrefcheck_close_range_int
254 \keys_define:nn { zref-check }
255 {
256   closerange .code:n =
257   {
258     \__zrefcheck_is_integer_rgx:nTF {#1}
259     { \int_set:Nn \l__zrefcheck_close_range_int { \int_eval:n {#1} } }
260     {
261       \msg_warning:nn { zref-check } { closerange-not-positive-integer }
262       \int_set:Nn \l__zrefcheck_close_range_int { 5 }
263     }
264   } ,
265   closerange .value_required:n = true ,
266   closerange .initial:n = 5
267 }
```

labelcmd option

```
268 \keys_define:nn { zref-check }
269 {
270   labelcmd .code:n =
271   {
272     % NOTE Option value deprecated in 2022-02-08 for v0.2.4.
273     \msg_warning:nnn { zref-check } { option-deprecated }
274     { labelcmd }
275   } ,
276 }
```

Package options

zref-check does not accept load-time options. Despite the tradition of so doing, Joseph Wright has a point in recommending otherwise at <https://chat.stackexchange.com/transcript/message/60360822#60360822>: separating “loading the package” from “configuring the package” grants less trouble with “option clashes” and with expansion of options at load-time.

```
277 \bool_lazy_and:nnT
278 { \tl_if_exist_p:c { opt@ zref-check.sty } }
279 { ! \tl_if_empty_p:c { opt@ zref-check.sty } }
280 { \msg_warning:nn { zref-check } { load-time-options } }
```

\zrefchecksetup Provide \zrefchecksetup.

```
281 \NewDocumentCommand \zrefchecksetup { m }
282 { \keys_set:nn { zref-check } {#1} }
```

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

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

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

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

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

enabled *with french* and we load the package *after babel* things do break, but not quite because of the colons in the labels. Even siunitx breaks in the same conditions...

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

\g_zrefcheck_auxfile_lblseq_prop

```
283 \prop_new:N \g_zrefcheck_auxfile_lblseq_prop
```

```
284 \tl_gset:Nn \g_tmpa_tl { \c_sys_jobname_str .aux }
```

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

```
286 {
```

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.

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

```
288 \group_begin:
```

```
289 \int_zero:N \l_tmpa_int
```

```
290 \tl_clear:N \l_tmpa_tl
```

```
291 \tl_clear:N \l_tmpb_tl
```

```
292 \bool_set_false:N \l_tmpa_bool
```

```
293 \ior_map_variable:NNn \g_tmpa_ior \l_tmpa_tl
```

```
294 {
```

```
295 \tl_map_variable:NNn \l_tmpa_tl \l_tmpb_tl
```

```
296 {
```

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

```
298 {
```

Found a \zref@label, signal it.

```
299 \bool_set_true:N \l_tmpa_bool
```

```
300 }
```

```
301 {
```

```
302 \bool_if:NTF \l_tmpa_bool
```

```
303 {
```

```
304 \bool_set_false:N \l_tmpa_bool
```

```
305 \int_incr:N \l_tmpa_int
```

```
306 \prop_gput:Nxx \g_zrefcheck_auxfile_lblseq_prop
```

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

```
308 }
```

```
309 {
```

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.

```

310         \tl_map_break:
311     }
312 }
313 }
314 }
315 \group_end:
316 \ior_close:N \g_tmpa_ior
317 }

```

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.

There's an interesting answer by David Carlisle at <https://tex.stackexchange.com/a/419189> to decide whether to typeset “above” or “below” using a method which essentially boils down to “position in the `.aux` file”.

4.5 Counter

We need a dedicated counter for the labels generated by the checks and targets. The value of the counter is not relevant, we just need it to be able to set proper anchors with `\refstepcounter`. And, since I couldn't find a `\refstepcounter` equivalent in L3, we use a standard 2e counter here. I'm also using the technique to ensure the counter is never reset that is used by `zref-abspage.sty` and `\zref@require@unique`. Indeed, the requirements are the same, we need numbers ensured to be *unique* in the counter.

```

318 \begingroup
319 \let \@addtoreset \ltx@gobbletwo
320 \newcounter { zrefcheck }
321 \endgroup
322 \setcounter { zrefcheck } { 0 }

```

4.6 Label formats

```

\__zrefcheck_check_lblfmt:n \__zrefcheck_check_lblfmt:n {<check id int>}
323 \cs_new:Npn \__zrefcheck_check_lblfmt:n #1 { zrefcheck@ \int_use:N #1 }
(End of definition for \__zrefcheck_check_lblfmt:n.)

\__zrefcheck_end_lblfmt:n \__zrefcheck_end_lblfmt:n {<label>}
324 \cs_new:Npn \__zrefcheck_end_lblfmt:n #1 { #1 @zrefcheck }

```

(End of 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>}

325 \cs_new_protected:Npn \zrefcheck_get_astl:nnn #1#2#3
326 {
327   \tl_clear:N #3
328   \tl_if_eq:nnTF {#2} { lblseq }
329   {
330     \prop_get:NnNF \g__zrefcheck_auxfile_lblseq_prop {#1} #3
331     {
332       \msg_warning:nnnn { zref-check }
333       { property-not-in-label } {#1} {#2}
334     }
335   }
336   {

```

There are three things we need to check to ensure the information we are trying to retrieve here exists: the existence of `{<label>}`, the existence of `{<prop>}`, and whether the particular label being queried actually contains the property. If that’s all in place, the value is passed to the checks, and it’s their responsibility to verify the consistency of this value.

The existence of the label is an user facing issue, and a warning for this is placed in `__zrefcheck_zcheck:nnnnn` (and done with `\zref@refused`). We do check here though for definition with `\zref@ifrefundefined` and silently do nothing if it is undefined, to reduce irrelevant warnings in a fresh compilation round. The other two are more “internal” problems, either some problem with the checks, or with the configuration of `zref` for their consumption.

```

337   \zref@ifrefundefined {#1}
338   {}
339   {
340     \zref@ifpropundefined {#2}
341     { \msg_warning:nnnn { zref-check } { property-undefined } {#2} }
342     {
343       \zref@ifrefcontainsprop {#1} {#2}
344       {

```

```

345         \tl_set:Nx #3
346         { \zref@extractdefault {#1} {#2} { \c_empty_tl } }
347     }
348     {
349         \msg_warning:nnnn
350         { zref-check } { property-not-in-label } {#1} {#2}
351     }
352 }
353 }
354 }
355 }

```

(End of 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).

```

356 \bool_new:N \l__zrefcheck_integer_bool

```

\l__zrefcheck_propval_tl

```

357 \tl_new:N \l__zrefcheck_propval_tl

```

\zrefcheck_get_asint:nnn

```

\zrefcheck_get_asint:nnn {<label>} {<prop>} {<int var>}
358 \cs_new_protected:Npn \zrefcheck_get_asint:nnn #1#2#3
359 {
360     \zrefcheck_get_astl:nnn {#1} {#2} { \l__zrefcheck_propval_tl }
361     \__zrefcheck_is_integer:nTF { \l__zrefcheck_propval_tl }
362     {

```

Make it an integer data type.

```

363         \int_set:Nn #3 { \int_eval:n { \l__zrefcheck_propval_tl } }
364     }
365     {
366         \bool_set_false:N \l__zrefcheck_integer_bool
367         \zref@ifrefundefined {#1}

```

Keep silent if ref is undefined to reduce irrelevant warnings in a fresh compilation round. Again, this is also not the point to check for undefined references, that's a task for __zrefcheck_zcheck:nnnnn.

```

368     { }
369     {

```

```

370         \msg_warning:nnnn { zref-check }
371         { property-not-integer } {#2} {#1}
372     }
373 }
374 }

```

(End of definition for \zrefcheck_get_asint:nnn.)

