

The zref-check package implementation*

Gustavo Barros[†]

2023-08-13

Contents

1	Initial setup	2
2	Dependencies	2
3	zref setup	2
4	Plumbing	3
	4.1 Messages	3
	4.2 Integer testing	5
	4.3 Options	6
	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	17
	6.1 Single label checks	17
	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	23
	6.4.4 Pages	24
	6.4.5 Close / Far	26
	6.4.6 Chapter	27
	6.4.7 Section	29
	6.4.8 Manual	31
7	zref-clever integration	31

*This file describes v0.3.4, released 2023-08-13.

[†]<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 ltcmd-hooks, which was released with the 2021/06/01 L^AT_EX kernel.

```
3 \def\zrefcheck@required@kernel{2021-06-01}
4 \NeedsTeXFormat{LaTeX2e}[\zrefcheck@required@kernel]
5 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
6 \IfFormatAtLeastTF{\zrefcheck@required@kernel}
7   {}
8   {%
9     \PackageError{zref-check}{LaTeX kernel too old}
10    {%
11      'zref-check' requires a LaTeX kernel \zrefcheck@required@kernel\space or newer.%
12    }%
13  }%
```

Identify the package.

```
14 \ProvidesExplPackage {zref-check} {2023-08-13} {0.3.4}
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

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.

```
19 \newcounter { zc@abschap }
20 \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.

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

```

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

```

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

```

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

```

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

```

4 Plumbing

4.1 Messages

```

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

```

(End of definition for __zrefcheck_message:nnnn.)

```

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

```

```

62 {
63   Same-page-check~\msg_line_context:..~
64   Double-check~'#1'~for~label~'#2'~on~page~#3.
65 }
66 \msg_new:nnn { zref-check } { empty-label }
67 {
68   Check~failed~\msg_line_context:..~
69   Failed-check~'#1'~for~empty~label.
70 }
71 \msg_new:nnn { zref-check } { no-checks }
72 { No~checks~for~'\iow_char:N\zcheck'~\msg_line_context:. }
73 \msg_new:nnn { zref-check } { check-missing }
74 { Check~'#1'~not~defined~\msg_line_context:. }
75 \msg_new:nnn { zref-check } { property-undefined }
76 { Property~'#1'~not~defined~\msg_line_context:. }
77 \msg_new:nnn { zref-check } { property-not-in-label }
78 { Label~'#1'~has~no~property~'#2'~\msg_line_context:. }
79 \msg_new:nnn { zref-check } { property-not-integer }
80 { Property~'#1'~for~label~'#2'~not~an~integer~\msg_line_context:. }
81 \msg_new:nnn { zref-check } { hyperref-preamble-only }
82 {
83   Option~'hyperref'~only~available~in~the~preamble. \iow_newline:
84   Use~the~starred~version~of~'\iow_char:N\zcheck'~instead.
85 }
86 \msg_new:nnn { zref-check } { missing-hyperref }
87 { Missing~'hyperref'~package. \iow_newline: Setting~'hyperref=false'. }
88 \msg_new:nnn { zref-check } { ignore-ok-document-only }
89 {
90   Option~'#1'~only~available~in~the~document. \iow_newline:
91   Use~option~'msglevel'~instead.
92 }
93 \msg_new:nnn { zref-check } { option-preamble-only }
94 { Option~'#1'~is~preamble~only~\msg_line_context:. }
95 \msg_new:nnn { zref-check } { closerange-not-positive-integer }
96 {
97   Option~'closerange'~not~a~positive~integer~\msg_line_context:..~
98   Using~default~value.
99 }
100 \msg_new:nnn { zref-check } { labelcmd-undefined }
101 {
102   Control~sequence~named~'#1'~used~in~option~'labelcmd'~is~not~defined.~
103   Using~default~value.
104 }
105 \msg_new:nnn { zref-check } { option-deprecated-with-alternative }
106 {
107   Option~'#1'~has~been~deprecated~\msg_line_context:.\iow_newline:
108   Use~'#2'~instead.
109 }
110 \msg_new:nnn { zref-check } { option-deprecated }
111 { Option~'#1'~has~been~deprecated~\msg_line_context:. }
112 \msg_new:nnn { zref-check } { load-time-options }
113 {
114   'zref-check'~does~not~accept~load~time~options.~

```

```

115     To~configure~package~options,~use~'\iow_char:N\zrefchecksetup'.
116 }

```

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.

```

117 \cs_new_eq:NN \_zrefcheck_int_to_roman:w \_int_to_roman:w
118 \prg_new_conditional:Npnn \_zrefcheck_is_integer:n #1 { p, T , F , TF }
119 {
120   \tl_if_empty:oTF {#1}
121   { \prg_return_false: }
122   {
123     \tl_if_empty:oTF { \_zrefcheck_int_to_roman:w -0#1 }
124     { \prg_return_true: }
125     { \prg_return_false: }
126   }
127 }

```

(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.

```

128 \prg_new_protected_conditional:Npnn \_zrefcheck_is_integer_rgx:n #1 { TF }
129 {
130   \regex_match:nnTF { \A\d+\Z } {#1}
131   { \prg_return_true: }
132   { \prg_return_false: }
133 }

```

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

4.3 Options

hyperref option

`\l__zrefcheck_use_hyperref_bool`
`\l__zrefcheck_warn_hyperref_bool`

```
134 \bool_new:N \l__zrefcheck_use_hyperref_bool
135 \bool_new:N \l__zrefcheck_warn_hyperref_bool
136 \keys_define:nn { zref-check }
137 {
138   hyperref .choice: ,
139   hyperref / auto .code:n =
140     {
141       \bool_set_true:N \l__zrefcheck_use_hyperref_bool
142       \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
143     } ,
144   hyperref / true .code:n =
145     {
146       \bool_set_true:N \l__zrefcheck_use_hyperref_bool
147       \bool_set_true:N \l__zrefcheck_warn_hyperref_bool
148     } ,
149   hyperref / false .code:n =
150     {
151       \bool_set_false:N \l__zrefcheck_use_hyperref_bool
152       \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
153     } ,
154   hyperref .initial:n = auto ,
155   hyperref .default:n = auto
156 }

157 \AddToHook { begindocument }
158 {
159   \@ifpackageloaded { hyperref }
160     {
161       \bool_if:NT \l__zrefcheck_use_hyperref_bool
162         { \RequirePackage { zref-hyperref } }
163     }
164     {
165       \bool_if:NT \l__zrefcheck_warn_hyperref_bool
166         { \msg_warning:nn { zref-check } { missing-hyperref } }
167       \bool_set_false:N \l__zrefcheck_use_hyperref_bool
168     }
169   \keys_define:nn { zref-check }
170     {
171       hyperref .code:n =
172         { \msg_warning:nn { zref-check } { hyperref-preamble-only } }
173     }
174 }
```

msglevel option

\l__zrefcheck_msglevel_tl

```
175 \tl_new:N \l__zrefcheck_msglevel_tl
176 \keys_define:nn { zref-check }
177 {
178   msglevel .choice: ,
179   msglevel / warn .code:n =
180     { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } } ,
181   msglevel / info .code:n =
182     { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } } ,
183   msglevel / none .code:n =
184     { \tl_set:Nn \l__zrefcheck_msglevel_tl { none } } ,
185   msglevel / infoifdraft .code:n =
186     {
187       \ifdraft
188         { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
189         { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
190     } ,
191   msglevel / warniffinal .code:n =
192     {
193       \ifoptionfinal
194         { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
195         { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
196     } ,
197   msglevel .value_required:n = true ,
198   msglevel .initial:n = warn ,
```

ignore and ok are convenience aliases for msglevel=none, but only for use in the document body.

