The zref-check package implementation[*]

Gustavo Barros[†]

2023-08-13

Contents

[*]This file describes v0.3.4, released 2023-08-13.
[†]https://github.com/gusbrs/zref-check

1 Initial setup

Start the DocStrip guards.

```
⟨*package⟩
```

Identify the internal prefix (LaTeX3 DocStrip convention).

```
⟨@@=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 LaTeX kernel.

```
\def\zrefcheck@required@kernel{2021-06-01}
\NeedsTeXFormat{LaTeX2e}[\zrefcheck@required@kernel]
\providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
\IfFormatAtLeastTF{\zrefcheck@required@kernel}
  {}
  {%
    \PackageError{zref-check}{LaTeX kernel too old}
      {%
        'zref-check' requires a LaTeX kernel \zrefcheck@required@kernel\space or newer.%
      }%
  }%
```

Identify the package.

```
\ProvidesExplPackage {zref-check} {2023-08-13} {0.3.4}
  {Flexible cross-references with contextual checks based on zref}
```

2 Dependencies

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

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

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

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

```
\zref@newlist { zrefcheck-check }
\zref@addprops { zrefcheck-check }
  {
    page , % for messages
    abspage ,
    zc@abschap ,
    zc@abssec
  }
\zref@newlist { zrefcheck-end }
\zref@addprops { zrefcheck-end }
  {
    abspage ,
    zc@abschap ,
    zc@abssec
  }
```

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

```
\zref@newlist { zrefcheck-zrefvario }
\zref@addprops { zrefcheck-zrefvario }
  {
    page , % for messages
    abspage ,
  }
```

4 Plumbing

4.1 Messages

__zrefcheck_message:nnnn
__zrefcheck_message:nnnx

```
\cs_new_protected: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 of definition for* `\__zrefcheck_message:nnnn`.)

```
\msg_new:nnn { zref-check } { check-failed }
  {
    Check~failed~\msg_line_context:.~
    Failed~check~'#1'~for~label~'#2'~on~page~#3.
  }
\msg_new:nnn { zref-check } { double-check }
```

```
  {
    Same~page~check~\msg_line_context:.~
    Double-check~'#1'~for~label~'#2'~on~page~#3.
  }
\msg_new:nnn { zref-check } { empty-label }
  {
    Check~failed~\msg_line_context:.~
    Failed~check~'#1'~for~empty~label.
  }
\msg_new:nnn { zref-check } { no-checks }
  { No~checks~for~'\iow_char:N\\zcheck'~\msg_line_context:. }

\msg_new:nnn { zref-check } { check-missing }
  { Check~'#1'~not~defined~\msg_line_context:. }
\msg_new:nnn { zref-check } { property-undefined }
  { Property~'#1'~not~defined~\msg_line_context:. }
\msg_new:nnn { zref-check } { property-not-in-label }
  { Label~'#1'~has~no~property~'#2'~\msg_line_context:. }
\msg_new:nnn { zref-check } { property-not-integer }
  { Property~'#1'~for~label~'#2'~not~an~integer~\msg_line_context:. }

\msg_new:nnn { zref-check } { hyperref-preamble-only }
  {
    Option~'hyperref'~only~available~in~the~preamble. \iow_newline:
    Use~the~starred~version~of~'\iow_char:N\\zcheck'~instead.
  }
\msg_new:nnn { zref-check } { missing-hyperref }
  { Missing~'hyperref'~package. \iow_newline: Setting~'hyperref=false'. }
\msg_new:nnn { zref-check } { ignore-ok-document-only }
  {
    Option~'#1'~only~available~in~the~document. \iow_newline:
    Use~option~'msglevel'~instead.
  }
\msg_new:nnn { zref-check } { option-preamble-only }
  { Option~'#1'~is~preamble~only~\msg_line_context:. }
\msg_new:nnn { zref-check } { closerange-not-positive-integer }
  {
    Option~'closerange'~not~a~positive~integer~\msg_line_context:.~
    Using~default~value.
  }
\msg_new:nnn { zref-check } { labelcmd-undefined }
  {
    Control~sequence~named~'#1'~used~in~option~'labelcmd'~is~not~defined.~
    Using~default~value.
  }
\msg_new:nnn { zref-check } { option-deprecated-with-alternative }
  {
    Option~'#1'~has~been~deprecated~\msg_line_context:.\iow_newline:
    Use~'#2'~instead.
  }
\msg_new:nnn { zref-check } { option-deprecated }
  { Option~'#1'~has~been~deprecated~\msg_line_context:. }
\msg_new:nnn { zref-check } { load-time-options }
  {
    'zref-check'~does~not~accept~load-time~options.~
    To~configure~package~options,~use~'\iow_char:N\\zrefchecksetup'.
  }
```

4.2 Integer testing

__zrefcheck_is_integer:n
__zrefcheck_int_to_roman:w

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.

```
\cs_new_eq:NN \__zrefcheck_int_to_roman:w \__int_to_roman:w
\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 { \__zrefcheck_int_to_roman:w -0#1 }
          { \prg_return_true:  }
          { \prg_return_false: }
      }
  }
```

