The zref-check package[*]

Gustavo Barros[†]

2021-07-27

Contents

[*]This file describes v0.1.0-alpha, last revised 2021-07-27.
[†]`https://github.com/gusbrs/zref-check`

1 Introduction

zref-check provides an user interface for making LaTeX cross-references exploiting document contextual information to enrich the way the reference can be rendered. In so doing, it caters to the same kind of need as varioref does. But the UI concept is quite different. I think it is fair to say that, in relation to varioref, zref-check offers a little less automation and a lot more flexibility.

The basic idea is that, instead of trying to provide the text to be typeset based on the contextual information (as varioref does), zref-check lets the user supply an arbitrary text and specify a number of checks to be done on the label(s) being referred to. If the checks fail, a warning is issued upon compilation, so that the user can go back to that cross-reference and correct it as needed. In a way, this shares the spirit of widows-and-orphans: instead of trying to fix it for you automatically (as nowidow does), it just provides a warning so that the problem can be identified (and fixed) without having to rely on burdensome and error prone manual proof-reading.

Though, admittedly, the kind of automation varioref provides may be preferred in a number of use cases, in others there is a lot to gain on the extra flexibility zref-check provides. The writing style, the variety of expressions you may use for similar situations, does not need to be sacrificed for the convenience. zref-check cross-references do not need to "feel" automated to be consistently checked. Localization is also not an issue, for the same reason. There is also much more document context we can leverage by separating "typesetting" from "checking" (see Section 5).

zref-check depends on zref, as the name entails, which means it is able to work with zref labels, in general created by `\zlabel`, but also with `\zrctarget` and the `zrcregion` environment provided by this package. This has some advantages, particularly the data flexibility of zref, and the absence of the ubiquitous "load-order" and compatibility problems which are well known to afflict LaTeX packages of this area of functionality. On the other hand, the reliance on zref labels may be seen as an inconvenience, since users of the standard cross-reference infrastructure will need to add extra labels for this. That's true. But zref-check is not meant to replace the existing functionality of the traditional packages (to my knowledge, it only intersects directly with varioref). Indeed, it is easy to see the use in tandem with standard references, for example:

```
... Figure~\ref{fig:figure-1}, \zrcheck*{fig:figure-1}[nextpage]{on
the next page}.
```

Besides, zref does not share the label name-space with the standard labels, so that you can call both `\label` and `\zlabel` with the same label name (manually, or through a convenience macro), to ease the label set administration. The example above presumes that was the case.

1.1 Hard vs. soft cross-references

The standard LaTeX cross-reference infrastructure, even considering the package ecosystem, is made to work with and refer to specific numbered document elements. Chapters, sections, figures, tables, equations, etc. The cross-reference will normally produce that number (which is the element's "id") and, eventually, its "type" (the counter). We may also refer to the page that element occurs and even its "title" (in which case, atypically, we may even get to refer to an unnumbered section, provided we also implicitly supply by some means the "id"). These are what I'm calling here "hard" cross-references.

However, there are other kinds of "soft" cross-references we routinely do in our documents. Expressions such as "as previously discussed", "as mentioned before", "as will be soon elaborated", and so on, are a powerful discursive instrument, which enriches the text, by offering hints to the arguments' threads, without necessarily "smashing them into the reader's face". So, we don't say "on footnote 57, pag. 34", but "previously", not "on Section 3.4", but "below", or "later on".

Granted, the need and usage of this type of document self reference certainly depend on the kind of document, on the area of knowledge, etc. However, they do tend to be employed in a number of places, particularly in longer documents. And that's precisely the scenario in which they may become problematic. If your document is short (say, a paper/article) and it was made in a reasonably short spurt, you'll probably won't bother with this kind of references. In this case, and to use varioref's expression, you "usually know (!)" them to be correct. However, if you are preparing one of those long, complex, and "long ground" documents, with several rounds of editing and content rearranging, this kind of references will likely bring you trouble. They are not only hard to check and find, but they are also hard to fix. After all, if you are making one such reference, you are taking that statement as a premisse at the current point in the text. So, if that reference is missing, or relocated, you may need to bring in the support to the premisse for your argument to close, rather than just "adjust the reference text".

To my knowledge, there is no LaTeX package providing support for this kind of cross-reference, zref-check does so. Of course, this is already possible with the standard infrastructure, zref-check just streamlines the task.

2 Loading the package

As usual:

```
\usepackage[⟨options⟩]{zref-check}
```

3 Dependencies

zref is required, of course, but in particular, its modules zref-user and zref-abspage are loaded by default. ifdraft is also loaded by default. A recent LaTeX kernel is required, since we rely on the new hook system from ltcmdhooks for the sectioning checks. If hyperref is loaded and option hyperref is given, zref-check makes use of it, but it does not load the package for you.

4 User interface

`\zrcheck` `\zrcheck[⟨options⟩]{⟨labels⟩}[⟨checks⟩]{⟨text⟩}`

Typesets {⟨*text*⟩}, as given, while performing the comma separated list of [⟨*checks*⟩] on each of the comma separated list of {⟨*labels*⟩}. In addition to that, it places a pair of (internal) `zlabel`s, one at the start of {⟨*text*⟩}, another one at the end of {⟨*text*⟩}, which are used to run the checks against each of the {⟨*labels*⟩}. When hyperref support is enabled, {⟨*text*⟩} will be made a hyperlink to *the first* label in {⟨*labels*⟩}. The starred version of the command does the same as the plain one, just does not form a link. The [⟨*options*⟩] are (mostly) the same as those of the package, and can be given to local effect. Note that the {⟨*text*⟩} argument of `\zrcheck` is not long because, if it was, we'd not be able to use if for building hyperlinks, besides there is no need for it to be.

`\zrctarget` `\zrctarget{⟨label⟩}{⟨text⟩}`

Typesets {⟨*text*⟩}, as given, and places a pair of `zlabel`s, one at the start of {⟨*text*⟩}, using {⟨*label*⟩} as label name, another one (internal) at the end of {⟨*text*⟩}.

`zrcregion`
```
\begin{zrcregion}{⟨label⟩}
  ...
\end{zrcregion}
```

Just an environment that does pretty much the same as `\zrctarget`, for cases of longer stretches of text. If you don't like to use the environment for whatever reason, you may also set two `\zrctarget`s (with empty {⟨*text*⟩} arguments), one at the beginning and another one at the end, and run `\zrcheck` against both of them to the same effect.

`\zrchecksetup` `\zrchecksetup{⟨options⟩}`

Sets zref-check's options (see Section 6).

5 Checks

zref-check provides several "checks" to be used with `\zrcheck`. The checks may be combined in a `\zrcheck` call, e.g. `[close, after]`, or `[thischap, before]`. In this case, each check in [⟨*checks*⟩] is performed against each of the {⟨*labels*⟩}. This is done independently for each check, which means, in practice, that the checks bear a logical `AND` relation to the others. Whether the combination is meaningful, is up to the user. As is the correspondence between the [⟨*checks*⟩] and the {⟨*text*>⟩} in `\zrcheck`.