```
199   ignore .code:n =
200     { \msg_warning:nnn { zref-check } { ignore-ok-document-only } { ignore } } ,
201   ignore .value_forbidden:n = true ,
202   ok .code:n =
203     { \msg_warning:nnn { zref-check } { ignore-ok-document-only } { ok } } ,
204   ok .value_forbidden:n = true ,
205 }

206 \AddToHook { begindocument }
207 {
208   \keys_define:nn { zref-check }
209   {
210     ignore .meta:n = { msglevel = none } ,
211     ok .meta:n = { msglevel = none } ,
212   }
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 .value_required:n = true ,
239   onpage .initial:n = labelseq
240 }
```

closerange option

\l__zrefcheck_close_range_int

```
241 \int_new:N \l__zrefcheck_close_range_int
242 \keys_define:nn { zref-check }
243 {
244   closerange .code:n =
245     {
246       \__zrefcheck_is_integer_rgx:nTF {#1}
247         { \int_set:Nn \l__zrefcheck_close_range_int { \int_eval:n {#1} } }
248         {
249           \msg_warning:nn { zref-check } { closerange-not-positive-integer }
250           \int_set:Nn \l__zrefcheck_close_range_int { 5 }
251         }
252     } ,
253   closerange .value_required:n = true ,
254   closerange .initial:n = 5
255 }
```


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.

```
256 \bool_lazy_and:nnT
257 { \tl_if_exist_p:c { opt@ zref-check.sty } }
258 { ! \tl_if_empty_p:c { opt@ zref-check.sty } }
259 { \msg_warning:nn { zref-check } { load-time-options } }
```

`\zrefchecksetup` Provide `\zrefchecksetup`.

```
260 \NewDocumentCommand \zrefchecksetup { m }
261 { \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 `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@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`

```

262 \prop_new:N \g__zrefcheck_auxfile_lblseq_prop
263 \tl_gset:Nn \g_tmpa_tl { \c_sys_jobname_str .aux }
264 \file_if_exist:nT { \g_tmpa_tl }
265 {

```

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.

```

266   \ior_open:Nn \g_tmpa_ior { \g_tmpa_tl }
267   \group_begin:
268   \int_zero:N \l_tmpa_int
269   \tl_clear:N \l_tmpa_tl
270   \tl_clear:N \l_tmpb_tl
271   \bool_set_false:N \l_tmpa_bool
272   \ior_map_variable:NNn \g_tmpa_ior \l_tmpa_tl
273   {

```

```

274     \tl_map_variable:NNn \l_tmpa_tl \l_tmpb_tl
275     {
276         \tl_if_eq:NnTF \l_tmpb_tl { \zref@newlabel }
277         {

```

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

```

278             \bool_set_true:N \l_tmpa_bool
279         }
280     {
281         \bool_if:NTF \l_tmpa_bool
282         {
283             \bool_set_false:N \l_tmpa_bool
284             \int_incr:N \l_tmpa_int
285             \prop_gput:Nxx \g__zrefcheck_auxfile_lblseq_prop
286                 { \l_tmpb_tl } { \int_use:N \l_tmpa_int }
287         }
288     {

```

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 `\catcode` calls 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.

```

289         \tl_map_break:
290     }
291 }
292 }
293 }
294 \group_end:
295 \ior_close:N \g_tmpa_ior
296 }

```

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.

```

297 \begingroup
298 \let \@addtoreset \ltx@gobbletwo
299 \newcounter { zrefcheck }
300 \endgroup
301 \setcounter { zrefcheck } { 0 }

```

4.6 Label formats

```

\__zrefcheck_check_lblfmt:n \__zrefcheck_check_lblfmt:n {<check id int>}
302 \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>}
303 \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>}
304 \cs_new_protected:Npn \zrefcheck_get_astl:nnn #1#2#3
305 {
306 \tl_clear:N #3
307 \tl_if_eq:nnTF {#2} { lblseq }
308 {
309 \prop_get:NnNF \g__zrefcheck_auxfile_lblseq_prop {#1} #3
310 {
311 \msg_warning:nnnn { zref-check }
312 { property-not-in-label } {#1} {#2}
313 }
314 }
315 {

```

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:nnnn` (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.

```

316     \zref@ifrefundefined {#1}
317     {}
318     {
319         \zref@ifpropundefined {#2}
320         { \msg_warning:nnnn { zref-check } { property-undefined } {#2} }
321         {
322             \zref@ifrefcontainsprop {#1} {#2}
323             {
324                 \tl_set:Nx #3
325                 { \zref@extractdefault {#1} {#2} { \c_empty_tl } }
326             }
327             {
328                 \msg_warning:nnnn
329                 { zref-check } { property-not-in-label } {#1} {#2}
330             }
331         }
332     }
333 }
334 }

```

(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).

```

335 \bool_new:N \l__zrefcheck_integer_bool

```

`\l__zrefcheck_propval_tl` `\tl_new:N \l__zrefcheck_propval_tl`

```

\zrefcheck_get_asint:nnn      \zrefcheck_get_asint:nnn {<label>} {<prop>} {<int var>}
337 \cs_new_protected:Npn \zrefcheck_get_asint:nnn #1#2#3
338 {
339   \zrefcheck_get_astl:nnn {#1} {#2} { \l__zrefcheck_propval_tl }
340   \__zrefcheck_is_integer:nTF { \l__zrefcheck_propval_tl }
341   {

```

Make it an integer data type.

```

342     \int_set:Nn #3 { \int_eval:n { \l__zrefcheck_propval_tl } }
343   }
344 {
345   \bool_set_false:N \l__zrefcheck_integer_bool
346   \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`.