5 User interface

5.1 \zcheck

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

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

```

375 \NewDocumentCommand \zcheck { s O { } m m }
376 { \zref@wrapper@babel \_zrefcheck_zcheck:nnnn {#3} {#1} {#2} {#4} }

```

(End of 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

```

```

377 \seq_new:N \l__zrefcheck_zcheck_labels_seq
378 \int_new:N \g__zrefcheck_id_int
379 \tl_new:N \l__zrefcheck_checkbeg_tl
380 \tl_new:N \l__zrefcheck_link_label_tl
381 \tl_new:N \l__zrefcheck_link_anchor_tl
382 \bool_new:N \l__zrefcheck_link_star_bool

```

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

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

```

383 \cs_new_protected:Npn \_zrefcheck_zcheck:nnnn #1#2#3#4
384 {
385     \group_begin:

```

Process local options and checks.

```

386     \keys_set:nn { zref-check / zcheck } {#3}
387     \seq_set_from_clist:Nn \l__zrefcheck_zcheck_labels_seq {#1}

```

Names of the labels for this zcheck call.

```

388      \int_gincr:N \g__zrefcheck_id_int
389      \tl_set:Nx \l__zrefcheck_checkbeg_tl
390      { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }

```

Set checkbeg label.

```

391      \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrefcheck-check }

```

Typeset $\langle\textit{text}\rangle$, 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.

```

392      \seq_get:NN \l__zrefcheck_zcheck_labels_seq \l__zrefcheck_link_label_tl
393      \bool_set:Nn \l__zrefcheck_link_star_bool {#2}
394      \zref@ifrefundefined { \l__zrefcheck_link_label_tl }

```

If the reference is undefined, just typeset.

```

395      {#4}
396      {
397        \bool_if:nTF
398        {
399          \l__zrefcheck_use_hyperref_bool &&
400          ! \l__zrefcheck_link_star_bool
401        }
402        {
403          \exp_args:Nx \zrefcheck_get_astl:nnn
404          { \l__zrefcheck_link_label_tl }
405          { anchor } { \l__zrefcheck_link_anchor_tl }
406          \hyperlink { \l__zrefcheck_link_anchor_tl } {#4}
407        }
408        {#4}
409      }

```

Set checkend label.

```

410      \bool_if:NT \l__zrefcheck_zcheck_end_label_bool
411      {
412        \zref@labelbylist
413        { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
414        { zrefcheck-end }
415      }

```

Check if $\langle\textit{labels}\rangle$ are defined.

```

416      \seq_map_function:NN \l__zrefcheck_zcheck_labels_seq \zref@refused

```

Run the checks.

```

417      \__zrefcheck_run_checks:nnx { \l__zrefcheck_zcheck_checks_seq }
418      { \l__zrefcheck_zcheck_labels_seq } { \l__zrefcheck_checkbeg_tl }
419      \group_end:
420      }

```

(End of definition for $\backslash_zrefcheck_zcheck:nnnn$.)

5.2 Targets

```

\zctarget      \zctarget{<label>}{<text>}
421 \NewDocumentCommand \zctarget { m +m }
422 {
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.
423 \group_begin:
424 \refstepcounter { zrefcheck }
425 \zref@wrapper@babel \zref@label {#1}
426 #2
427 \tl_if_empty:nF {#2}
428 {
429 \zref@wrapper@babel
430 \zref@labelbylist { \__zrefcheck_end_lblfmt:n {#1} } { zrefcheck-end }
431 }
432 \group_end:
433 }

```

(End of definition for \zctarget.)

```

zcregion      \begin{zcregion}{<label>}
...
\end{zcregion}
434 \NewDocumentEnvironment {zcregion} { m }
435 {
436 \refstepcounter { zrefcheck }
437 \zref@wrapper@babel \zref@label {#1}
438 }
439 {
440 \zref@wrapper@babel
441 \zref@labelbylist { \__zrefcheck_end_lblfmt:n {#1} } { zrefcheck-end }
442 }

```

(End of 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 $\langle check \rangle$ is the name of the check, the first argument is the $\{\langle label \rangle\}$ and the second the $\{\langle reference \rangle\}$. 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 *<reference>* argument is also a label, actually a pair of them, as set by `\zcheck`. For the “labels”, any `zref` property in `zref`’s main list is available, the “references” store the properties in the `zrefcheck` list. Besides those, there is also the `lblseq` (fake) property (for either “labels” or “references”), stored in `\g__zrefcheck_auxfile_lblseq_prop`.

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

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

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

6.1 Single label checks

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

`\c__zrefcheck_single_label_checks_seq`

```

443 \seq_const_from_clist:Nn \c__zrefcheck_single_label_checks_seq
444 {
445   thischap ,
446   prevchap ,
447   nextchap ,
448   chapsbefore ,
449   chapsafter ,
450   thissec ,
451   prevsec ,
452   nextsec ,
453   secsbefore ,
454   secsafter ,
455 }
```

6.2 Setup

`\l__zrefcheck_zcheck_checks_seq`
`\l__zrefcheck_end_label_required_bool`

```

456 \seq_new:N \l__zrefcheck_zcheck_checks_seq
457 \bool_new:N \l__zrefcheck_zcheck_end_label_bool
```

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

```

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

Then we add the checks to it.

```

459 \clist_map_inline:nn
460 {
461   thispage ,
462   prevpage ,
```

```

463     nextpage ,
464     facing ,
465     otherpage ,
466     pagegap ,
467     above ,
468     below ,
469     pagesbefore ,
470     ppbefore ,
471     pagesafter ,
472     ppafter ,
473     before ,
474     after ,
475     thischap ,
476     prevchap ,
477     nextchap ,
478     chapsbefore ,
479     chapsafter ,
480     thissec ,
481     prevsec ,
482     nextsec ,
483     secsbefore ,
484     secsafter ,
485     close ,
486     far ,
487 }
488 {
489   \keys_define:nn { zref-check / zcheck }
490   {
491     #1 .code:n =
492     {
493       \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq {#1}
494       \seq_if_in:NnF \c__zrefcheck_single_label_checks_seq {#1}
495       { \bool_set_true:N \l__zrefcheck_zcheck_end_label_bool }
496     } ,
497     #1 .value_forbidden:n = true ,
498   }
499 }