Note that the naming convention of the checks adopts the perspective of `\zrcheck`. That is, the name of the check describes the position of the label being checked, relative to the `\zrcheck` call being made. For example, the `before` check should issue no message if the {⟨*label*⟩} occurs before `\zrcheck`.

The available checks are the following:

`thispage`	{⟨*label*⟩} occurs on the same page as `\zrcheck`.
`prevpage`	{⟨*label*⟩} occurs on the previous page relative to `\zrcheck`.
`nextpage`	{⟨*label*⟩} occurs on the next page relative to `\zrcheck`.

`facing`	On a `twoside` document, both `{`⟨*label*⟩`}` and `\zrcheck` fall onto a double spread, each on one of the two facing pages.
`above`	`{`⟨*label*⟩`}` and `\zrcheck` are both on the same page, and `{`⟨*label*⟩`}` occurs "above" `\zrcheck` (for how this is inferred, see Section 8.2).
`below`	`{`⟨*label*⟩`}` and `\zrcheck` are both on the same page, and `{`⟨*label*⟩`}` occurs "below" `\zrcheck`.
`pagesbefore`	`{`⟨*label*⟩`}` occurs on any page before the one of `\zrcheck`.
`ppbefore`	Convenience alias for `pagesbefore`.
`pagesafter`	`{`⟨*label*⟩`}` occurs on any page after the one of `\zrcheck`.
`ppafter`	Convenience alias for `pagesafter`.
`before`	Either `above` or `pagesbefore`.
`after`	Either `below` or `pagesafter`.
`thischap`	`{`⟨*label*⟩`}` occurs on the same chapter as `\zrcheck`.
`prevchap`	`{`⟨*label*⟩`}` occurs on the previous chapter relative to the one of `\zrcheck`.
`nextchap`	`{`⟨*label*⟩`}` occurs on the next chapter relative to the one of `\zrcheck`.
`chapsbefore`	`{`⟨*label*⟩`}` occurs on any chapter before the one of `\zrcheck`.
`chapsafter`	`{`⟨*label*⟩`}` occurs on any chapter after the one of `\zrcheck`.
`thissec`	`{`⟨*label*⟩`}` occurs on the same section as `\zrcheck`.
`prevsec`	`{`⟨*label*⟩`}` occurs on the previous section (of the same chapter) relative to the one of `\zrcheck`.
`nextsec`	`{`⟨*label*⟩`}` occurs on the next section (of the same chapter) relative to the one of `\zrcheck`.
`secsbefore`	`{`⟨*label*⟩`}` occurs on any section (of the same chapter) before the one of `\zrcheck`.
`secsafter`	`{`⟨*label*⟩`}` occurs on any section (of the same chapter) after the one of `\zrcheck`.
`close`	`{`⟨*label*⟩`}` occurs within a page range from `closerange` pages before the one of `\zrcheck` to `closerange` pages after it (about `closerange`, see Section 6).
`far`	Not `close`.

6 Options

Options are a standard `key=value` comma separated list, and can be set globally either as `\usepackage[`⟨*options*⟩`]` at load-time (see Section 2), or by means of `\zrchecksetup` (see Section 4) in the preamble. Most options can also be used with local effects, through the optional argument `[`⟨*options*⟩`]` of `\zrcheck`.

`hyperref` Controls the use of hyperref by zref-check and takes values `auto`, `true`, `false`. The default value, `auto`, makes zref-check use hyperref if it is loaded, meaning `\zrcheck` can be hyperlinked to the *first label* in `{`⟨*labels*⟩`}`. `true` does the same thing, but warns if hyperref is not loaded (hyperref is never loaded for you). In either case, if hyperref is loaded, module zref-hyperref is also loaded by zref-check. `false` means not to use hyperref regardless of its availability. This is a preamble only option, but `\zrcheck` provides granular control of hyperlinking by means of its starred version.

`msglevel` Sets the level of messages issued by `\zrcheck` failed checks and takes values `warn`, `info`, `none`, `obeydraft`, `obeyfinal`. The default value, `warn`, issues messages both to the terminal and to the log file, `info` issues messages to the log file only, `none` suppresses all messages. `obeydraft` corresponds to `info` if option `draft` is passed to `\documentclass`, and to `warn` otherwise. `obeyfinal` corresponds to `warn` if option `final` is (explicitly) passed to `\documentclass` and `info` otherwise. `ignore` is provided as convenience alias for `msglevel=none` for local use only. This option only affects the messages issued by the checks in `\zrcheck`, not other messages or warnings of the package. In particular, it does not affect warnings issued for undefined labels, which just use `\zref@refused` and thus are the same as standard LaTeX ones for this purpose.

`onpage` Allows to control the messaging style for "within page checks", and takes values `labelseq`, `msg`, `obeydraft`, `obeyfinal`. The default, `labelseq` uses the labels' shipout sequence, as retrieved from the `.aux` file, to infer relative position within the page. `msg` also uses the same method for checking relative position, but issues a (different) message **even if the check passes**, to provide a simple workflow for robust checking of "false negatives" at a final typesetting stage of the document, considering the label sequence is not fool proof (for details, see Section 8.2). `msg` also issues its messages at the same level defined in `msglevel`. `obeydraft` corresponds to `labelseq` if option `draft` is passed to `\documentclass` and to `msg` otherwise. `obeyfinal` corresponds to `msg` if option `final` is (explicitly) passed to `\documentclass`, and to `labelseq` otherwise.

`closerange` Defines the width of range of pages relative to the reference, that are considered "close" by the `close` check. Takes an integer as value, with default 5.

7 Label names

All user commands have their `{`⟨*label*⟩`}` arguments protected by `\zref@wrapper@babel`, so that we should have equivalent support in that regard, as zref itself does. However, zref-check sets labels which either start with `zrefcheck@` or end with `@zrefcheck`, for internal use. Label names with either of those are considered reserved by the package.

8 Technique and limitations

There are three qualitatively different kinds of checks being used by `\zrcheck`, according to the source and reliability of the information they mobilize: page number checks, within page checks, and sectioning checks.

8.1 Page number checks

Page number checks – `thispage`, `prevpage`, `nextpage`, `facing`, `pagesbefore`, `pagesafter` – use the abspage property provided by the zref-abspage module. This is a solid piece of information, on which we can rely upon. However, despite that, page number checks may still become ill-defined, if the `{`⟨*text*⟩`}` argument in `\zrcheck`, when typeset, crosses page boundaries, starting in one page, and finishing in another. The same can happen with the text in `\zrctarget` and the `zrcregion` environment.

This is why the user commands of this package set always a pair or labels around `{⟨text⟩}`. So, when checking `\zrcheck` against a regular `zlabel` both the start and the end of the `{⟨text⟩}` are checked against the label, and the check fails if either of them fails. When checking `\zrcheck` against a `\zrctarget` or a `zrcregion`, both beginnings and ends are checked against each other two by two, and if any of them fails, the check fails. In other words, if a page number checks passes, we know that the entire `{⟨text⟩}` arguments pass it.

This is a corner case (albeit relevant) which must be taken care of, and it is possible to do so robustly. Hence, we can expect fully reliable results in these tests.