(*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.

```
\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: }
  }
```

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

4.3 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 .initial:n = auto ,
    hyperref .default:n = auto
  }

\AddToHook { begindocument }
  {
    \@ifpackageloaded { hyperref }
      {
        \bool_if:NT \l__zrefcheck_use_hyperref_bool
          { \RequirePackage { zref-hyperref } }
      }
      {
        \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 / infoifdraft .code:n =
      {
        \ifdraft
          { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
          { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
      } ,
    msglevel / warniffinal .code:n =
      {
        \ifoptionfinal
          { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
          { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
      } ,
    msglevel .value_required:n = true ,
    msglevel .initial:n = warn ,
```

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

```
    ignore .code:n =
      { \msg_warning:nnn { zref-check } { ignore-ok-document-only } { ignore } } ,
    ignore .value_forbidden:n = true ,
    ok .code:n =
      { \msg_warning:nnn { zref-check } { ignore-ok-document-only } { ok } } ,
    ok .value_forbidden:n = true ,
  }
```

```
\AddToHook { begindocument }
  {
    \keys_define:nn { zref-check }
      {
        ignore .meta:n = { msglevel = none } ,
        ok .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 / labelseqifdraft .code:n =
      {
        \ifdraft
          { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
          { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
      } ,
    onpage / msgiffinal .code:n =
      {
        \ifoptionfinal
          { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
          { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
      } ,
    onpage .value_required:n = true ,
    onpage .initial:n = labelseq
  }
```

closerange option

`\l__zrefcheck_close_range_int`

```
\int_new:N \l__zrefcheck_close_range_int
\keys_define:nn { zref-check }
  {
    closerange .code:n =
      {
        \__zrefcheck_is_integer_rgx:nTF {#1}
          { \int_set:Nn \l__zrefcheck_close_range_int { \int_eval:n {#1} } }
          {
            \msg_warning:nn { zref-check } { closerange-not-positive-integer }
            \int_set:Nn \l__zrefcheck_close_range_int { 5 }
          }
      } ,
    closerange .value_required:n = true ,
    closerange .initial:n = 5
  }
```

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.

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

\zrefchecksetup Provide \zrefchecksetup.

```
\NewDocumentCommand \zrefchecksetup { m }
  { \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 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 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`

```
\prop_new:N \g__zrefcheck_auxfile_lblseq_prop

\tl_gset: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. 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.

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

4.6 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 of 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 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⟩}
```

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

```
        \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 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).

```
\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*⟩}

```
\cs_new_protected: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_zcheck:nnnnn`.

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

(*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*⟩}

```
\NewDocumentCommand \zcheck { s O { } m m }
  { \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
```

```
\seq_new:N \l__zrefcheck_zcheck_labels_seq
\int_new:N \g__zrefcheck_id_int
\tl_new:N \l__zrefcheck_checkbeg_tl
\tl_new:N \l__zrefcheck_link_label_tl
\tl_new:N \l__zrefcheck_link_anchor_tl
\bool_new:N \l__zrefcheck_link_star_bool
```

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

```
\__zrefcheck_zcheck:nnnn {⟨labels⟩} {⟨*⟩} {⟨checks/options⟩} {⟨text⟩}
```

```
\cs_new_protected:Npn \__zrefcheck_zcheck:nnnn #1#2#3#4
 {
   \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.

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

Names of the labels for this zcheck call.

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

Set checkbeg label.

```
   \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.

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

If the reference is undefined, just typeset.

```
     {#4}
     {
       \bool_if:nTF
         {
           \l__zrefcheck_use_hyperref_bool &&
           ! \l__zrefcheck_link_star_bool
         }
         {
           \exp_args:Nx \zrefcheck_get_astl:nnn
```

```
              { \l__zrefcheck_link_label_tl }
              { anchor } { \l__zrefcheck_link_anchor_tl }
            \hyperlink { \l__zrefcheck_link_anchor_tl } {#4}
          }
          {#4}
      }
```

Set checkend label.

```
    \bool_if:NT \l__zrefcheck_zcheck_end_label_bool
      {
        \zref@labelbylist
          { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
          { zrefcheck-end }
      }
```

Check if ⟨*labels*⟩ are defined.

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

Run the checks.

```
    \__zrefcheck_run_checks:nnx { \l__zrefcheck_zcheck_checks_seq }
      { \l__zrefcheck_zcheck_labels_seq } { \l__zrefcheck_checkbeg_tl }
    \group_end:
  }
```

(*End of definition for* `\__zrefcheck_zcheck:nnnn`.)

5.2 Targets

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

```
\NewDocumentCommand \zctarget { m +m }
  {
```

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.