```

6.3 Running

```

\__zrefcheck_run_checks:nnn   \__zrefcheck_run_checks:nnn {<checks>} {<labels>} {<reference>}
<checks> are expected to be received as a sequence variable.
500 \cs_new_protected:Npn \__zrefcheck_run_checks:nnn #1#2#3
501 {
502   \group_begin:
503   \seq_map_inline:Nn #2
504   {
505     \seq_map_inline:Nn #1
506     { \__zrefcheck_do_check:nnn {####1} {##1} {#3} }
507   }
508   \group_end:
509 }
510 \cs_generate_variant:Nn \__zrefcheck_run_checks:nnn { nnx }

```

(End of definition for _zrefcheck_run_checks:nnn.)

```
\l__zrefcheck_passedcheck_bool
\l__zrefcheck_onpage_bool
\c__zrefcheck_onpage_checks_seq
```

```
511 \bool_new:N \l__zrefcheck_passedcheck_bool
512 \bool_new:N \l__zrefcheck_onpage_bool
513 \seq_const_from_clist:Nn \c__zrefcheck_onpage_checks_seq
514 { above , below , before , after }
```

Variant not provided by expl3.

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

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

```
516 \cs_new_protected:Npn \_zrefcheck_do_check:nnn #1#2#3
517 {
518   \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.

```
519   \zref@ifrefundefined {#2}
520   {}
521   {
522     \tl_if_empty:nF {#1}
523     {
524       \bool_set_true:N \l__zrefcheck_passedcheck_bool
525       \bool_set_false:N \l__zrefcheck_onpage_bool
526       \cs_if_exist:cTF { __zrefcheck_check_ #1 :nnF }
527       {
528         % ‘‘label beg’’ vs ‘‘reference beg’’.
529         \use:c { __zrefcheck_check_ #1 :nnF }
530         {#2} {#3}
531         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
532         % ‘‘reference end’’ \emph{may} exist or not depending on the
533         % checks.
534         \zref@ifrefundefined { \_zrefcheck_end_lblfmt:n {#3} }
535         {
536           % ‘‘label end’’ \emph{may} have been created by the
537           % target commands.
538           \zref@ifrefundefined { \_zrefcheck_end_lblfmt:n {#2} }
539           {}
540           {
541             % ‘‘label end’’ vs ‘‘reference beg’’.
542             \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
543             { \_zrefcheck_end_lblfmt:n {#2} } {#3}
544             { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
545           }
546         }
547         {
548           % ‘‘label beg’’ vs ‘‘reference end’’.
549           \exp_args:Nnno \use:c { __zrefcheck_check_ #1 :nnF }
```

```

550 {#2} { \__zrefcheck_end_lblfmt:n {#3} }
551 { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
552 % ‘‘label end’’ \emph{may} have been created by the
553 % target commands.
554 \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
555 {}
556 {
557 % ‘‘label end’’ vs ‘‘reference beg’’.
558 \exp_args:Nno \use:c { \__zrefcheck_check_ #1 :nnF }
559 { \__zrefcheck_end_lblfmt:n {#2} } {#3}
560 { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
561 % ‘‘label end’’ vs ‘‘reference end’’.
562 \exp_args:Nnoo \use:c { \__zrefcheck_check_ #1 :nnF }
563 { \__zrefcheck_end_lblfmt:n {#2} }
564 { \__zrefcheck_end_lblfmt:n {#3} }
565 { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
566 }
567 }

```

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.

```

568 \seq_if_in:NnT \c__zrefcheck_onpage_checks_seq {#1}
569 {
570   \__zrefcheck_check_thispage:nnT
571   {#2} {#3}
572   { \bool_set_true:N \l__zrefcheck_onpage_bool }
573   \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#3} }
574   {
575     \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
576     {}
577     {
578       \__zrefcheck_check_thispage:nnT
579       { \__zrefcheck_end_lblfmt:n {#2} } {#3}
580       { \bool_set_true:N \l__zrefcheck_onpage_bool }
581     }
582   }
583   {
584     \__zrefcheck_check_thispage:nnT
585     {#2} { \__zrefcheck_end_lblfmt:n {#3} }
586     { \bool_set_true:N \l__zrefcheck_onpage_bool }
587     \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
588     {}
589     {
590       \__zrefcheck_check_thispage:nnT
591       { \__zrefcheck_end_lblfmt:n {#2} } {#3}
592       { \bool_set_true:N \l__zrefcheck_onpage_bool }
593       \__zrefcheck_check_thispage:nnT
594       { \__zrefcheck_end_lblfmt:n {#2} }
595       { \__zrefcheck_end_lblfmt:n {#3} }
596       { \bool_set_true:N \l__zrefcheck_onpage_bool }
597     }
598   }
599 }

```

```

600         \bool_if:NTF \l__zrefcheck_passedcheck_bool
601         {
602             \bool_if:N
603             {
604                 \l__zrefcheck_msgonpage_bool &&
605                 \l__zrefcheck_onpage_bool
606             }
607             {
608                 \__zrefcheck_message:nnx { double-check } {#1} {#2}
609                 { \zref@extractdefault {#3} {page} {'unknown'} }
610             }
611         }
612         {
613             \__zrefcheck_message:nnx { check-failed } {#1} {#2}
614             { \zref@extractdefault {#3} {page} {'unknown'} }
615         }
616     }
617     { \msg_warning:nnn { zref-check } { check-missing } {#1} }
618 }
619 }
620 \group_end:
621 }
622 \cs_generate_variant:Nn \__zrefcheck_do_check:nnn { nnV }

```

(End of 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.

```

623 \int_new:N \l__zrefcheck_lbl_int
624 \int_new:N \l__zrefcheck_ref_int
625 \int_new:N \l__zrefcheck_lbl_b_int
626 \int_new:N \l__zrefcheck_ref_b_int

```

6.4.1 This page

```

\__zrefcheck_check_thispage:nn
\__zrefcheck_check_otherpage:nn

```

```

627 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thispage:nn #1#2 { T , 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         { \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.

```

638         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
639         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
640     }
641     { \group_insert_after:N \prg_return_true: }
642     { \group_insert_after:N \prg_return_false: }
643 \group_end:
644 }
645 \prg_new_protected_conditional:Npnn \__zrefcheck_check_otherpage:nn #1#2 { T , F , TF }
646 {
647     \__zrefcheck_check_thispage:nnTF {#1} {#2}
648     { \prg_return_false: }
649     { \prg_return_true: }
650 }

```

(End of definition for __zrefcheck_check_thispage:nn and __zrefcheck_check_otherpage:nn.)