8.2 Within page checks

When both label and reference fall on the same page things become much trickier. This is basically the case of the checks `above` and `below` (and, through them, `before` and `after`). There is no equally reliable information (that I know of) as we have for the page number checks for this, especially when floats come into play. Which, of course, is the interesting case to handle.

To infer relative position of label and reference on the same page, zref-check uses the labels' shipout sequence, which is retrieved at load-time from the order in which the labels occur in the `.aux` file. Indeed, zref writes labels to the `.aux` file at shipout (and, hence, in shipout order), and needs to do so, because a number of its properties are only available at that point.

However, even if this method will buy us a correct check for a regular float on a regular page (which, to be fair, is a good result), it is not difficult do conceive situations in which this sequence may not be meaningful, or even correct, for the case. A number of cases which may do so are: two column documents, text wrapping, scaling, overlays, etc. (I don't know if those make the method fail, I just don't know if they don't). Therefore, the `labelseq` should be taken as a *proxy* and not fully reliable, meaning that the user should be watchful of its results.

For this reason, zref-check provides an easy way to do so, by allowing specific control of the messaging style of the checks which do within page comparisons though the option `onpage`. The concern is not really with false positives (getting a warning when it was not due), but with false negatives (not getting a warning when it was due). Hence, setting `onpage` to `msg` (or to `obeydraft` or `obeyfinal` if that's part of your workflow) at a final typesetting stage provides a way to easily identify all cases of such checks (failing or passing), and double-check them. In case the test is passing though, the message is different from that of a failing check, to quickly convey why you are getting the message. This option can also be set at the local level, if the page in question is known to be problematic, or just atypical.

8.3 Sectioning checks

The information used by sectioning checks is provided by means of dedicated counters for chapters and sections, similarly as standard counters for them, but which are stepped and reset regardless of whether these sectioning commands are numbered or not (that is, starred or not). And this for two reasons. First, we don't need the absolute counter value to be able to make the kind of relative statement we want to do here. Second, this allows us to have these checks work for numbered and unnumbered sectioning commands without having to worry about how those are used within the document.

The caveat is that the package does this by hooking into `\chapter` and `\section`, which poses two restrictions for the proper working of these checks. First, we are using the new hook system for this, as provided by ltcmdhooks, which means a recent LaTeX kernel is required. Second, since we are hooking into `\chapter` and `\section`, these checks presume these commands are being used by the document class for this purpose (either directly, or internally as, for example, KOMA-Script's `\addchap` and `\addsec` do). If that's not the case, additional setup may be required for these checks to work as expected.

9 Implementation

Start the DocStrip guards.

```
⟨*package⟩
```

Identify the internal prefix (LaTeX3 DocStrip convention).

```
⟨@@=zrefcheck⟩
```

9.1 Initial setup

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

```
\providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
\IfFormatAtLeastTF{2021-06-01}
  {}
  {%
    \PackageError{zref-check}{LaTeX kernel too old}
      {%
        'zref-check' requires a LaTeX kernel newer than 2021-06-01.%
        \MessageBreak Loading will abort!%
      }%
    \endinput
  }%
```

```
\ProvidesExplPackage {zref-check} {2021-07-27} {0.1.0-alpha}
  {Flexible cross-references with contextual checks based on zref}
```

9.2 Dependencies

```
\RequirePackage { zref-user }
\RequirePackage { zref-abspage }
\RequirePackage { ifdraft }
```

9.3 zref setup

`\g__zrefcheck_abschap_int`
`\g__zrefcheck_abssec_int`

Provide absolute counters for section and chapter, and respective zref properties, so that we can make checks about relation of chapters/sections regardless of internal counters, since we don't get those for the unnumbered (starred) ones. About the proper place to make the hooks for this purpose, see https://tex.stackexchange.com/q/605533/105447, thanks Ulrike Fischer.

```
\int_new:N \g__zrefcheck_abschap_int
\int_new:N \g__zrefcheck_abssec_int
```

(*End definition for* `\g__zrefcheck_abschap_int` *and* `\g__zrefcheck_abssec_int`.)

If the documentclass does not define `\chapter` the only thing that happens is that the chapter counter is never incremented, and the section one never reset.

```
\AddToHook { cmd / chapter / before }
  {
    \int_gincr:N \g__zrefcheck_abschap_int
    \int_zero:N \g__zrefcheck_abssec_int
  }
\zref@newprop { abschap } [0] { \int_use:N \g__zrefcheck_abschap_int }
\zref@addprop \ZREF@mainlist { abschap }

\AddToHook { cmd / section / before }
  { \int_gincr:N \g__zrefcheck_abssec_int }
\zref@newprop { abssec } [0] { \int_use:N \g__zrefcheck_abssec_int }
\zref@addprop \ZREF@mainlist { abssec }
```

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

```
\zref@newlist { zrefcheck }
\zref@addprops { zrefcheck }
  {
    abspage ,
    abschap ,
    abssec ,
    page
  }
```

9.4 Plumbing

9.4.1 Messages

`\__zrefcheck_message:nnnn`
`\__zrefcheck_message:nnnx`

```
\cs_new:Npn \__zrefcheck_message:nnnn #1#2#3#4
  {
    \use:c { msg_ \l__zrefcheck_msglevel_tl :nnnnn }
      { zref-check } {#1} {#2} {#3} {#4}
  }
\cs_generate_variant:Nn \__zrefcheck_message:nnnn { nnnx }
```

(*End definition for* `\__zrefcheck_message:nnnn`.)

```
\msg_new:nnn { zref-check } { check-failed }
  {
    Failed~check~'#1'~for~label~'#2' \iow_newline:
    on~page~#3~on~input~line~\msg_line_number:.
  }
\msg_new:nnn { zref-check } { double-check }
  {
    Double-check~'#1'~for~label~'#2' \iow_newline:
    on~page~#3~on~input~line~\msg_line_number:.
  }
```

```
\msg_new:nnn { zref-check } { check-missing }
  { Check~'#1'~not~defined~on~input~line~\msg_line_number:. }
\msg_new:nnn { zref-check } { property-undefined }
  { Property~'#1'~not~defined~on~input~line~\msg_line_number:. }
\msg_new:nnn { zref-check } { property-not-in-label }
  { Label~'#1'~has~no~property~'#2'~on~input~line~\msg_line_number:. }
\msg_new:nnn { zref-check } { property-not-integer }
  {
    Property~'#1'~for~label~'#2'~not~an~integer \iow_newline:
    on~input~line~\msg_line_number:.
  }
\msg_new:nnn { zref-check } { hyperref-preamble-only }
  {
    Option~'hyperref'~only~available~in~the~preamble. \iow_newline:
    Use~the~starred~version~of~'\noexpand\zrcheck'~instead.
  }
\msg_new:nnn { zref-check } { missing-hyperref }
  { Missing~'hyperref'~package. \iow_newline: Setting~'hyperref=false'. }
\msg_new:nnn { zref-check } { ignore-document-only }
  {
    Option~'ignore'~only~available~in~the~document. \iow_newline:
    Use~option~'msglevel'~instead.
  }
```