```

347   { }
348   {
349     \msg_warning:nnnn { zref-check }
350     { property-not-integer } {#2} {#1}
351   }
352 }
353 }

```

(End of 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>}
354 \NewDocumentCommand \zcheck { s O { } m m }
355 { \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

```

```

356 \seq_new:N \l__zrefcheck_zcheck_labels_seq
357 \int_new:N \g__zrefcheck_id_int
358 \tl_new:N \l__zrefcheck_checkbeg_tl
359 \tl_new:N \l__zrefcheck_link_label_tl
360 \tl_new:N \l__zrefcheck_link_anchor_tl
361 \bool_new:N \l__zrefcheck_link_star_bool

```

`__zrefcheck_zcheck:nmmm` 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:nmmm {<labels>} {(*)} {<checks/options>} {<text>}

```

```

362 \cs_new_protected:Npn \__zrefcheck_zcheck:nmmm #1#2#3#4
363 {
364   \group_begin:

```

Process local options and checks. We use `\seq_set_split:Nnn` to set `\l__zrefcheck_zcheck_labels_seq` – instead of `\seq_set_from_clist:Nn` – to support empty labels.

```

365   \keys_set:nn { zref-check / zcheck } {#3}
366   \seq_set_split:Nnn \l__zrefcheck_zcheck_labels_seq { , } {#1}

```

Names of the labels for this `zcheck` call.

```

367   \int_gincr:N \g__zrefcheck_id_int
368   \tl_set:Nx \l__zrefcheck_checkbeg_tl
369   { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }

```

Set `checkbeg` label.

```

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

```

Typeset `{<text>}`, with hyperlink when appropriate. Even though the first argument can receive a list of labels, there is no meaningful way to set links to multiple targets. Hence, only the first one is considered for hyperlinking.

```

371   \seq_get:NN \l__zrefcheck_zcheck_labels_seq \l__zrefcheck_link_label_tl
372   \bool_set:Nn \l__zrefcheck_link_star_bool {#2}
373   \zref@ifrefundefined { \l__zrefcheck_link_label_tl }

```

If the reference is undefined, just typeset.

```

374   {#4}
375   {
376     \bool_if:nTF
377     {
378       \l__zrefcheck_use_hyperref_bool &&
379       ! \l__zrefcheck_link_star_bool
380     }
381     {
382       \exp_args:Nx \zrefcheck_get_astl:nmm

```

```

383         { \l__zrefcheck_link_label_tl }
384         { anchor } { \l__zrefcheck_link_anchor_tl }
385         \hyperlink { \l__zrefcheck_link_anchor_tl } {#4}
386     }
387     {#4}
388 }

```

Set checkend label.

```

389 \bool_if:NT \l__zrefcheck_zcheck_end_label_bool
390 {
391     \zref@labelbylist
392     { \l__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
393     { zrefcheck-end }
394 }

```

Check if *(labels)* are defined.

```

395 \seq_map_inline:Nn \l__zrefcheck_zcheck_labels_seq
396 { \tl_if_empty:nF {##1} { \zref@refused {##1} } }

```

Run the checks.

```

397 \__zrefcheck_run_checks:nxx { \l__zrefcheck_zcheck_checks_seq }
398 { \l__zrefcheck_zcheck_labels_seq } { \l__zrefcheck_checkbeg_tl }
399 \group_end:
400 }

```

(End of definition for __zrefcheck_zcheck:nnnn.)

5.2 Targets

```

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

```

```

401 \NewDocumentCommand \zctarget { m +m }
402 {

```

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.

```

403 \group_begin:
404 \refstepcounter { zrefcheck }
405 \zref@wrapper@babel \zref@label {#1}
406 #2
407 \tl_if_empty:nF {#2}
408 {
409     \zref@wrapper@babel
410     \zref@labelbylist { \l__zrefcheck_end_lblfmt:n {#1} } { zrefcheck-end }
411 }
412 \group_end:
413 }

```

(End of definition for \zctarget.)

```

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

```

```

414 \NewDocumentEnvironment {zcregion} { m }
415 {

```



```

416   \refstepcounter { zrefcheck }
417   \zref@wrapper@babel \zref@label {#1}
418 }
419 {
420   \zref@wrapper@babel
421   \zref@labelbylist { \_zrefcheck_end_lblfmt:n {#1} } { zrefcheck-end }
422 }

```

(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 *⟨check⟩* is the name of the check, the first argument is the *{⟨label⟩}* and the second the *{⟨reference⟩}*. The existence of the check is verified by the existence of the function with this name-scheme (and signatures). As usual, this function must return either `\prg_return_true:` or `\prg_return_false:.` Of course, you can define other variants if you need them internally, it is just that what the package does expect and verifies is the existence of the `:nnF` variant.

Note that the naming convention of the checks adopts the perspective of the *⟨reference⟩*. That is, the “before” check should return true if the *⟨label⟩* occurs before the “reference”.

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

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

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`

```
423 \seq_const_from_clist:Nn \c__zrefcheck_single_label_checks_seq
424 {
425   thischap ,
426   prevchap ,
427   nextchap ,
428   chapsbefore ,
429   chapsafter ,
430   thissec ,
431   prevsec ,
432   nextsec ,
433   secsbefore ,
434   secsafter ,
435   manual ,
436 }
```

6.2 Setup

`\l__zrefcheck_zcheck_checks_seq`
`\l__zrefcheck_end_label_required_bool`

```
437 \seq_new:N \l__zrefcheck_zcheck_checks_seq
438 \bool_new:N \l__zrefcheck_zcheck_end_label_bool
```

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

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

Then we add the checks to it.

```
440 \clist_map_inline:nn
441 {
442   thispage ,
443   prevpage ,
444   nextpage ,
445   facing ,
446   otherpage ,
447   pagegap ,
448   above ,
449   below ,
450   pagesbefore ,
451   ppbefore ,
452   pagesafter ,
453   ppafter ,
454   before ,
455   after ,
456   thischap ,
457   prevchap ,
458   nextchap ,
459   chapsbefore ,
460   chapsafter ,
461   thissec ,
462   prevsec ,
463   nextsec ,
```

```

464     secsbefore ,
465     secsafter ,
466     close ,
467     far ,
468     manual ,
469 }
470 {
471 \keys_define:nn { zref-check / zcheck }
472 {
473     #1 .code:n =
474     {
475         \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq {#1}
476         \seq_if_in:NnF \c__zrefcheck_single_label_checks_seq {#1}
477         { \bool_set_true:N \l__zrefcheck_zcheck_end_label_bool }
478     } ,
479     #1 .value_forbidden:n = true ,
480 }
481 }

```

6.3 Running

```

\__zrefcheck_run_checks:nnn
\__zrefcheck_run_checks:nnn {<checks>} {<labels>} {<reference>}
<checks> are expected to be received as a sequence variable.
482 \cs_new_protected:Npn \__zrefcheck_run_checks:nnn #1#2#3
483 {
484     \group_begin:
485     \seq_map_inline:Nn #2
486     {
487         \seq_if_empty:NTF #1
488         { \__zrefcheck_message:nnnn { no-checks } { } { } { } }
489         {
490             \seq_map_inline:Nn #1
491             { \__zrefcheck_do_check:nnn {###1} {##1} {#3} }
492         }
493     }
494     \group_end:
495 }
496 \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
\l__zrefcheck_empty_label_bool
\c__zrefcheck_onpage_checks_seq

```

```

497 \bool_new:N \l__zrefcheck_passedcheck_bool
498 \bool_new:N \l__zrefcheck_onpage_bool
499 \bool_new:N \l__zrefcheck_empty_label_bool
500 \seq_const_from_clist:Nn \c__zrefcheck_onpage_checks_seq
501 { above , below , before , after }

```

Variant not provided by expl3.