6.4.2 On page

__zrefcheck_check_above:nn
__zrefcheck_check_below:nn

```

651 \prg_new_protected_conditional:Npnn \__zrefcheck_check_above:nn #1#2 { F , TF }
652 {
653     \group_begin:
654     \__zrefcheck_check_thispage:nnTF {#1} {#2}
655     {
656         \bool_set_true:N \l__zrefcheck_integer_bool
657         \zrefcheck_get_asint:nnn {#1} { lblseq } { \l__zrefcheck_lbl_int }
658         \zrefcheck_get_asint:nnn {#2} { lblseq } { \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 } &&
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     }
670     { \group_insert_after:N \prg_return_false: }
671 \group_end:
672 }
673 \prg_new_protected_conditional:Npnn \__zrefcheck_check_below:nn #1#2 { F , TF }
674 {
675     \__zrefcheck_check_thispage:nnTF {#1} {#2}
676     {
677         \__zrefcheck_check_above:nnTF {#1} {#2}
678         { \prg_return_false: }
679         { \prg_return_true: }
680     }
681     { \prg_return_false: }
682 }

```

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

6.4.3 Before / After

```

\__zrefcheck_check_before:nn
\__zrefcheck_check_after:nn
683 \prg_new_protected_conditional:Npnn \__zrefcheck_check_before:nn #1#2 { F }
684 {
685   \__zrefcheck_check_pagesbefore:nnTF {#1} {#2}
686   { \prg_return_true: }
687   {
688     \__zrefcheck_check_above:nnTF {#1} {#2}
689     { \prg_return_true: }
690     { \prg_return_false: }
691   }
692 }
693 \prg_new_protected_conditional:Npnn \__zrefcheck_check_after:nn #1#2 { F }
694 {
695   \__zrefcheck_check_pagesafter:nnTF {#1} {#2}
696   { \prg_return_true: }
697   {
698     \__zrefcheck_check_below:nnTF {#1} {#2}
699     { \prg_return_true: }
700     { \prg_return_false: }
701   }
702 }

```

(End of 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
703 \prg_new_protected_conditional:Npnn \__zrefcheck_check_nextpage:nn #1#2 { F }
704 {
705   \group_begin:
706   \bool_set_true:N \l__zrefcheck_integer_bool
707   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
708   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
709   \bool_lazy_and:nnTF
710   { \l__zrefcheck_integer_bool }
711   {
712     \int_compare_p:nNn
713     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
714     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
715     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
716   }
717   { \group_insert_after:N \prg_return_true: }
718   { \group_insert_after:N \prg_return_false: }
719   \group_end:
720 }
721 \prg_new_protected_conditional:Npnn \__zrefcheck_check_prevpage:nn #1#2 { F }
722 {
723   \group_begin:
724   \bool_set_true:N \l__zrefcheck_integer_bool
725   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
726   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
727   \bool_lazy_and:nnTF

```



```

728     { \l__zrefcheck_integer_bool }
729     {
730         \int_compare_p:nNn
731         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
732         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
733         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
734     }
735     { \group_insert_after:N \prg_return_true: }
736     { \group_insert_after:N \prg_return_false: }
737 \group_end:
738 }
739 \prg_new_protected_conditional:Npnn \__zrefcheck_check_pagesbefore:nn #1#2 { F , TF }
740 {
741     \group_begin:
742     \bool_set_true:N \l__zrefcheck_integer_bool
743     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
744     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
745     \bool_lazy_and:nnTF
746     { \l__zrefcheck_integer_bool }
747     {
748         \int_compare_p:nNn
749         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
750         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
751         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
752     }
753     { \group_insert_after:N \prg_return_true: }
754     { \group_insert_after:N \prg_return_false: }
755 \group_end:
756 }
757 \cs_new_eq:NN \__zrefcheck_check_ppbefore:nnF \__zrefcheck_check_pagesbefore:nnF
758 \prg_new_protected_conditional:Npnn \__zrefcheck_check_pagesafter:nn #1#2 { F , TF }
759 {
760     \group_begin:
761     \bool_set_true:N \l__zrefcheck_integer_bool
762     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
763     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
764     \bool_lazy_and:nnTF
765     { \l__zrefcheck_integer_bool }
766     {
767         \int_compare_p:nNn
768         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
769         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
770         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
771     }
772     { \group_insert_after:N \prg_return_true: }
773     { \group_insert_after:N \prg_return_false: }
774 \group_end:
775 }
776 \cs_new_eq:NN \__zrefcheck_check_ppafter:nnF \__zrefcheck_check_pagesafter:nnF
777 \prg_new_protected_conditional:Npnn \__zrefcheck_check_pagegap:nn #1#2 { F }
778 {
779     \group_begin:
780     \bool_set_true:N \l__zrefcheck_integer_bool
781     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }

```

```

782 \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
783 \bool_lazy_and:nnTF
784 { \l__zrefcheck_integer_bool }
785 {
786   \int_compare_p:nNn
787     { \int_abs:n { \l__zrefcheck_lbl_int - \l__zrefcheck_ref_int } } > { 1 } &&
788     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
789     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
790   }
791   { \group_insert_after:N \prg_return_true: }
792   { \group_insert_after:N \prg_return_false: }
793 \group_end:
794 }
795 \prg_new_protected_conditional:Npnn \__zrefcheck_check_facing:nn #1#2 { F }
796 {
797   \group_begin:
798   \bool_set_true:N \l__zrefcheck_integer_bool
799   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
800   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
801   \bool_lazy_and:nnTF
802   { \l__zrefcheck_integer_bool }
803   {

```

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

```

804 \legacy_if_p:n { @twoside } &&

```

Now we test “facing”.

```

805 (
806   (
807     \int_if_odd_p:n { \l__zrefcheck_ref_int } &&
808     \int_compare_p:nNn
809       { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 }
810   ) ||
811   (
812     \int_if_even_p:n { \l__zrefcheck_ref_int } &&
813     \int_compare_p:nNn
814       { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 }
815   )
816 ) &&
817 ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
818 ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
819 }
820 { \group_insert_after:N \prg_return_true: }
821 { \group_insert_after:N \prg_return_false: }
822 \group_end:
823 }

```

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

6.4.5 Close / Far

```

\__zrefcheck_check_close:nn
\__zrefcheck_check_far:nn
824 \prg_new_protected_conditional:Npnn \__zrefcheck_check_close:nn #1#2 { F , TF }
825 {
826   \group_begin:

```

```

827 \bool_set_true:N \l__zrefcheck_integer_bool
828 \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
829 \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
830 \bool_lazy_and:nnTF
831 { \l__zrefcheck_integer_bool }
832 {
833   \int_compare_p:nNn
834     { \int_abs:n { \l__zrefcheck_lbl_int - \l__zrefcheck_ref_int } }
835     <
836     { \l__zrefcheck_close_range_int + 1 } &&
837     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
838     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
839   }
840   { \group_insert_after:N \prg_return_true: }
841   { \group_insert_after:N \prg_return_false: }
842 \group_end:
843 }
844 \prg_new_protected_conditional:Npnn \__zrefcheck_check_far:nn #1#2 { F }
845 {
846   \__zrefcheck_check_close:nnTF {#1} {#2}
847   { \prg_return_false: }
848   { \prg_return_true: }
849 }

```

(End of 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
850 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thischap:nn #1#2 { F }
851 {
852   \group_begin:
853     \bool_set_true:N \l__zrefcheck_integer_bool
854     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
855     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
856     \bool_lazy_and:nnTF
857     { \l__zrefcheck_integer_bool }
858     {
859       \int_compare_p:nNn
860         { \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.