9.4.2 Options

`hyperref` option

\l__zrefcheck_use_hyperref_bool
\l__zrefcheck_warn_hyperref_bool

```
\bool_new:N \l__zrefcheck_use_hyperref_bool
\bool_new:N \l__zrefcheck_warn_hyperref_bool
\keys_define:nn { zref-check }
  {
    hyperref .choice: ,
    hyperref / auto .code:n =
      {
        \bool_set_true:N \l__zrefcheck_use_hyperref_bool
        \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
      } ,
    hyperref / true .code:n =
      {
        \bool_set_true:N \l__zrefcheck_use_hyperref_bool
        \bool_set_true:N \l__zrefcheck_warn_hyperref_bool
      } ,
    hyperref / false .code:n =
      {
        \bool_set_false:N \l__zrefcheck_use_hyperref_bool
        \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
      } ,
    hyperref .default:n = auto
  }
```

(*End definition for* `\l__zrefcheck_use_hyperref_bool` *and* `\l__zrefcheck_warn_hyperref_bool`*.*)

```
\AtBeginDocument
  {
    \@ifpackageloaded { hyperref }
      {
        \bool_if:NT \l__zrefcheck_use_hyperref_bool
          {
            \RequirePackage { zref-hyperref }
            \zref@addprop { zrefcheck } { anchor }
          }
      }
      {
        \bool_if:NT \l__zrefcheck_warn_hyperref_bool
          { \msg_warning:nn { zref-check } { missing-hyperref } }
        \bool_set_false:N \l__zrefcheck_use_hyperref_bool
      }
    \keys_define:nn { zref-check }
      {
        hyperref .code:n =
          { \msg_warning:nn { zref-check } { hyperref-preamble-only } }
      }
  }
```

`msglevel` option

`\l__zrefcheck_msglevel_tl`

```
\tl_new:N \l__zrefcheck_msglevel_tl
\keys_define:nn { zref-check }
  {
    msglevel .choice: ,
    msglevel / warn .code:n =
      { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } } ,
    msglevel / info .code:n =
      { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } } ,
    msglevel / none .code:n =
      { \tl_set:Nn \l__zrefcheck_msglevel_tl { none } } ,
    msglevel / obeydraft .code:n =
      {
        \ifdraft
          { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
          { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
      } ,
    msglevel / obeyfinal .code:n =
      {
        \ifoptionfinal
          { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
          { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
      } ,
```

`ignore`: alias for `msglevel=none`

```
    ignore .code:n =
      { \msg_warning:nn { zref-check } { ignore-document-only } }
  }
```

(*End definition for `\l__zrefcheck_msglevel_tl`.*)

```
\AtBeginDocument
```

```
  {
    \keys_define:nn { zref-check }
      {
        ignore .meta:n =
          { msglevel = none }
      }
  }
```

onpage option

\l__zrefcheck_msgonpage_bool

```
\bool_new:N \l__zrefcheck_msgonpage_bool
\keys_define:nn { zref-check }
  {
    onpage .choice: ,
    onpage / labelseq .code:n =
      {
        \bool_set_false:N \l__zrefcheck_msgonpage_bool
      } ,
    onpage / msg .code:n =
      {
        \bool_set_true:N \l__zrefcheck_msgonpage_bool
      } ,
    onpage / obeydraft .code:n =
      {
        \ifdraft
          { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
          { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
      } ,
    onpage / obeyfinal .code:n =
      {
        \ifoptionfinal
          { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
          { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
      }
  }
```

(*End definition for* `\l__zrefcheck_msgonpage_bool`.)

closerange option

\l__zrefcheck_close_range_int

```
\int_new:N \l__zrefcheck_close_range_int
\keys_define:nn { zref-check }
  {
    closerange .int_set:N = \l__zrefcheck_close_range_int ,
  }
```

(*End definition for* `\l__zrefcheck_close_range_int`.)

Set load-time default values

```
\keys_set:nn { zref-check }
  {
    hyperref   = auto ,
    msglevel   = warn ,
    onpage     = labelseq ,
    closerange = 5
  }
```

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

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

\zrchecksetup Provide \zrchecksetup.

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

(*End definition for* \zrchecksetup. *This function is documented on page 4.*)

9.4.3 Position on page

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

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

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

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

`\g__zrefcheck_auxfile_lblseq_prop`

```
\prop_new:N \g__zrefcheck_auxfile_lblseq_prop
```

(*End definition for* `\g__zrefcheck_auxfile_lblseq_prop`.)

```
\tl_set:Nn \g_tmpa_tl { \c_sys_jobname_str .aux }
\file_if_exist:nT { \g_tmpa_tl }
  {
```

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.

```
    \ior_open:Nn \g_tmpa_ior { \g_tmpa_tl }
    \group_begin:
      \int_zero:N \l_tmpa_int
      \tl_clear:N \l_tmpa_tl
      \tl_clear:N \l_tmpb_tl
      \bool_set_false:N \l_tmpa_bool
      \ior_map_variable:NNn \g_tmpa_ior \l_tmpa_tl
        {
          \tl_map_variable:NNn \l_tmpa_tl \l_tmpb_tl
            {
              \tl_if_eq:NnTF \l_tmpb_tl { \zref@newlabel }
                {
```

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

```
                  \bool_set_true:N \l_tmpa_bool
                }
                {
                  \bool_if:NTF \l_tmpa_bool
```

```
                      {
                        \bool_set_false:N \l_tmpa_bool
                        \int_incr:N \l_tmpa_int
                        \prop_gput:Nxx \g__zrefcheck_auxfile_lblseq_prop
                          { \l_tmpb_tl } { \int_use:N \l_tmpa_int }
                      }
                      {
```

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.

```
                        \tl_map_break:
                      }
                  }
              }
          }
    \group_end:
    \ior_close:N \g_tmpa_ior
  }
```

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

9.4.4 Counter

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

```
\begingroup
  \let \@addtoreset \ltx@gobbletwo
  \newcounter { zrefcheck }
\endgroup
\setcounter { zrefcheck } { 0 }
```

9.4.5 Label formats

__zrefcheck_check_lblfmt:n

```
\__zrefcheck_check_lblfmt:n {⟨check id int⟩}
```

```
\cs_new:Npn \__zrefcheck_check_lblfmt:n #1 { zrefcheck@ \int_use:N #1 }
```

(*End definition for* `\__zrefcheck_check_lblfmt:n`.)

__zrefcheck_end_lblfmt:n

```
\__zrefcheck_end_lblfmt:n {⟨label⟩}
```

```
\cs_new:Npn \__zrefcheck_end_lblfmt:n #1 { #1 @zrefcheck }
```

(*End definition for* `\__zrefcheck_end_lblfmt:n`.)

9.4.6 Property values

\zrefcheck_get_astl:nnn

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

We cannot use the "return value" of `\__zrefcheck_get_astl:nnn` or `\__zrefcheck_get_asint:nnn` directly, because we need to use the retrieved property values as arguments in the checks, however we use here a number of non-expandable operations. Hence, we receive a local `tl/int` variable as third argument and set that, so that it is available (and expandable) at the place of use. For this reason, we do not group here, because we are passing a local variable around, but it is expected this function will be called within a group.