```

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

```

```

503 \zrefcheck_do_check:nnn \zrefcheck_do_check:nnn {<check>} {<label beg>} {<reference beg>}
504 {
505   \group_begin:
506   \bool_set_true:N \l__zrefcheck_passedcheck_bool
507   \bool_set_false:N \l__zrefcheck_onpage_bool
508   \bool_set_false:N \l__zrefcheck_empty_label_bool
509   \cs_if_exist:cTF { __zrefcheck_check_ #1 :nnF }
510   {

```

<label beg> may be defined or not, it is arbitrary user input. Whether this is the case is checked in `\zrefcheck_zcheck:nmnn`, and due warning already ensues. So there's no need to do it again here. The only exception is the case of empty labels, for which we want to issue a failed check warning.

```

511   \zref@ifrefundefined {#2}
512   {
513     \tl_if_empty:nT {#2}
514     {
515       \bool_set_false:N \l__zrefcheck_passedcheck_bool
516       \bool_set_true:N \l__zrefcheck_empty_label_bool
517     }
518   }
519   {
520     % ‘label beg’ vs ‘reference beg’.
521     \use:c { __zrefcheck_check_ #1 :nnF }
522     {#2} {#3}
523     { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
524     % ‘reference end’ \emph{may} exist or not depending on the
525     % checks.
526     \zref@ifrefundefined { \zrefcheck_end_lblfmt:n {#3} }
527     {
528       % ‘label end’ \emph{may} have been created by the
529       % target commands.
530       \zref@ifrefundefined { \zrefcheck_end_lblfmt:n {#2} }
531       {}
532       {
533         % ‘label end’ vs ‘reference beg’.
534         \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
535         { \zrefcheck_end_lblfmt:n {#2} } {#3}
536         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
537       }
538     }
539   }
540   {
541     % ‘label beg’ vs ‘reference end’.
542     \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
543     {#2} { \zrefcheck_end_lblfmt:n {#3} }
544     { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
545     % ‘label end’ \emph{may} have been created by the
546     % target commands.
547     \zref@ifrefundefined { \zrefcheck_end_lblfmt:n {#2} }
548     {}
549     {
550       % ‘label end’ vs ‘reference beg’.
551       \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }

```

```

551         { \_zrefcheck_end_lblfmt:n {#2} } {#3}
552         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
553         % ‘label end’ vs ‘reference end’.
554         \exp_args:Nnoo \use:c { \_zrefcheck_check_ #1 :nnF }
555         { \_zrefcheck_end_lblfmt:n {#2} }
556         { \_zrefcheck_end_lblfmt:n {#3} }
557         { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
558     }
559 }

```

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.

```

560     \seq_if_in:NnT \c__zrefcheck_onpage_checks_seq {#1}
561     {
562         \_zrefcheck_check_thispage:nnT
563         {#2} {#3}
564         { \bool_set_true:N \l__zrefcheck_onpage_bool }
565         \zref@ifrefundefined { \_zrefcheck_end_lblfmt:n {#3} }
566         {
567             \zref@ifrefundefined { \_zrefcheck_end_lblfmt:n {#2} }
568             {}
569             {
570                 \_zrefcheck_check_thispage:nnT
571                 { \_zrefcheck_end_lblfmt:n {#2} } {#3}
572                 { \bool_set_true:N \l__zrefcheck_onpage_bool }
573             }
574         }
575         {
576             \_zrefcheck_check_thispage:nnT
577             {#2} { \_zrefcheck_end_lblfmt:n {#3} }
578             { \bool_set_true:N \l__zrefcheck_onpage_bool }
579             \zref@ifrefundefined { \_zrefcheck_end_lblfmt:n {#2} }
580             {}
581             {
582                 \_zrefcheck_check_thispage:nnT
583                 { \_zrefcheck_end_lblfmt:n {#2} } {#3}
584                 { \bool_set_true:N \l__zrefcheck_onpage_bool }
585                 \_zrefcheck_check_thispage:nnT
586                 { \_zrefcheck_end_lblfmt:n {#2} }
587                 { \_zrefcheck_end_lblfmt:n {#3} }
588                 { \bool_set_true:N \l__zrefcheck_onpage_bool }
589             }
590         }
591     }
592 }
593 }
594 { \msg_warning:nnn { zref-check } { check-missing } {#1} }
595 \bool_if:NTF \l__zrefcheck_passedcheck_bool
596 {
597     \bool_if:nT
598     {
599         \l__zrefcheck_msgonpage_bool &&
600         \l__zrefcheck_onpage_bool

```

```

601     }
602     {
603         \_zrefcheck_message:nnx { double-check } {#1} {#2}
604         { \zref@extractdefault {#3} {page} {'unknown'} }
605     }
606 }
607 {
608     \bool_if:NTF \l__zrefcheck_empty_label_bool
609     { \_zrefcheck_message:nnnn { empty-label } {#1} { } { } }
610     {
611         \_zrefcheck_message:nnx { check-failed } {#1} {#2}
612         { \zref@extractdefault {#3} {page} {'unknown'} }
613     }
614 }
615 \group_end:
616 }
617 \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.

```

618 \int_new:N \l__zrefcheck_lbl_int
619 \int_new:N \l__zrefcheck_ref_int
620 \int_new:N \l__zrefcheck_lbl_b_int
621 \int_new:N \l__zrefcheck_ref_b_int

```

6.4.1 This page

```

\_zrefcheck_check_thispage:nn
\_zrefcheck_check_otherpage:nn
622 \prg_new_protected_conditional:Npnn \_zrefcheck_check_thispage:nn #1#2 { T , F , TF }
623 {
624     \group_begin:
625     \bool_set_true:N \l__zrefcheck_integer_bool
626     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
627     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
628     \bool_lazy_and:nnTF
629     { \l__zrefcheck_integer_bool }
630     {
631         \int_compare_p:nNn
632         { \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.