```

861       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
862       ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
863     }
864     { \group_insert_after:N \prg_return_true: }
865     { \group_insert_after:N \prg_return_false: }
866   \group_end:
867 }
868 \prg_new_protected_conditional:Npnn \__zrefcheck_check_nextchap:nn #1#2 { F }

```

```

869 {
870   \group_begin:
871     \bool_set_true:N \l__zrefcheck_integer_bool
872     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
873     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
874     \bool_lazy_and:nnTF
875       { \l__zrefcheck_integer_bool }
876       {
877         \int_compare_p:nNn
878           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
879           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
880       }
881       { \group_insert_after:N \prg_return_true: }
882       { \group_insert_after:N \prg_return_false: }
883   \group_end:
884 }
885 \prg_new_protected_conditional:Npnn \__zrefcheck_check_prevchap:nn #1#2 { F }
886 {
887   \group_begin:
888     \bool_set_true:N \l__zrefcheck_integer_bool
889     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
890     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
891     \bool_lazy_and:nnTF
892       { \l__zrefcheck_integer_bool }
893       {
894         \int_compare_p:nNn
895           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
896           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
897           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
898       }
899       { \group_insert_after:N \prg_return_true: }
900       { \group_insert_after:N \prg_return_false: }
901   \group_end:
902 }
903 \prg_new_protected_conditional:Npnn \__zrefcheck_check_chapsafter:nn #1#2 { F }
904 {
905   \group_begin:
906     \bool_set_true:N \l__zrefcheck_integer_bool
907     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
908     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
909     \bool_lazy_and:nnTF
910       { \l__zrefcheck_integer_bool }
911       {
912         \int_compare_p:nNn
913           { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
914           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
915       }
916       { \group_insert_after:N \prg_return_true: }
917       { \group_insert_after:N \prg_return_false: }
918   \group_end:
919 }
920 \prg_new_protected_conditional:Npnn \__zrefcheck_check_chapsbefore:nn #1#2 { F }
921 {
922   \group_begin:

```

```

923 \bool_set_true:N \l__zrefcheck_integer_bool
924 \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
925 \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
926 \bool_lazy_and:nnTF
927 { \l__zrefcheck_integer_bool }
928 {
929   \int_compare_p:nNn
930     { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
931     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
932     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
933   }
934   { \group_insert_after:N \prg_return_true: }
935   { \group_insert_after:N \prg_return_false: }
936 \group_end:
937 }

```

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

```

‘0’ is the default value of `zc@abssec` property, and means here no `\section` has yet been issued since its counter has been reset, which occurs at the beginning of the document and at every chapter. Hence, as is the case for chapters, ‘0’ is just “not a section”. The same observation about the need of the “current” section to exist to be able to refer to a “future” one also holds. This comment extends to all section checks.

```

953       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
954       ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
955     }
956     { \group_insert_after:N \prg_return_true: }
957     { \group_insert_after:N \prg_return_false: }
958   \group_end:
959 }
960 \prg_new_protected_conditional:Npnn \__zrefcheck_check_nextsec:nn #1#2 { F }
961 {
962   \group_begin:
963     \bool_set_true:N \l__zrefcheck_integer_bool
964     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }

```

```

965 \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
966 \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
967 \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
968 \bool_lazy_and:nnTF
969 { \l__zrefcheck_integer_bool }
970 {
971   \int_compare_p:nNn
972     { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
973   \int_compare_p:nNn
974     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
975   ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
976 }
977 { \group_insert_after:N \prg_return_true: }
978 { \group_insert_after:N \prg_return_false: }
979 \group_end:
980 }
981 \prg_new_protected_conditional:Npnn \__zrefcheck_check_prevsec:nn #1#2 { F }
982 {
983   \group_begin:
984     \bool_set_true:N \l__zrefcheck_integer_bool
985     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
986     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
987     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
988     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
989     \bool_lazy_and:nnTF
990     { \l__zrefcheck_integer_bool }
991     {
992       \int_compare_p:nNn
993         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
994       \int_compare_p:nNn
995         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
996       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
997       ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
998     }
999     { \group_insert_after:N \prg_return_true: }
1000     { \group_insert_after:N \prg_return_false: }
1001   \group_end:
1002 }
1003 \prg_new_protected_conditional:Npnn \__zrefcheck_check_secsafter:nn #1#2 { F }
1004 {
1005   \group_begin:
1006     \bool_set_true:N \l__zrefcheck_integer_bool
1007     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
1008     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
1009     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
1010     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
1011     \bool_lazy_and:nnTF
1012     { \l__zrefcheck_integer_bool }
1013     {
1014       \int_compare_p:nNn
1015         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
1016       \int_compare_p:nNn
1017         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
1018       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }

```

```

1019     }
1020     { \group_insert_after:N \prg_return_true: }
1021     { \group_insert_after:N \prg_return_false: }
1022   \group_end:
1023 }
1024 \prg_new_protected_conditional:Npnn \__zrefcheck_check_secsbefore:nn #1#2 { F }
1025 {
1026   \group_begin:
1027     \bool_set_true:N \l__zrefcheck_integer_bool
1028     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
1029     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
1030     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
1031     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
1032     \bool_lazy_and:nnTF
1033       { \l__zrefcheck_integer_bool }
1034       {
1035         \int_compare_p:nNn
1036           { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
1037         \int_compare_p:nNn
1038           { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
1039         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
1040         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
1041       }
1042     { \group_insert_after:N \prg_return_true: }
1043     { \group_insert_after:N \prg_return_false: }
1044   \group_end:
1045 }

```

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

7 zref-clever integration

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

```

\zrefcheck_zceref_beg_label:
  \zrefcheck_zceref_end_label_maybe:
  \zrefcheck_zceref_run_checks_on_labels:n
1046 \cs_new_protected:Npn \zrefcheck_zceref_beg_label:
1047 {
1048   \int_gincr:N \g__zrefcheck_id_int
1049   \tl_set:Nx \l__zrefcheck_checkbeg_tl
1050     { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
1051   \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrefcheck-check }
1052 }
1053 \cs_new_protected:Npn \zrefcheck_zceref_end_label_maybe:
1054 {
1055   \bool_if:NT \l__zrefcheck_zcheck_end_label_bool
1056   {
1057     \zref@labelbylist
1058       { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
1059       { zrefcheck-end }

```

```

1060     }
1061   }
1062   \cs_new_protected:Npn \zrefcheck_zcref_run_checks_on_labels:n #1
1063   {
1064     \__zrefcheck_run_checks:nnx
1065     { \l__zrefcheck_zcheck_checks_seq } {#1} { \l__zrefcheck_checkbeg_tl }
1066   }

```

(End of 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 ??.)