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

```
\zrefcheck_get_astl:nnn {⟨label⟩} {⟨prop⟩} {⟨tl var⟩}
```

```
\cs_new:Npn \zrefcheck_get_astl:nnn #1#2#3
  {
    \tl_clear:N #3
    \tl_if_eq:nnTF {#2} { lblseq }
      {
        \prop_get:NnNF \g__zrefcheck_auxfile_lblseq_prop {#1} #3
          {
            \msg_warning:nnnn { zref-check }
              { property-not-in-label } {#1} {#2}
          }
      }
      {
```

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

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

```
          \zref@ifrefundefined {#1}
            {}
            {
              \zref@ifpropundefined {#2}
                { \msg_warning:nnnn { zref-check } { property-undefined } {#2} }
                {
                  \zref@ifrefcontainsprop {#1} {#2}
                    {
                      \tl_set:Nx #3
                        { \zref@extractdefault {#1} {#2} { \c_empty_tl } }
                    }
                    {
                      \msg_warning:nnnn
                        { zref-check } { property-not-in-label } {#1} {#2}
                    }
                }
            }
        }
  }
```

(*End definition for* `\zrefcheck_get_astl:nnn`.)

`\l__zrefcheck_integer_bool`

`\zrefcheck_get_asint:nnn` is a very convenient wrapper around the more general `\zrefcheck_get_astl:nnn`, since almost always we'll be wanting to compare numbers in the checks. However, it is quite hard for it to ensure an integer is *always* returned in the case of errors. And those do occur, even in a well structured document (e.g., in a first round of compilation). To complicate things, the L3 integer predicates are *very* sensitive to receiving any other kind of data, and they *scream*. To handle this `\zrefcheck_get_asint:nnn` uses `\l__zrefcheck_integer_bool` to signal if an integer could not be returned. To use this function always set `\l__zrefcheck_integer_bool` to true first, then call it as much as you need. If any of these calls got is returning anything which is not an integer, `\l__zrefcheck_integer_bool` will have been set to false, and you should check that this hasn't happened before actually comparing the integers (`\bool_lazy_and:nnTF` is your friend).

```
\bool_new:N \l__zrefcheck_integer_bool
```

(*End definition for* `\l__zrefcheck_integer_bool`.)

`\l__zrefcheck_propval_tl`

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

(*End definition for* `\l__zrefcheck_propval_tl`.)

`\zrefcheck_get_asint:nnn`

`\zrefcheck_get_asint:nnn {⟨label⟩} {⟨prop⟩} {⟨int var⟩}`

```
\cs_new:Npn \zrefcheck_get_asint:nnn #1#2#3
  {
    \zrefcheck_get_astl:nnn {#1} {#2} { \l__zrefcheck_propval_tl }
    \__zrefcheck_is_integer:nTF { \l__zrefcheck_propval_tl }
      {
```

Make it an integer data type.

```
        \int_set:Nn #3 { \int_eval:n { \l__zrefcheck_propval_tl } }
      }
      {
        \bool_set_false:N \l__zrefcheck_integer_bool
        \zref@ifrefundefined {#1}
```

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

```
          { }
          {
            \msg_warning:nnnn { zref-check }
              { property-not-integer } {#2} {#1}
          }
      }
  }
```

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

`\__zrefcheck_is_integer:n` Thanks egreg: https://tex.stackexchange.com/a/244405, also see https://tex.stackexchange.com/a/19769. But I replaced `\__int_to_roman:w` with the equivalent `\tex_romannumeral:D`, since the use of the former raised a warning from `l3build doc`, complaining of the use of an internal function from the `int` module. And I'm using `\tl_if_empty:oTF` instead of `\tl_if_blank:oTF` as in egreg's answer, since `\romannumeral` is defined so that "the expansion is empty if the number is zero or negative", not "blank". A couple of comments about this technique: `\tex_romannumeral:D` ignores space tokens and explicit signs $+$ and $-$ in the expansion and hence it can only be used to test positive integers; also the technique cannot distinguish whether it received an empty argument or if "the expansion was empty" as a result of receiving a zero or negative number as argument, so this must also be controlled for since, in our use case, this may happen.

```
\prg_new_conditional:Npnn \__zrefcheck_is_integer:n #1 { p, T , F , TF }
  {
    \tl_if_empty:oTF {#1}
      { \prg_return_false: }
      {
        \tl_if_empty:oTF { \tex_romannumeral:D -0#1 }
          { \prg_return_true:  }
          { \prg_return_false: }
      }
  }
```

(*End definition for* `\__zrefcheck_is_integer:n`.)

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

```
\prg_new_protected_conditional:Npnn \__zrefcheck_is_integer_rgx:n #1 { TF }
  {
    \regex_match:nnTF { \A\d+\Z } {#1}
      { \prg_return_true:  }
      { \prg_return_false: }
  }
\prg_generate_conditional_variant:Nnn \__zrefcheck_is_integer_rgx:n { x } { TF }
```

(*End definition for* `\__zrefcheck_is_integer_rgx:n`.)

9.5 User interface

9.5.1 \zrcheck

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

```
\zrcheck⟨*⟩[⟨options⟩]{⟨labels⟩}[⟨checks⟩]{⟨text⟩}
```

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

(*End definition for* \zrcheck. *This function is documented on page 4.*)

\g__zrefcheck_id_int
\l__zrefcheck_checkbeg_tl
\l__zrefcheck_checkend_tl
\l__zrefcheck_link_label_tl
\l__zrefcheck_link_anchor_tl
\l__zrefcheck_link_star_tl

```
\int_new:N \g__zrefcheck_id_int
\tl_new:N \l__zrefcheck_checkbeg_tl
\tl_new:N \l__zrefcheck_checkend_tl
\tl_new:N \l__zrefcheck_link_label_tl
\tl_new:N \l__zrefcheck_link_anchor_tl
\bool_new:N \l__zrefcheck_link_star_tl
```

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

__zrefcheck_zrcheck:nnnnn An intermediate internal function, which places {⟨*labels*⟩} as first argument, so that it can be protected by \zref@wrapper@babel. This is more or less what the definition of \zref in zref-user.sty does for this.

```
\__zrefcheck_zrcheck:nnnnn {⟨labels⟩} {⟨*⟩} {⟨options⟩} {⟨checks⟩} {⟨text⟩}
```

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

Process local options.

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

Names of the labels for this zrefcheck call.

```
      \int_gincr:N \g__zrefcheck_id_int
      \tl_set:Nx \l__zrefcheck_checkbeg_tl
        { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
      \tl_set:Nx \l__zrefcheck_checkend_tl
        { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
```

Set checkbeg label.

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

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

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

If the reference is undefined, just typeset.

```
          {#5}
          {
            \bool_if:nTF
              {
                \l__zrefcheck_use_hyperref_bool &&
                ! \l__zrefcheck_link_star_tl
              }
              {
                \exp_args:Nx \zrefcheck_get_astl:nnn
                  { \l__zrefcheck_link_label_tl }
                  { anchor } { \l__zrefcheck_link_anchor_tl }
                \hyperlink { \l__zrefcheck_link_anchor_tl } {#5}
              }
              {#5}
          }
```

Set checkend label.

```
        \zref@labelbylist { \l__zrefcheck_checkend_tl } { zrefcheck }
```

Check definition. Note that, even if not indicated in zref's documentation by the usual 'babel' markup, `\zref@refused` *is* protected by `\zref@wrapper@babel`.