```

633         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
634         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
635     }
636     { \group_insert_after:N \prg_return_true: }

```

```

637     { \group_insert_after:N \prg_return_false: }
638   \group_end:
639 }
640 \prg_new_protected_conditional:Npnn \__zrefcheck_check_otherpage:nn #1#2 { T , F , TF }
641 {
642   \__zrefcheck_check_thispage:nnTF {#1} {#2}
643   { \prg_return_false: }
644   { \prg_return_true: }
645 }

```

(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
646 \prg_new_protected_conditional:Npnn \__zrefcheck_check_above:nn #1#2 { F , TF }
647 {
648   \group_begin:
649   \__zrefcheck_check_thispage:nnTF {#1} {#2}
650   {
651     \bool_set_true:N \l__zrefcheck_integer_bool
652     \zrefcheck_get_asint:nnn {#1} { lblseq } { \l__zrefcheck_lbl_int }
653     \zrefcheck_get_asint:nnn {#2} { lblseq } { \l__zrefcheck_ref_int }
654     \bool_lazy_and:nnTF
655     { \l__zrefcheck_integer_bool }
656     {
657       \int_compare_p:nNn
658       { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
659       ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
660       ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
661     }
662     { \group_insert_after:N \prg_return_true: }
663     { \group_insert_after:N \prg_return_false: }
664   }
665   { \group_insert_after:N \prg_return_false: }
666   \group_end:
667 }
668 \prg_new_protected_conditional:Npnn \__zrefcheck_check_below:nn #1#2 { F , TF }
669 {
670   \__zrefcheck_check_thispage:nnTF {#1} {#2}
671   {
672     \__zrefcheck_check_above:nnTF {#1} {#2}
673     { \prg_return_false: }
674     { \prg_return_true: }
675   }
676   { \prg_return_false: }
677 }

```

(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
678 \prg_new_protected_conditional:Npnn \__zrefcheck_check_before:nn #1#2 { F }

```

```

679 {
680   \_zrefcheck_check_pagesbefore:nnTF {#1} {#2}
681   { \prg_return_true: }
682   {
683     \_zrefcheck_check_above:nnTF {#1} {#2}
684     { \prg_return_true: }
685     { \prg_return_false: }
686   }
687 }
688 \prg_new_protected_conditional:Npnn \_zrefcheck_check_after:nn #1#2 { F }
689 {
690   \_zrefcheck_check_pagesafter:nnTF {#1} {#2}
691   { \prg_return_true: }
692   {
693     \_zrefcheck_check_below:nnTF {#1} {#2}
694     { \prg_return_true: }
695     { \prg_return_false: }
696   }
697 }

```

(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
698 \prg_new_protected_conditional:Npnn \_zrefcheck_check_nextpage:nn #1#2 { F }
699 {
700   \group_begin:
701   \bool_set_true:N \l__zrefcheck_integer_bool
702   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
703   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
704   \bool_lazy_and:nnTF
705   { \l__zrefcheck_integer_bool }
706   {
707     \int_compare_p:nNn
708     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
709     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
710     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
711   }
712   { \group_insert_after:N \prg_return_true: }
713   { \group_insert_after:N \prg_return_false: }
714   \group_end:
715 }
716 \prg_new_protected_conditional:Npnn \_zrefcheck_check_prevpage:nn #1#2 { F }
717 {
718   \group_begin:
719   \bool_set_true:N \l__zrefcheck_integer_bool
720   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
721   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
722   \bool_lazy_and:nnTF
723   { \l__zrefcheck_integer_bool }
724   {
725     \int_compare_p:nNn
726     { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&

```



```

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

```

```

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

```

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

```

799     \legacy_if_p:n { @twoside } &&

```

Now we test “facing”.

```

800     (
801     (
802       \int_if_odd_p:n { \l__zrefcheck_ref_int } &&
803       \int_compare_p:nNn
804       { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 }
805     ) ||
806     (
807       \int_if_even_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_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 }

```

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

6.4.5 Close / Far

```

\__zrefcheck_check_close:nn
\__zrefcheck_check_far:nn
819 \prg_new_protected_conditional:Npnn \__zrefcheck_check_close:nn #1#2 { F , TF }
820 {
821   \group_begin:
822   \bool_set_true:N \l__zrefcheck_integer_bool
823   \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
824   \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
825   \bool_lazy_and:nnTF

```

```

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

```

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

6.4.6 Chapter

```

\__zrefcheck_check_thischap:mn
\__zrefcheck_check_nextchap:mn 845 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thischap:nn #1#2 { F }
\__zrefcheck_check_prevchap:mn 846 {
\__zrefcheck_check_chapsafter:mn 847     \group_begin:
\__zrefcheck_check_chapsbefore:mn 848     \bool_set_true:N \l__zrefcheck_integer_bool
849     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
850     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_int }
851     \bool_lazy_and:nnTF
852     { \l__zrefcheck_integer_bool }
853     {
854         \int_compare_p:nNn
855         { \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.

```

856         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
857         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
858     }
859     { \group_insert_after:N \prg_return_true: }
860     { \group_insert_after:N \prg_return_false: }
861 \group_end:
862 }
863 \prg_new_protected_conditional:Npnn \__zrefcheck_check_nextchap:nn #1#2 { F }
864 {
865     \group_begin:
866     \bool_set_true:N \l__zrefcheck_integer_bool
867     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }

```

```

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

```

```

922     { \l__zrefcheck_integer_bool }
923     {
924         \int_compare_p:nNn
925         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
926         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
927         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
928     }
929     { \group_insert_after:N \prg_return_true: }
930     { \group_insert_after:N \prg_return_false: }
931 \group_end:
932 }

```

(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
933 \prg_new_protected_conditional:Npnn \__zrefcheck_check_thissec:nn #1#2 { F }
934 {
935     \group_begin:
936     \bool_set_true:N \l__zrefcheck_integer_bool
937     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
938     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
939     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
940     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
941     \bool_lazy_and:nnTF
942     { \l__zrefcheck_integer_bool }
943     {
944         \int_compare_p:nNn
945         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
946         \int_compare_p:nNn
947         { \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.