8 zref-vario integration

```

\zrefcheck_zrefvario_label:
\zrefcheck_zrefvario_run_check_on_label:n
1067 \cs_new_protected:Npn \zrefcheck_zrefvario_label:
1068 {
1069   \int_gincr:N \g__zrefcheck_id_int
1070   \tl_set:Nx \l__zrefcheck_checkbeg_tl
1071   { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
1072   \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrefcheck-zrefvario }
1073 }
1074 \cs_new_protected:Npn \zrefcheck_zrefvario_run_check_on_label:nn #1#2
1075 { \__zrefcheck_do_check:nnV {#1} {#2} \l__zrefcheck_checkbeg_tl }
1076 \cs_generate_variant:Nn \zrefcheck_zrefvario_run_check_on_label:nn { Vn }

```

(End of definition for \zrefcheck_zrefvario_label: and \zrefcheck_zrefvario_run_check_on_label:n. These functions are documented on page ??.)

```

1077 \</package>

```

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols	
\	78, 109
A	
\A	124
\AddToHook	22, 26, 151, 209
B	
\begingroup	318
bool commands:	
\bool_if:NTF	155, 159, 302, 410, 600, 1055
\bool_if:nTF	397, 602
\bool_lazy_and:nnTF	14, 277, 633, 659, 709, 727, 745, 764, 783, 801, 830, 856, 874, 891, 909, 926, 946, 968, 989, 1011, 1032
C	
\catcode	11
\chapter	2, 27
clist commands:	
\clist_map_inline:nn	459
\bool_new:N	128, 129, 214, 356, 382, 457, 511, 512
\bool_set:Nn	393
\bool_set_false:N	136, 145, 146, 161, 220, 229, 236, 292, 304, 366, 525, 531, 544, 551, 560, 565
\bool_set_true:N	135, 140, 141, 224, 230, 235, 299, 495, 524, 572, 580, 586, 592, 596, 630, 656, 706, 724, 742, 761, 780, 798, 827, 853, 871, 888, 906, 923, 941, 963, 984, 1006, 1027
\l_tmpa_bool	12, 292, 299, 302, 304

cs commands:		\int_compare_p:nNn	
\cs_generate_variant:Nn	636, 638, 639, 662,
.....	56, 510, 515, 622, 1076	664, 665, 712, 714, 715, 730, 732,	
\cs_if_exist:NTF	526	733, 748, 750, 751, 767, 769, 770,	
\cs_new:Npn	323, 324	786, 788, 789, 808, 813, 817, 818,	
\cs_new_eq:NN	111, 757, 776	833, 837, 838, 859, 861, 862, 877,	
\cs_new_protected:Npn		879, 894, 896, 897, 912, 914, 929,	
.....	51, 325, 358, 383,	931, 932, 949, 951, 953, 954, 971,	
500, 516, 1046, 1053, 1062, 1067, 1074		973, 975, 992, 994, 996, 997, 1014,	
		1016, 1018, 1035, 1037, 1039, 1040	
D		\int_eval:n	259, 363
\d	124	\int_gincr:N	388, 1048, 1069
E		\int_if_even_p:n	812
\emph	532, 536, 552	\int_if_odd_p:n	807
\endgroup	321	\int_incr:N	305
\endinput	13	\int_new:N .	253, 378, 623, 624, 625, 626
exp commands:		\int_set:Nn	259, 262, 363
\exp_args:Nnno	515, 549	\int_use:N	24, 28, 307, 323
\exp_args:Nno	542, 558	\int_zero:N	289
\exp_args:Noo	562	\l_tmpa_int	289, 305, 307
\exp_args:Nx	403	int internal commands:	
F		__int_to_roman:w	
file commands:		ior commands:	
\file_if_exist:nTF	285	\ior_close:N	316
\fmtversion	4	\ior_map_variable:NNn	293
G		\ior_open:Nn	287
group commands:		\g_tmpa_ior	287, 293, 316
\group_begin:	288, 385,	iow commands:	
423, 502, 518, 629, 653, 705, 723,		\iow_char:N	78, 109
741, 760, 779, 797, 826, 852, 870,		\iow_newline:	77, 81, 84, 101
887, 905, 922, 940, 962, 983, 1005, 1026		K	
\group_end:	315, 419, 432,	keys commands:	
508, 620, 643, 671, 719, 737, 755,		\keys_define:nn	130,
774, 793, 822, 842, 866, 883, 901,		163, 170, 211, 215, 254, 268, 458, 489	
918, 936, 958, 979, 1001, 1022, 1044		\keys_set:nn	282, 386
\group_insert_after:N		L	
.....	641, 642, 667, 668,	\label	12
670, 717, 718, 735, 736, 753, 754,		legacy commands:	
772, 773, 791, 792, 820, 821, 840,		\legacy_if_p:n	804
841, 864, 865, 881, 882, 899, 900,		\let	319
916, 917, 934, 935, 956, 957, 977,		M	
978, 999, 1000, 1020, 1021, 1042, 1043		\MessageBreak	11
H		msg commands:	
\hyperlink	15, 406	\msg_line_context:	59,
I		64, 68, 70, 72, 74, 88, 91, 101, 105	
\ifdraft	181, 228	\msg_new:nnn . . .	57, 62, 67, 69, 71,
\IfFormatAtLeastTF	4, 5	73, 75, 80, 82, 87, 89, 94, 99, 104, 106	
\ifoptionfinal	187, 234	\msg_warning:nn	160, 166, 206, 261, 280
int commands:		\msg_warning:nnn	273, 617
\int_abs:n	787, 834	\msg_warning:nnnn	
		194, 200, 241, 247, 332, 341, 349, 370	