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

Run the checks.

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

(*End definition for* `\__zrefcheck_zrcheck:nnnnn`.)

9.5.2 Targets

\zrctarget

```
\zrctarget{⟨label⟩}{⟨text⟩}
```

```
\NewDocumentCommand \zrctarget { m +m }
  {
    \refstepcounter { zrefcheck }
    \zref@wrapper@babel \zref@labelbylist {#1} { zrefcheck }
    #2
    \zref@wrapper@babel
      \zref@labelbylist { \__zrefcheck_end_lblfmt:n {#1} } { zrefcheck }
  }
```

(*End definition for* `\zrctarget`. *This function is documented on page 4.*)

zrcregion

```
\begin{zrcregion}{⟨label⟩}
  ...
\end{zrcregion}
```

```
\NewDocumentEnvironment {zrcregion} { m }
  {
    \refstepcounter { zrefcheck }
    \zref@wrapper@babel \zref@labelbylist {#1} { zrefcheck }
  }
  {
    \zref@wrapper@babel
```

```
      \zref@labelbylist { \__zrefcheck_end_lblfmt:n {#1} } { zrefcheck }
  }
```

(*End definition for* `zrcregion`*. This function is documented on page 4.*)

9.6 Checks

What is needed define a zref-check check?

First, a conditional function defined with:

`\prg_new_conditional:Npnn \__zrefcheck_check_⟨check⟩:nn #1#2 { F }`
where ⟨*check*⟩ is the name of the check, the first argument is the `{`⟨*label*⟩`}` and the second the `{`⟨*reference*⟩`}`. The existence of the check is verified by the existence of the function with this name-scheme (and signatures). As usual, this function must return either `\prg_return_true:` or `\prg_return_false:`. Of course, you can define other variants if you need them internally, it is just that what the package does expect and verifies is the existence of the `:nnF` variant.

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

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

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

9.6.1 Running

`\__zrefcheck_run_checks:nnn`
`\__zrefcheck_run_checks:nnV`

`\__zrefcheck_run_checks:nnn {⟨checks⟩} {⟨labels⟩} {⟨reference⟩}`

```
\cs_new:Npn \__zrefcheck_run_checks:nnn #1#2#3
  {
    \group_begin:
      \tl_map_inline:nn {#2}
        {
          \tl_map_inline:nn {#1}
            { \__zrefcheck_do_check:nnn {####1} {##1} {#3} }
        }
    \group_end:
  }
\cs_generate_variant:Nn \__zrefcheck_run_checks:nnn { nnV }
```

(*End definition for* `\__zrefcheck_run_checks:nnn`*.*)

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

```
\bool_new:N \l__zrefcheck_passedcheck_bool
\bool_new:N \l__zrefcheck_onpage_bool
\seq_new:N \c__zrefcheck_onpage_checks_seq
\seq_set_from_clist:Nn \c__zrefcheck_onpage_checks_seq
  { above , below , before , after }
```

(*End definition for* `\l__zrefcheck_passedcheck_bool`*,* `\l__zrefcheck_onpage_bool`*, and* `\c__zrefcheck_-onpage_checks_seq`*.*)

Variant not provided by expl3.

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

`\__zrefcheck_do_check:nnn`

```
\__zrefcheck_do_check:nnn {⟨check⟩} {⟨label beg⟩} {⟨reference beg⟩}
```

```
\cs_new:Npn \__zrefcheck_do_check:nnn #1#2#3
  {
    \group_begin:
```

⟨*label beg*⟩ may be defined or not, it is arbitrary user input. Whether this is the case is checked in `\__zrefcheck_zrcheck:nnnnn`, and due warning already ensues. And there is no point in checking “relative position” of an undefined label. Hence, in the absence of `#2`, we do nothing at all here.

```
      \zref@ifrefundefined {#2}
        {}
        {
          \bool_set_true:N \l__zrefcheck_passedcheck_bool
          \bool_set_false:N \l__zrefcheck_onpage_bool
          \cs_if_exist:cTF { __zrefcheck_check_ #1 :nnF }
            {
```

“label beg” vs “reference beg”.

```
              \use:c { __zrefcheck_check_ #1 :nnF }
                {#2} {#3}
                { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
```

“label beg” vs “reference end”.

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

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

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

“label end” vs “reference beg”.

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

“label end” vs “reference end”.

```
                  \exp_args:Nnoo \use:c { __zrefcheck_check_ #1 :nnF }
                    { \__zrefcheck_end_lblfmt:n {#2} }
                    { \__zrefcheck_end_lblfmt:n {#3} }
                    { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
                }
```

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.

```
              \seq_if_in:NnT \c__zrefcheck_onpage_checks_seq {#1}
                {
                  \__zrefcheck_check_thispage:nnT
```

```
         {#2} {#3}
         { \bool_set_true:N \l__zrefcheck_onpage_bool }
        \__zrefcheck_check_thispage:nnT
         {#2} { \__zrefcheck_end_lblfmt:n {#3} }
         { \bool_set_true:N \l__zrefcheck_onpage_bool }
        \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
         {}
         {
          \__zrefcheck_check_thispage:nnT
           { \__zrefcheck_end_lblfmt:n {#2} } {#3}
           { \bool_set_true:N \l__zrefcheck_onpage_bool }
          \__zrefcheck_check_thispage:nnT
           { \__zrefcheck_end_lblfmt:n {#2} }
           { \__zrefcheck_end_lblfmt:n {#3} }
           { \bool_set_true:N \l__zrefcheck_onpage_bool }
         }
       }
      \bool_if:NTF \l__zrefcheck_passedcheck_bool
       {
        \bool_if:nT
         {
          \l__zrefcheck_msgonpage_bool &&
          \l__zrefcheck_onpage_bool
         }
         {
          \__zrefcheck_message:nnnx { double-check } {#1} {#2}
           { \zref@extractdefault {#3} {page} {'unknown'} }
         }
       }
       {
        \__zrefcheck_message:nnnx { check-failed } {#1} {#2}
         { \zref@extractdefault {#3} {page} {'unknown'} }
       }
     }
     { \msg_warning:nnn { zref-check } { check-missing } {#1} }
   }
 \group_end:
}
```

(*End definition for* `\__zrefcheck_do_check:nnn`.)

9.6.2 Conditionals

`\l__zrefcheck_lbl_int`
`\l__zrefcheck_ref_int`
`\l__zrefcheck_lbl_b_int`
`\l__zrefcheck_ref_b_int`

More readable scratch variables for the tests.

```
\int_new:N \l__zrefcheck_lbl_int
\int_new:N \l__zrefcheck_ref_int
\int_new:N \l__zrefcheck_lbl_b_int
\int_new:N \l__zrefcheck_ref_b_int
```

(*End definition for* `\l__zrefcheck_lbl_int` *and others.*)

This page

__zrefcheck_check_thispage:nn

```
\prg_new_conditional:Npnn \__zrefcheck_check_thispage:nn #1#2 { T , F , TF }
  {
    \group_begin:
      \bool_set_true:N \l__zrefcheck_integer_bool
      \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
      \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
      \bool_lazy_and:nnTF
        { \l__zrefcheck_integer_bool }
        {
          \int_compare_p:nNn
            { \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.