```

948         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
949         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
950     }
951     { \group_insert_after:N \prg_return_true: }
952     { \group_insert_after:N \prg_return_false: }
953 \group_end:
954 }
955 \prg_new_protected_conditional:Npnn \__zrefcheck_check_nextsec:nn #1#2 { F }
956 {
957     \group_begin:
958     \bool_set_true:N \l__zrefcheck_integer_bool
959     \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
960     \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
961     \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
962     \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
963     \bool_lazy_and:nnTF

```

```

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

```

```

1018 }
1019 \prg_new_protected_conditional:Npnn \__zrefcheck_check_secsbefore:nn #1#2 { F }
1020 {
1021   \group_begin:
1022   \bool_set_true:N \l__zrefcheck_integer_bool
1023   \zrefcheck_get_asint:nnn {#1} { zc@abssec } { \l__zrefcheck_lbl_int }
1024   \zrefcheck_get_asint:nnn {#2} { zc@abssec } { \l__zrefcheck_ref_int }
1025   \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
1026   \zrefcheck_get_asint:nnn {#2} { zc@abschap } { \l__zrefcheck_ref_b_int }
1027   \bool_lazy_and:nnTF
1028   { \l__zrefcheck_integer_bool }
1029   {
1030     \int_compare_p:nNn
1031     { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
1032     \int_compare_p:nNn
1033     { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
1034     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
1035     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
1036   }
1037   { \group_insert_after:N \prg_return_true: }
1038   { \group_insert_after:N \prg_return_false: }
1039   \group_end:
1040 }

```

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

6.4.8 Manual

__zrefcheck_check_manual:nn

```

1041 \prg_new_protected_conditional:Npnn \__zrefcheck_check_manual:nn #1#2 { F }
1042 { \prg_return_false: }

```

(End of definition for __zrefcheck_check_manual:nn.)

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: 1043 \cs_new_protected:Npn \zrefcheck_zcref_beg_label:
\zrefcheck_zcref_run_checks_on_labels:n 1044 {
1045   \int_gincr:N \g__zrefcheck_id_int
1046   \tl_set:Nx \l__zrefcheck_checkbeg_tl
1047   { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
1048   \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrefcheck-check }
1049 }
1050 \cs_new_protected:Npn \zrefcheck_zcref_end_label_maybe:
1051 {
1052   \bool_if:NT \l__zrefcheck_zcheck_end_label_bool

```

```

1053     {
1054       \zref@labelbylist
1055       { \_zrefcheck_end_lblfmt:n { \l_zrefcheck_checkbeg_tl } }
1056       { zrefcheck-end }
1057     }
1058   }
1059 \cs_new_protected:Npn \zrefcheck_zcref_run_checks_on_labels:n #1
1060 {
1061   \_zrefcheck_run_checks:nnx
1062   { \l_zrefcheck_zcheck_checks_seq } {#1} { \l_zrefcheck_checkbeg_tl }
1063 }