N		T	
<code>\NeedsTeXFormat</code>	3	TeX and L ^A T _E X 2 _ε commands:	
<code>\newcounter</code>	20, 21, 320	<code>\@addtoreset</code>	319
<code>\NewDocumentCommand</code>	281, 375, 421	<code>\@currentlabel</code>	17
<code>\NewDocumentEnvironment</code>	434	<code>\@ifl@t@r</code>	4
P		<code>\@ifpackageloaded</code>	153
<code>\PackageError</code>	8	<code>\@newl@bel</code>	10
<code>\pageref</code>	12	<code>\c@z@abschap</code>	24
prg commands:		<code>\c@z@abssec</code>	28
<code>\prg_new_conditional:Npnn</code>	112	<code>\ltx@gobbletwo</code>	319
<code>\prg_new_protected_conditional:Npnn</code>	17, 122, 627, 645, 651, 673, 683, 693, 703, 721, 739, 758, 777, 795, 824, 844, 850, 868, 885, 903, 920, 938, 960, 981, 1003, 1024	<code>\zref@addprop</code>	18, 25, 29
<code>\prg_return_false:</code>	17, 115, 119, 126, 642, 648, 668, 670, 678, 681, 690, 700, 718, 736, 754, 773, 792, 821, 841, 847, 865, 882, 900, 917, 935, 957, 978, 1000, 1021, 1043	<code>\zref@addprops</code>	31, 39, 46
<code>\prg_return_true:</code>	17, 118, 125, 641, 649, 667, 679, 686, 689, 696, 699, 717, 735, 753, 772, 791, 820, 840, 848, 864, 881, 899, 916, 934, 956, 977, 999, 1020, 1042	<code>\zref@extractdefault</code>	13, 346, 609, 614
prop commands:		<code>\zref@ifpropundefined</code>	340
<code>\prop_get:NnNTF</code>	330	<code>\zref@ifrefcontainsprop</code>	343
<code>\prop_gput:Nnn</code>	306	<code>\zref@ifrefundefined</code>	13, 337, 367, 394, 519, 534, 538, 554, 573, 575, 587
<code>\prop_new:N</code>	283	<code>\ZREF@label</code>	12
<code>\providecommand</code>	4	<code>\zref@label</code>	11, 12, 425, 437
<code>\ProvidesExplPackage</code>	15	<code>\zref@labelbylist</code>	391, 412, 430, 441, 1051, 1057, 1072
R		<code>\ZREF@mainlist</code>	25, 29
<code>\refstepcounter</code>	12, 17, 424, 436	<code>\zref@newlabel</code>	10–12, 297
regex commands:		<code>\zref@newlist</code>	30, 38, 45
<code>\regex_match:nnTF</code>	124	<code>\zref@newprop</code>	18, 24, 28
<code>\RequirePackage</code>	17, 18, 19, 156	<code>\zref@refused</code>	13, 416
<code>\romannumeral</code>	5	<code>\zref@require@unique</code>	12
S		<code>\zref@wrapper@babel</code>	10, 15, 376, 425, 429, 437, 440
<code>\section</code>	29	tl commands:	
seq commands:		<code>\c_empty_tl</code>	13, 346
<code>\seq_const_from_clist:Nn</code> ...	443, 513	<code>\tl_clear:N</code>	13, 290, 291, 327
<code>\seq_get:NN</code>	392	<code>\tl_gset:Nn</code>	284
<code>\seq_if_in:NnTF</code>	494, 568	<code>\tl_if_blank:nTF</code>	5
<code>\seq_map_function:NN</code>	416	<code>\tl_if_empty:nTF</code>	5, 114, 117, 427, 522
<code>\seq_map_inline:Nn</code>	503, 505	<code>\tl_if_empty_p:N</code>	279
<code>\seq_new:N</code>	377, 456	<code>\tl_if_eq:NnTF</code>	297
<code>\seq_put_right:Nn</code>	493	<code>\tl_if_eq:nnTF</code>	328
<code>\seq_set_from_clist:Nn</code>	387	<code>\tl_if_exist_p:N</code>	278
<code>\setcounter</code>	322	<code>\tl_map_break:</code>	310
<code>\stepcounter</code>	23, 27	<code>\tl_map_variable:NNn</code>	295
sys commands:		<code>\tl_new:N</code>	169, 357, 379, 380, 381
<code>\c_sys_jobname_str</code>	284	<code>\tl_set:Nn</code>	174, 176, 178, 182, 183, 188, 189, 345, 389, 1049, 1070
		<code>\g_tmpa_tl</code>	284, 285, 287
		<code>\l_tmpa_tl</code>	290, 293, 295
		<code>\l_tmpb_tl</code>	291, 295, 297, 307
		U	
		use commands:	
		<code>\use:N</code>	53, 529, 542, 549, 558, 562
		Z	
		<code>\Z</code>	124
		<code>\zcheck</code>	15, 18, 375