```
            ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
            ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
        }
        { \group_insert_after:N \prg_return_true:  }
        { \group_insert_after:N \prg_return_false: }
    \group_end:
  }
```

(*End definition for __zrefcheck_check_thispage:nn.*)

On page

__zrefcheck_check_above:nn
__zrefcheck_check_below:nn

```
\prg_new_conditional:Npnn \__zrefcheck_check_above:nn #1#2 { F , TF }
  {
    \group_begin:
      \__zrefcheck_check_thispage:nnTF {#1} {#2}
        {
          \bool_set_true:N \l__zrefcheck_integer_bool
          \zrefcheck_get_asint:nnn {#1} { lblseq } { \l__zrefcheck_lbl_int }
          \zrefcheck_get_asint:nnn {#2} { lblseq } { \l__zrefcheck_ref_int }
          \bool_lazy_and:nnTF
            { \l__zrefcheck_integer_bool }
            {
              \int_compare_p:nNn
                { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
              ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
              ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
            }
            { \group_insert_after:N \prg_return_true:  }
            { \group_insert_after:N \prg_return_false: }
        }
        { \group_insert_after:N \prg_return_false: }
    \group_end:
  }
\prg_new_conditional:Npnn \__zrefcheck_check_below:nn #1#2 { F , TF }
  {
    \__zrefcheck_check_thispage:nnTF {#1} {#2}
      {
        \__zrefcheck_check_above:nnTF {#1} {#2}
          { \prg_return_false: }
          { \prg_return_true:  }
      }
      { \prg_return_false: }
  }
```

(*End definition for* `\__zrefcheck_check_above:nn` *and* `\__zrefcheck_check_below:nn`*.*)

Before / After

`\__zrefcheck_check_before:nn`
`\__zrefcheck_check_after:nn`

```
\prg_new_conditional:Npnn \__zrefcheck_check_before:nn #1#2 { F }
  {
    \__zrefcheck_check_pagesbefore:nnTF {#1} {#2}
      { \prg_return_true: }
      {
        \__zrefcheck_check_above:nnTF {#1} {#2}
          { \prg_return_true:  }
          { \prg_return_false: }
      }
  }
\prg_new_conditional:Npnn \__zrefcheck_check_after:nn #1#2 { F }
  {
    \__zrefcheck_check_pagesafter:nnTF {#1} {#2}
      { \prg_return_true: }
      {
        \__zrefcheck_check_below:nnTF {#1} {#2}
          { \prg_return_true:  }
          { \prg_return_false: }
      }
  }
```

(*End definition for* `\__zrefcheck_check_before:nn` *and* `\__zrefcheck_check_after:nn`*.*)

Pages

`\__zrefcheck_check_nextpage:nn`
`\__zrefcheck_check_prevpage:nn`
`\__zrefcheck_check_pagesbefore:nn`
`\__zrefcheck_check_ppbefore:nn`
`\__zrefcheck_check_pagesafter:nn`
`\__zrefcheck_check_ppafter:nn`
`\__zrefcheck_check_facing:nn`

```
\prg_new_conditional:Npnn \__zrefcheck_check_nextpage:nn #1#2 { F }
  {
    \group_begin:
      \bool_set_true:N \l__zrefcheck_integer_bool
      \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
      \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
      \bool_lazy_and:nnTF
        { \l__zrefcheck_integer_bool }
        {
          \int_compare_p:nNn
            { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
          ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
          ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
        }
      { \group_insert_after:N \prg_return_true:  }
      { \group_insert_after:N \prg_return_false: }
  \group_end:
}
\prg_new_conditional:Npnn \__zrefcheck_check_prevpage:nn #1#2 { F }
{
  \group_begin:
    \bool_set_true:N \l__zrefcheck_integer_bool
    \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
    \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
    \bool_lazy_and:nnTF
      { \l__zrefcheck_integer_bool }
      {
        \int_compare_p:nNn
          { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
        ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
        ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
      }
      { \group_insert_after:N \prg_return_true:  }
      { \group_insert_after:N \prg_return_false: }
  \group_end:
}
\prg_new_conditional:Npnn \__zrefcheck_check_pagesbefore:nn #1#2 { F , TF }
{
  \group_begin:
    \bool_set_true:N \l__zrefcheck_integer_bool
    \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
    \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
    \bool_lazy_and:nnTF
      { \l__zrefcheck_integer_bool }
      {
        \int_compare_p:nNn
          { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
        ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
        ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
      }
      { \group_insert_after:N \prg_return_true:  }
      { \group_insert_after:N \prg_return_false: }
  \group_end:
}
\cs_new_eq:NN \__zrefcheck_check_ppbefore:nnF \__zrefcheck_check_pagesbefore:nnF
\prg_new_conditional:Npnn \__zrefcheck_check_pagesafter:nn #1#2 { F , TF }
{
  \group_begin:
    \bool_set_true:N \l__zrefcheck_integer_bool
    \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
    \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
    \bool_lazy_and:nnTF
      { \l__zrefcheck_integer_bool }
      {
        \int_compare_p:nNn
          { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
        ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
        ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
        }
        { \group_insert_after:N \prg_return_true:  }
        { \group_insert_after:N \prg_return_false: }
    \group_end:
  }
\cs_new_eq:NN \__zrefcheck_check_ppafter:nnF \__zrefcheck_check_pagesafter:nnF
\prg_new_conditional:Npnn \__zrefcheck_check_facing:nn #1#2 { F }
  {
    \group_begin:
      \bool_set_true:N \l__zrefcheck_integer_bool
      \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
      \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
      \bool_lazy_and:nnTF
        { \l__zrefcheck_integer_bool }
        {
```

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

```
          \legacy_if_p:n { @twoside } &&
```

Now we test “facing”.

```
          (
            (
              \int_if_odd_p:n { \l__zrefcheck_ref_int } &&
              \int_compare_p:nNn
                { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 }
            ) ||
            (
              \int_if_even_p:n { \l__zrefcheck_ref_int } &&
              \int_compare_p:nNn
                { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 }
            )
          ) &&
          ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
          ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
        }
        { \group_insert_after:N \prg_return_true:  }
        { \group_insert_after:N \prg_return_false: }
    \group_end:
  }
```

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

Close / Far

__zrefcheck_check_close:nn
__zrefcheck_check_far:nn

```
\prg_new_conditional:Npnn \__zrefcheck_check_close:nn #1#2 { F , TF }
  {
    \group_begin:
      \bool_set_true:N \l__zrefcheck_integer_bool
      \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
      \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
      \bool_lazy_and:nnTF
        { \l__zrefcheck_integer_bool }
        {
          \int_compare_p:nNn
```

```
            { \int_abs:n { \l__zrefcheck_lbl_int - \l__zrefcheck_ref_int } }
            <
            { \l__zrefcheck_close_range_int + 1 } &&
          ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
          ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
        }
        { \group_insert_after:N \prg_return_true:  }
        { \group_insert_after:N \prg_return_false: }
    \group_end:
  }
\prg_new_conditional:Npnn \__zrefcheck_check_far:nn #1#2 { F }
  {
    \__zrefcheck_check_close:nnTF {#1} {#2}
      { \prg_return_false: }
      { \prg_return_true:  }
  }
```

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

Chapter

__zrefcheck_check_thischap:nn
__zrefcheck_check_nextchap:nn
__zrefcheck_check_prevchap:nn
__zrefcheck_check_chapsafter:nn
__zrefcheck_check_chapsbefore:nn