```

(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

```

```

1064 \cs_new_protected:Npn \zrefcheck_zrefvario_label:
1065 {
1066   \int_gincr:N \g_zrefcheck_id_int
1067   \tl_set:Nx \l_zrefcheck_checkbeg_tl
1068   { \_zrefcheck_check_lblfmt:n { \g_zrefcheck_id_int } }
1069   \zref@labelbylist { \l_zrefcheck_checkbeg_tl } { zrefcheck-zrefvario }
1070 }
1071 \cs_new_protected:Npn \zrefcheck_zrefvario_run_check_on_label:nn #1#2
1072 { \_zrefcheck_do_check:nnV {#1} {#2} \l_zrefcheck_checkbeg_tl }
1073 \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 ??.)

```

1074 </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	
<code>\</code>	72, 84, 115
A	
<code>\A</code>	130
<code>\AddToHook</code>	21, 25, 157, 206
B	
<code>\begingroup</code>	297
bool commands:	
<code>\bool_if:NTF</code>	161, 165, 281, 389, 595, 608, 1052
<code>\bool_if:nTF</code>	376, 597
<code>\bool_lazy_and:nnTF</code>	13, 256, 628, 654, 704, 722, 740, 759, 778, 796, 825, 851, 869, 886, 904, 921, 941, 963, 984, 1006, 1027
<code>\bool_new:N</code>	134, 135, 214, 335, 361, 438, 497, 498, 499
<code>\bool_set:Nn</code>	372
<code>\bool_set_false:N</code>	142, 151, 152, 167, 220, 229, 236, 271, 283, 345, 507, 508, 515, 523, 536, 543, 552, 557
<code>\bool_set_true:N</code>	141, 146, 147, 224, 230, 235, 278, 477, 506, 516, 564, 572,

578, 584, 588, 625, 651, 701, 719, 737, 756, 775, 793, 822, 848, 866, 883, 901, 918, 936, 958, 979, 1001, 1022	
\l_tmpa_bool	11, 271, 278, 281, 283
C	
\catcode	11
\chapter	2, 27
clist commands:	
\clist_map_inline:nn	440
cs commands:	
\cs_generate_variant:Nn	
.	55, 496, 502, 617, 1073
\cs_if_exist:NTF	509
\cs_new:Npn	302, 303
\cs_new_eq:NN	117, 752, 771
\cs_new_protected:Npn	
.	50, 304, 337, 362, 482, 503, 1043, 1050, 1059, 1064, 1071
D	
\d	130
\def	3
E	
\emph	524, 528, 544
\endgroup	300
exp commands:	
\exp_args:Nnno	502, 541
\exp_args:Nno	534, 550
\exp_args:Noo	554
\exp_args:Nx	382
F	
file commands:	
\file_if_exist:nTF	264
\fmtversion	5
G	
group commands:	
\group_begin:	267, 364, 403, 484, 505, 624, 648, 700, 718, 736, 755, 774, 792, 821, 847, 865, 882, 900, 917, 935, 957, 978, 1000, 1021
\group_end:	294, 399, 412, 494, 615, 638, 666, 714, 732, 750, 769, 788, 817, 837, 861, 878, 896, 913, 931, 953, 974, 996, 1017, 1039
\group_insert_after:N	
.	636, 637, 662, 663, 665, 712, 713, 730, 731, 748, 749, 767, 768, 786, 787, 815, 816, 835, 836, 859, 860, 876, 877, 894, 895, 911, 912, 929, 930, 951, 952, 972, 973, 994, 995, 1015, 1016, 1037, 1038
H	
\hyperlink	14, 385
I	
\ifdraft	187, 228
\IfFormatAtLeastTF	5, 6
\ifoptionfinal	193, 234
int commands:	
\int_abs:n	782, 829
\int_compare_p:nNn	
.	631, 633, 634, 657, 659, 660, 707, 709, 710, 725, 727, 728, 743, 745, 746, 762, 764, 765, 781, 783, 784, 803, 808, 812, 813, 828, 832, 833, 854, 856, 857, 872, 874, 889, 891, 892, 907, 909, 924, 926, 927, 944, 946, 948, 949, 966, 968, 970, 987, 989, 991, 992, 1009, 1011, 1013, 1030, 1032, 1034, 1035
\int_eval:n	247, 342
\int_gincr:N	367, 1045, 1066
\int_if_even_p:n	807
\int_if_odd_p:n	802
\int_incr:N	284
\int_new:N	241, 357, 618, 619, 620, 621
\int_set:Nn	247, 250, 342
\int_use:N	23, 27, 286, 302
\int_zero:N	268
\l_tmpa_int	268, 284, 286
int internal commands:	
__int_to_roman:w	5, 117
ior commands:	
\ior_close:N	295
\ior_map_variable:NNn	272
\ior_open:Nn	266
\g_tmpa_ior	266, 272, 295
iow commands:	
\iow_char:N	72, 84, 115
\iow_newline:	83, 87, 90, 107
K	
keys commands:	
\keys_define:nn	
.	136, 169, 176, 208, 215, 242, 439, 471
\keys_set:nn	261, 365
L	
\label	11
legacy commands:	
\legacy_if_p:n	799
\let	298

M	
msg commands:	
\msg_line_context:	58, 63, 68, 72, 74, 76, 78, 80, 94, 97, 107, 111
\msg_new:nnn	56, 61, 66, 71, 73, 75, 77, 79, 81, 86, 88, 93, 95, 100, 105, 110, 112
\msg_warning:nn	166, 172, 249, 259
\msg_warning:nnn	200, 203, 594
\msg_warning:nnnn	311, 320, 328, 349
N	
\NeedsTeXFormat	4
\newcounter	19, 20, 299
\NewDocumentCommand	260, 354, 401
\NewDocumentEnvironment	414
P	
\PackageError	9
\pageref	11
prg commands:	
\prg_new_conditional:Npnn	118
\prg_new_protected_conditional:Npnn	17, 128, 622, 640, 646, 668, 678, 688, 698, 716, 734, 753, 772, 790, 819, 839, 845, 863, 880, 898, 915, 933, 955, 976, 998, 1019, 1041
\prg_return_false:	17, 121, 125, 132, 637, 643, 663, 665, 673, 676, 685, 695, 713, 731, 749, 768, 787, 816, 836, 842, 860, 877, 895, 912, 930, 952, 973, 995, 1016, 1038, 1042
\prg_return_true:	17, 124, 131, 636, 644, 662, 674, 681, 684, 691, 694, 712, 730, 748, 767, 786, 815, 835, 843, 859, 876, 894, 911, 929, 951, 972, 994, 1015, 1037
prop commands:	
\prop_get:NnNTF	309
\prop_gput:Nnn	285
\prop_new:N	262
\providecommand	5
\ProvidesExplPackage	14
R	
\refstepcounter	12, 16, 404, 416
regex commands:	
\regex_match:nnTF	130
\RequirePackage	16, 17, 18, 162
\romannumeral	5
S	
\section	29
seq commands:	
\seq_const_from_clist:Nn	423, 500
\seq_get:NN	371
\seq_if_empty:NnTF	487
\seq_if_in:NnTF	476, 560
\seq_map_inline:Nn	395, 485, 490
\seq_new:N	356, 437
\seq_put_right:Nn	475
\seq_set_from_clist:Nn	15
\seq_set_split:Nnn	15, 366
\setcounter	301
\space	11
\stepcounter	22, 26
sys commands:	
\c_sys_jobname_str	263
T	
TeX and L ^A T _E X 2 _ε commands:	
\@addtoreset	298
\@currentlabel	16
\@ifl@t@r	5
\@ifpackageloaded	159
\@newl@bel	9
\c@zcc@abschap	23
\c@zcc@abssec	27
\ltx@gobbletwo	298
\zref@addprop	17, 24, 28
\zref@addprops	30, 38, 45
\zref@extractdefault	12, 325, 604, 612
\zref@ifpropundefined	319
\zref@ifrefcontainsprop	322
\zref@ifrefundefined	13, 316, 346, 373, 511, 526, 530, 546, 565, 567, 579
\ZREF@label	11
\zref@label	11, 405, 417
\zref@labelbylist	370, 391, 410, 421, 1048, 1054, 1069
\ZREF@mainlist	24, 28
\zref@newlabel	9–11, 276
\zref@newlist	29, 37, 44
\zref@newprop	17, 23, 27
\zref@refused	13, 396
\zref@require@unique	12
\zref@wrapper@babel	10, 15, 355, 405, 409, 417, 420
\zrefcheck@required@kernel	3, 4, 6, 11
tl commands:	
\c_empty_tl	12, 325
\tl_clear:N	12, 269, 270, 306
\tl_gset:Nn	263
\tl_if_blank:nTF	5
\tl_if_empty:nTF	5, 120, 123, 396, 407, 513
\tl_if_empty_p:N	258
\tl_if_eq:NnTF	276
\tl_if_eq:nnTF	307
\tl_if_exist_p:N	257

<code>\tl_map_break:</code>	289	<code>__zrefcheck_check_below:nnTF</code> ..	693
<code>\tl_map_variable:NNn</code>	274	<code>__zrefcheck_check_chapsafter:nn</code>	845, 898