<code>\zceref</code>	31	<code>__zrefcheck_check_nextsec:nn</code> ...	
<code>zcregion</code>	434	938 , 960
<code>\zctarget</code>	17 , 421	<code>__zrefcheck_check_otherpage:nn</code> ..	
<code>\zref</code>	15	627 , 645
zrefcheck commands:			
<code>\zrefcheck_get_asint:nnn</code>		<code>__zrefcheck_check_pagegap:nn</code> ...	
.....	14 , 18 , 358 , 358 ,	703 , 777
.....	631 , 632 , 657 , 658 , 707 , 708 , 725 ,	<code>__zrefcheck_check_pagesafter:nn</code>	
.....	726 , 743 , 744 , 762 , 763 , 781 , 782 ,	703 , 758
.....	799 , 800 , 828 , 829 , 854 , 855 , 872 ,	<code>__zrefcheck_check_pagesafter:nnTF</code>	
.....	873 , 889 , 890 , 907 , 908 , 924 , 925 ,	695 , 776
.....	942 , 943 , 944 , 945 , 964 , 965 , 966 ,	<code>__zrefcheck_check_pagesbefore:nn</code>	
.....	967 , 985 , 986 , 987 , 988 , 1007 , 1008 ,	703 , 739
.....	1009 , 1010 , 1028 , 1029 , 1030 , 1031	<code>__zrefcheck_check_pagesbefore:nnTF</code>	
<code>\zrefcheck_get_astl:nnn</code>	685 , 757
.....	13 , 14 , 18 , 325 , 325 , 360 , 403	<code>__zrefcheck_check_ppafter:nn</code> ..	703
<code>\zrefcheck_zceref_beg_label:</code>		<code>__zrefcheck_check_ppafter:nnTF</code>	776
.....	1046 , 1046	<code>__zrefcheck_check_ppbefore:nn</code> ..	703
<code>\zrefcheck_zceref_end_label-</code>		<code>__zrefcheck_check_ppbefore:nnTF</code>	757
maybe:	1046 , 1053	<code>__zrefcheck_check_prevchap:nn</code> ..	
<code>\zrefcheck_zceref_run_checks_on-</code>		850 , 885
labels:n	1046 , 1062	<code>__zrefcheck_check_prevpage:nn</code> ..	
<code>\zrefcheck_zrefvario_label:</code>	703 , 721
.....	1067 , 1067	<code>__zrefcheck_check_prevsec:nn</code> ...	
<code>\zrefcheck_zrefvario_run_check-</code>		938 , 981
on_label:n	1067	<code>__zrefcheck_check_secsafter:nn</code> ..	
<code>\zrefcheck_zrefvario_run_check-</code>		938 , 1003
on_label:nn	1074 , 1076	<code>__zrefcheck_check_secsbefore:nn</code>	
zrefcheck internal commands:			
<code>\g__zrefcheck_auxfile_lblseq-</code>		938 , 1024
prop	11 , 13 , 18 , 283 , 306 , 330	<code>__zrefcheck_check_thischap:nn</code> ..	
<code>__zrefcheck_check_⟨check⟩:nn</code> ...	17	850 , 850
<code>__zrefcheck_check_above:nn</code> 651 , 651		<code>__zrefcheck_check_thispage:nn</code> ..	
<code>__zrefcheck_check_above:nnTF</code>	627 , 627
.....	677 , 688	<code>__zrefcheck_check_thispage:nnTF</code>	
<code>__zrefcheck_check_after:nn</code> 683 , 693		570 , 578 , 584 , 590 , 593 , 647 , 654 , 675
<code>__zrefcheck_check_before:nn</code> 683 , 683		<code>__zrefcheck_check_thissec:nn</code> ...	
<code>__zrefcheck_check_below:nn</code> 651 , 673		938 , 938
<code>__zrefcheck_check_below:nnTF</code> ..	698	<code>\l__zrefcheck_checkbeg_tl</code>	
<code>__zrefcheck_check_chapsafter:nn</code>		15 , 379 , 389 , 391 , 413 , 418 , 1049 ,
.....	850 , 903	1051 , 1058 , 1065 , 1070 , 1072 , 1075
<code>__zrefcheck_check_chapsbefore:nn</code>		<code>\l__zrefcheck_close_range_int</code> ...	
.....	850 , 920	9 , 253 , 259 , 262 , 836
<code>__zrefcheck_check_close:nn</code> 824 , 824		<code>__zrefcheck_do_check:nnn</code>	
<code>__zrefcheck_check_close:nnTF</code> ..	846	20 , 506 , 516 , 516 , 622 , 1075
<code>__zrefcheck_check_facing:nn</code> 703 , 795		<code>\l__zrefcheck_end_label_required-</code>	
<code>__zrefcheck_check_far:nn</code> ..	824 , 844	bool	18
<code>__zrefcheck_check_lblfmt:n</code>		<code>__zrefcheck_end_lblfmt:n</code> ...	12 ,
.....	12 , 323 , 323 , 390 , 1050 , 1071	324 , 324 , 413 , 430 , 441 , 534 , 538 ,
<code>__zrefcheck_check_nextchap:nn</code>	543 , 550 , 554 , 559 , 563 , 564 , 573 ,
.....	850 , 868	575 , 579 , 585 , 587 , 591 , 594 , 595 , 1058
<code>__zrefcheck_check_nextpage:nn</code> ..		<code>__zrefcheck_get_asint:nnn</code>	13
.....	703 , 703	<code>__zrefcheck_get_astl:nnn</code>	13
		<code>\g__zrefcheck_id_int</code>	15 ,
		378 , 388 , 390 , 1048 , 1050 , 1069 , 1071

_zrefcheck_int_to_roman:w	\l_zrefcheck_onpage_bool . . . 20,
. 111, 111, 117	512, 525, 572, 580, 586, 592, 596, 605
\l_zrefcheck_integer_bool	\c_zrefcheck_onpage_checks_seq .
. 14, 356, 366, 630, 634, 656, 20, 513, 568
660, 706, 710, 724, 728, 742, 746,	\l_zrefcheck_passedcheck_bool . .
761, 765, 780, 784, 798, 802, 827, 20,
831, 853, 857, 871, 875, 888, 892,	511, 524, 531, 544, 551, 560, 565, 600
906, 910, 923, 927, 941, 947, 963,	\l_zrefcheck_propval_tl
969, 984, 990, 1006, 1012, 1027, 1033 14, 357, 360, 361, 363
_zrefcheck_is_integer:n 5, 111, 112	\l_zrefcheck_ref_b_int
_zrefcheck_is_integer:nTF . . . 361 22, 626, 945, 950, 967,
_zrefcheck_is_integer_rgx:n . . .	972, 988, 993, 1010, 1015, 1031, 1036
. 5, 122, 122	\l_zrefcheck_ref_int . . . 22, 624,
_zrefcheck_is_integer_rgx:nTF 258	632, 637, 639, 658, 663, 665, 708,
\l_zrefcheck_lbl_b_int	713, 715, 726, 731, 733, 744, 749,
. 22, 625, 944, 950, 966,	751, 763, 768, 770, 782, 787, 789,
972, 987, 993, 1009, 1015, 1030, 1036	800, 807, 809, 812, 814, 818, 829,
\l_zrefcheck_lbl_int	834, 838, 855, 860, 862, 873, 878,
. 22, 623, 631, 637,	890, 895, 897, 908, 913, 925, 930,
638, 657, 663, 664, 707, 713, 714,	932, 943, 952, 954, 965, 974, 986,
725, 731, 732, 743, 749, 750, 762,	995, 997, 1008, 1017, 1029, 1038, 1040
768, 769, 781, 787, 788, 799, 809,	_zrefcheck_run_checks:nnn
814, 817, 828, 834, 837, 854, 860, 19, 417, 500, 500, 510, 1064
861, 872, 878, 879, 889, 895, 896,	\c_zrefcheck_single_label_-
907, 913, 914, 924, 930, 931, 942,	checks_seq 18, 443, 494
952, 953, 964, 974, 975, 985, 995,	\l_zrefcheck_use_hyperref_bool .
996, 1007, 1017, 1018, 1028, 1038, 1039	. . . 6, 128, 135, 140, 145, 155, 161, 399
\l_zrefcheck_link_anchor_tl . . .	\l_zrefcheck_warn_hyperref_bool
. 15, 381, 405, 406 6, 129, 136, 141, 146, 159
\l_zrefcheck_link_label_tl	_zrefcheck_zcheck:nnnn
. 15, 380, 392, 394, 404 15, 376, 383, 383
\l_zrefcheck_link_star_bool . . .	_zrefcheck_zcheck:nnnnn 13, 14, 20
. 15, 382, 393, 400	\l_zrefcheck_zcheck_checks_seq .
_zrefcheck_message:nnnn 18, 417, 456, 493, 1065
. 51, 51, 56, 608, 613	\l_zrefcheck_zcheck_end_label_-
\l_zrefcheck_msglevel_tl . 7, 53,	bool 410, 457, 495, 1055
169, 174, 176, 178, 182, 183, 188, 189	\l_zrefcheck_zcheck_labels_seq .
\l_zrefcheck_msgonpage_bool . 8, 15, 377, 387, 392, 416, 418
214, 220, 224, 229, 230, 235, 236, 604	\zrefchecksetup 9, 281