```
\prg_new_conditional:Npnn \__zrefcheck_check_thischap:nn #1#2 { F }
  {
    \group_begin:
      \bool_set_true:N \l__zrefcheck_integer_bool
      \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
      \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
      \bool_lazy_and:nnTF
        { \l__zrefcheck_integer_bool }
        {
          \int_compare_p:nNn
            { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int } &&
```

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

```
          ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
          ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
        }
        { \group_insert_after:N \prg_return_true:  }
        { \group_insert_after:N \prg_return_false: }
    \group_end:
  }
\prg_new_conditional:Npnn \__zrefcheck_check_nextchap:nn #1#2 { F }
  {
    \group_begin:
      \bool_set_true:N \l__zrefcheck_integer_bool
      \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
      \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
      \bool_lazy_and:nnTF
        { \l__zrefcheck_integer_bool }
        {
          \int_compare_p:nNn
            { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
          ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
        }
        { \group_insert_after:N \prg_return_true:  }
        { \group_insert_after:N \prg_return_false: }
    \group_end:
  }
\prg_new_conditional:Npnn \__zrefcheck_check_prevchap:nn #1#2 { F }
  {
    \group_begin:
      \bool_set_true:N \l__zrefcheck_integer_bool
      \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
      \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
      \bool_lazy_and:nnTF
        { \l__zrefcheck_integer_bool }
        {
          \int_compare_p:nNn
            { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
          ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
          ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
        }
        { \group_insert_after:N \prg_return_true:  }
        { \group_insert_after:N \prg_return_false: }
    \group_end:
  }
\prg_new_conditional:Npnn \__zrefcheck_check_chapsafter:nn #1#2 { F }
  {
    \group_begin:
      \bool_set_true:N \l__zrefcheck_integer_bool
      \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
      \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
      \bool_lazy_and:nnTF
        { \l__zrefcheck_integer_bool }
        {
          \int_compare_p:nNn
            { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
          ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
        }
        { \group_insert_after:N \prg_return_true:  }
        { \group_insert_after:N \prg_return_false: }
    \group_end:
  }
\prg_new_conditional:Npnn \__zrefcheck_check_chapsbefore:nn #1#2 { F }
  {
    \group_begin:
      \bool_set_true:N \l__zrefcheck_integer_bool
      \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
      \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
      \bool_lazy_and:nnTF
        { \l__zrefcheck_integer_bool }
        {
          \int_compare_p:nNn
          { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
        ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
        ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
      }
      { \group_insert_after:N \prg_return_true:  }
      { \group_insert_after:N \prg_return_false: }
  \group_end:
}
```

(*End definition for* `\__zrefcheck_check_thischap:nn` *and others.*)

Section

__zrefcheck_check_thissec:nn
__zrefcheck_check_nextsec:nn
__zrefcheck_check_prevsec:nn
__zrefcheck_check_secsafter:nn
__zrefcheck_check_secsbefore:nn

```
\prg_new_conditional:Npnn \__zrefcheck_check_thissec:nn #1#2 { F }
  {
    \group_begin:
      \bool_set_true:N \l__zrefcheck_integer_bool
      \zrefcheck_get_asint:nnn {#1} { abssec  } { \l__zrefcheck_lbl_int }
      \zrefcheck_get_asint:nnn {#2} { abssec  } { \l__zrefcheck_ref_int }
      \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
      \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
      \bool_lazy_and:nnTF
        { \l__zrefcheck_integer_bool }
        {
          \int_compare_p:nNn
            { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
          \int_compare_p:nNn
            { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int } &&
```

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

```
          ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
          ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
        }
        { \group_insert_after:N \prg_return_true:  }
        { \group_insert_after:N \prg_return_false: }
    \group_end:
  }
\prg_new_conditional:Npnn \__zrefcheck_check_nextsec:nn #1#2 { F }
  {
    \group_begin:
      \bool_set_true:N \l__zrefcheck_integer_bool
      \zrefcheck_get_asint:nnn {#1} { abssec  } { \l__zrefcheck_lbl_int }
      \zrefcheck_get_asint:nnn {#2} { abssec  } { \l__zrefcheck_ref_int }
      \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
      \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
      \bool_lazy_and:nnTF
        { \l__zrefcheck_integer_bool }
        {
          \int_compare_p:nNn
          { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
        \int_compare_p:nNn
          { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
        ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
      }
      { \group_insert_after:N \prg_return_true:  }
      { \group_insert_after:N \prg_return_false: }
    \group_end:
  }
\prg_new_conditional:Npnn \__zrefcheck_check_prevsec:nn #1#2 { F }
  {
    \group_begin:
      \bool_set_true:N \l__zrefcheck_integer_bool
      \zrefcheck_get_asint:nnn {#1} { abssec  } { \l__zrefcheck_lbl_int }
      \zrefcheck_get_asint:nnn {#2} { abssec  } { \l__zrefcheck_ref_int }
      \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
      \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
      \bool_lazy_and:nnTF
        { \l__zrefcheck_integer_bool }
        {
          \int_compare_p:nNn
            { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
          \int_compare_p:nNn
            { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
          ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
          ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
        }
        { \group_insert_after:N \prg_return_true:  }
        { \group_insert_after:N \prg_return_false: }
    \group_end:
  }
\prg_new_conditional:Npnn \__zrefcheck_check_secsafter:nn #1#2 { F }
  {
    \group_begin:
      \bool_set_true:N \l__zrefcheck_integer_bool
      \zrefcheck_get_asint:nnn {#1} { abssec  } { \l__zrefcheck_lbl_int }
      \zrefcheck_get_asint:nnn {#2} { abssec  } { \l__zrefcheck_ref_int }
      \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
      \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
      \bool_lazy_and:nnTF
        { \l__zrefcheck_integer_bool }
        {
          \int_compare_p:nNn
            { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
          \int_compare_p:nNn
            { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
          ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
        }
        { \group_insert_after:N \prg_return_true:  }
        { \group_insert_after:N \prg_return_false: }
    \group_end:
  }
\prg_new_conditional:Npnn \__zrefcheck_check_secsbefore:nn #1#2 { F }
  {
    \group_begin:
      \bool_set_true:N \l__zrefcheck_integer_bool
      \zrefcheck_get_asint:nnn {#1} { abssec  } { \l__zrefcheck_lbl_int }
      \zrefcheck_get_asint:nnn {#2} { abssec  } { \l__zrefcheck_ref_int }
      \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
      \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
      \bool_lazy_and:nnTF
        { \l__zrefcheck_integer_bool }
        {
          \int_compare_p:nNn
            { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
          \int_compare_p:nNn
            { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
          ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
          ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
        }
        { \group_insert_after:N \prg_return_true:  }
        { \group_insert_after:N \prg_return_false: }
    \group_end:
  }
```

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

```
⟨/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.