<code>\tl_new:N</code>	175, 336, 358, 359, 360	<code>__zrefcheck_check_chapsbefore:nn</code>	845, 915
<code>\tl_set:Nn</code>	180, 182, 184, 188, 189, 194, 195, 324, 368, 1046, 1067	<code>__zrefcheck_check_close:nn</code>	819, 819
<code>\g_tmpa_tl</code>	263, 264, 266	<code>__zrefcheck_check_close:nnTF</code> ..	841
<code>\l_tmpa_tl</code>	269, 272, 274	<code>__zrefcheck_check_facing:nn</code>	698, 790
<code>\l_tmpb_tl</code>	270, 274, 276, 286	<code>__zrefcheck_check_far:nn</code> ..	819, 839
U			
use commands:		<code>__zrefcheck_check_lblfmt:n</code>	12, 302, 302, 369, 1047, 1068
<code>\use:N</code>	52, 521, 534, 541, 550, 554	<code>__zrefcheck_check_manual:nn</code>	1041, 1041
Z			
<code>\Z</code>	130	<code>__zrefcheck_check_nextchap:nn</code> ..	845, 863
<code>\zcheck</code>	14, 15, 17, 354	<code>__zrefcheck_check_nextpage:nn</code> ..	698, 698
<code>\zcref</code>	31	<code>__zrefcheck_check_nextsec:nn</code> ..	933, 955
<code>zcregion</code>	414	<code>__zrefcheck_check_otherpage:nn</code> ..	622, 640
<code>\zctarget</code>	16, 401	<code>__zrefcheck_check_pagegap:nn</code> ..	698, 772
<code>\zref</code>	15	<code>__zrefcheck_check_pagesafter:nn</code>	698, 753
zrefcheck commands:		<code>__zrefcheck_check_pagesafter:nnTF</code>	690, 771
<code>\zrefcheck_get_asint:nnn</code>	13, 14, 17, 337, 337, 626, 627, 652, 653, 702, 703, 720, 721, 738, 739, 757, 758, 776, 777, 794, 795, 823, 824, 849, 850, 867, 868, 884, 885, 902, 903, 919, 920, 937, 938, 939, 940, 959, 960, 961, 962, 980, 981, 982, 983, 1002, 1003, 1004, 1005, 1023, 1024, 1025, 1026	<code>__zrefcheck_check_pagesbefore:nn</code>	698, 734
<code>\zrefcheck_get_astl:nnn</code>	12, 13, 17, 304, 304, 339, 382	<code>__zrefcheck_check_pagesbefore:nnTF</code>	680, 752
<code>\zrefcheck_zcref_beg_label:</code>	1043, 1043	<code>__zrefcheck_check_ppafter:nn</code> ..	698
<code>\zrefcheck_zcref_end_label_-</code> maybe:	1043, 1050	<code>__zrefcheck_check_ppafter:nnTF</code>	771
<code>\zrefcheck_zcref_run_checks_on_-</code> labels:n	1043, 1059	<code>__zrefcheck_check_ppbefore:nn</code> ..	698
<code>\zrefcheck_zrefvario_label:</code>	1064, 1064	<code>__zrefcheck_check_ppbefore:nnTF</code>	752
<code>\zrefcheck_zrefvario_run_check_-</code> on_label:n	1064	<code>__zrefcheck_check_prevchap:nn</code> ..	845, 880
<code>\zrefcheck_zrefvario_run_check_-</code> on_label:nn	1071, 1073	<code>__zrefcheck_check_prevpage:nn</code> ..	698, 716
zrefcheck internal commands:		<code>__zrefcheck_check_prevsec:nn</code> ..	933, 976
<code>\g_zrefcheck_auxfile_lblseq_-</code> prop	10, 12, 17, 262, 285, 309	<code>__zrefcheck_check_secsafter:nn</code> ..	933, 998
<code>__zrefcheck_check_(check):nn</code> ..	17	<code>__zrefcheck_check_secsbefore:nn</code>	933, 1019
<code>__zrefcheck_check_above:nn</code>	646, 646	<code>__zrefcheck_check_thischap:nn</code> ..	845, 845
<code>__zrefcheck_check_above:nnTF</code> ..	672, 683	<code>__zrefcheck_check_thispage:nn</code> ..	622, 622
<code>__zrefcheck_check_after:nn</code>	678, 688	<code>__zrefcheck_check_thispage:nnTF</code> 562, 570, 576, 582, 585, 642, 649, 670	
<code>__zrefcheck_check_before:nn</code>	678, 678	<code>__zrefcheck_check_thissec:nn</code> ..	933, 933
<code>__zrefcheck_check_below:nn</code>	646, 668		

\l_zrefcheck_checkbeg_tl	\l_zrefcheck_link_label_tl
. 15, 358, 368, 370, 392, 398, 1046, 15, 359, 371, 373, 383
1048, 1055, 1062, 1067, 1069, 1072	\l_zrefcheck_link_star_bool
\l_zrefcheck_close_range_int 15, 361, 372, 379
. 8, 241, 247, 250, 831	_zrefcheck_message:nnnn
_zrefcheck_do_check:nnn 50, 50, 55, 488, 603, 609, 611
. 20, 491, 503, 503, 617, 1072	\l_zrefcheck_msglevel_tl . . 7, 52,
\l_zrefcheck_empty_label_bool	175, 180, 182, 184, 188, 189, 194, 195
. 19, 499, 508, 516, 608	\l_zrefcheck_msgonpage_bool . 8,
\l_zrefcheck_end_label_required_	214, 220, 224, 229, 230, 235, 236, 599
bool 18	\l_zrefcheck_onpage_bool . . . 19,
_zrefcheck_end_lblfmt:n . . . 12,	498, 507, 564, 572, 578, 584, 588, 600
303, 303, 392, 410, 421, 526, 530,	\c_zrefcheck_onpage_checks_seq .
535, 542, 546, 551, 555, 556, 565, 19, 500, 560
567, 571, 577, 579, 583, 586, 587, 1055	\l_zrefcheck_passedcheck_bool . .
_zrefcheck_get_asint:nnn 12 19, 497,
_zrefcheck_get_astl:nnn 12	506, 515, 523, 536, 543, 552, 557, 595
\g_zrefcheck_id_int 15,	\l_zrefcheck_propval_tl
357, 367, 369, 1045, 1047, 1066, 1068 13, 336, 339, 340, 342
_zrefcheck_int_to_roman:w	\l_zrefcheck_ref_b_int
. 117, 117, 123 22, 621, 940, 945, 962,
\l_zrefcheck_integer_bool	967, 983, 988, 1005, 1010, 1026, 1031
. 13, 335, 345, 625, 629, 651,	\l_zrefcheck_ref_int . . 22, 619,
655, 701, 705, 719, 723, 737, 741,	627, 632, 634, 653, 658, 660, 703,
756, 760, 775, 779, 793, 797, 822,	708, 710, 721, 726, 728, 739, 744,
826, 848, 852, 866, 870, 883, 887,	746, 758, 763, 765, 777, 782, 784,
901, 905, 918, 922, 936, 942, 958,	795, 802, 804, 807, 809, 813, 824,
964, 979, 985, 1001, 1007, 1022, 1028	829, 833, 850, 855, 857, 868, 873,
_zrefcheck_is_integer:n 5, 117, 118	885, 890, 892, 903, 908, 920, 925,
_zrefcheck_is_integer:nTF . . . 340	927, 938, 947, 949, 960, 969, 981,
_zrefcheck_is_integer_rgx:n	990, 992, 1003, 1012, 1024, 1033, 1035
. 5, 128, 128	_zrefcheck_run_checks:nnn
_zrefcheck_is_integer_rgx:nTF 246 19, 397, 482, 482, 496, 1061
\l_zrefcheck_lbl_b_int	\c_zrefcheck_single_label_
. 22, 620, 939, 945, 961,	checks_seq 17, 18, 423, 476
967, 982, 988, 1004, 1010, 1025, 1031	\l_zrefcheck_use_hyperref_bool .
\l_zrefcheck_lbl_int 6, 134, 141, 146, 151, 161, 167, 378
. 22, 618, 626, 632,	\l_zrefcheck_warn_hyperref_bool
633, 652, 658, 659, 702, 708, 709, 6, 135, 142, 147, 152, 165
720, 726, 727, 738, 744, 745, 757,	_zrefcheck_zcheck:nnnn
763, 764, 776, 782, 783, 794, 804, 15, 355, 362, 362
809, 812, 823, 829, 832, 849, 855,	_zrefcheck_zcheck:nnnnn 13, 14, 20
856, 867, 873, 874, 884, 890, 891,	\l_zrefcheck_zcheck_checks_seq .
902, 908, 909, 919, 925, 926, 937, 18, 397, 437, 475, 1062
947, 948, 959, 969, 970, 980, 990,	\l_zrefcheck_zcheck_end_label_
991, 1002, 1012, 1013, 1023, 1033, 1034	bool 389, 438, 477, 1052
\l_zrefcheck_link_anchor_tl	\l_zrefcheck_zcheck_labels_seq .
. 15, 360, 384, 385 15, 356, 366, 371, 395, 398
	\zrefchecksetup 9, 260