```
    \group_begin:
    \refstepcounter { zrefcheck }
    \zref@wrapper@babel \zref@label {#1}
    #2
    \tl_if_empty:nF {#2}
      {
        \zref@wrapper@babel
          \zref@labelbylist { \__zrefcheck_end_lblfmt:n {#1} } { zrefcheck-end }
      }
    \group_end:
  }
```

(*End of definition for* `\zctarget`.)

`zcregion`

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

```
\NewDocumentEnvironment {zcregion} { m }
  {
    \refstepcounter { zrefcheck }
    \zref@wrapper@babel \zref@label {#1}
  }
  {
    \zref@wrapper@babel
      \zref@labelbylist { \__zrefcheck_end_lblfmt:n {#1} } { zrefcheck-end }
  }
```

(*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
```

```
\seq_const_from_clist:Nn \c__zrefcheck_single_label_checks_seq
  {
    thischap ,
    prevchap ,
    nextchap ,
    chapsbefore ,
    chapsafter ,
    thissec ,
    prevsec ,
    nextsec ,
    secsbefore ,
    secsafter ,
    manual ,
  }
```

6.2 Setup

```
\l__zrefcheck_zcheck_checks_seq
\l__zrefcheck_end_label_required_bool
```

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

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

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

Then we add the checks to it.

```
\clist_map_inline:nn
  {
    thispage ,
    prevpage ,
    nextpage ,
    facing ,
    otherpage ,
    pagegap ,
    above ,
    below ,
    pagesbefore ,
    ppbefore ,
    pagesafter ,
    ppafter ,
    before ,
    after ,
    thischap ,
    prevchap ,
    nextchap ,
    chapsbefore ,
    chapsafter ,
    thissec ,
    prevsec ,
    nextsec ,
    secsbefore ,
    secsafter ,
    close ,
    far ,
    manual ,
  }
  {
    \keys_define:nn { zref-check / zcheck }
      {
        #1 .code:n =
          {
            \seq_put_right:Nn \l__zrefcheck_zcheck_checks_seq {#1}
            \seq_if_in:NnF \c__zrefcheck_single_label_checks_seq {#1}
              { \bool_set_true:N \l__zrefcheck_zcheck_end_label_bool }
          } ,
        #1 .value_forbidden:n = true ,
     }
  }
```

6.3 Running

__zrefcheck_run_checks:nnn

__zrefcheck_run_checks:nnn {⟨*checks*⟩} {⟨*labels*⟩} {⟨*reference*⟩}

⟨*checks*⟩ are expected to be received as a sequence variable.

```
\cs_new_protected:Npn \__zrefcheck_run_checks:nnn #1#2#3
  {
    \group_begin:
    \seq_map_inline:Nn #2
      {
        \seq_if_empty:NTF #1
          { \__zrefcheck_message:nnnn { no-checks } { } { } { } }
          {
            \seq_map_inline:Nn #1
              { \__zrefcheck_do_check:nnn {####1} {##1} {#3} }
          }
      }
    \group_end:
  }
\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

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

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_protected:Npn \__zrefcheck_do_check:nnn #1#2#3
  {
    \group_begin:
    \bool_set_true:N \l__zrefcheck_passedcheck_bool
    \bool_set_false:N \l__zrefcheck_onpage_bool
    \bool_set_false:N \l__zrefcheck_empty_label_bool
    \cs_if_exist:cTF { __zrefcheck_check_ #1 :nnF }
      {
```

⟨*label beg*⟩ may be defined or not, it is arbitrary user input. Whether this is the case is checked in `\__zrefcheck_zcheck:nnnnn`, and due warning already ensues. 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.

```
        \zref@ifrefundefined {#2}
          {
            \tl_if_empty:nT {#2}
              {
                \bool_set_false:N \l__zrefcheck_passedcheck_bool
                \bool_set_true:N \l__zrefcheck_empty_label_bool
              }
          }
          {
            % ‘‘label beg’’ vs ‘‘reference beg’’.
            \use:c { __zrefcheck_check_ #1 :nnF }
              {#2} {#3}
              { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
            % ‘‘reference end’’ \emph{may} exist or not depending on the
            % checks.
            \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#3} }
              {
                % ‘‘label end’’ \emph{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 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’’ \emph{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 }
                \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#3} }
                  {
                    \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
                      {#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 }
                      }
                  }
              }
          }
      }
      { \msg_warning:nnn { zref-check } { check-missing } {#1} }
    \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'} }
          }
      }
      {
        \bool_if:NTF \l__zrefcheck_empty_label_bool
          { \__zrefcheck_message:nnnn { empty-label } {#1} { } { } }
          {
            \__zrefcheck_message:nnnx { check-failed } {#1} {#2}
              { \zref@extractdefault {#3} {page} {'unknown'} }
          }
      }
    \group_end:
  }
\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.

```
\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
```

6.4.1 This page

`\__zrefcheck_check_thispage:nn`
`\__zrefcheck_check_otherpage:nn`

```
\prg_new_protected_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:
  }
\prg_new_protected_conditional:Npnn \__zrefcheck_check_otherpage:nn #1#2 { T , F , TF }
  {
    \__zrefcheck_check_thispage:nnTF {#1} {#2}
      { \prg_return_false: }
      { \prg_return_true:  }
  }
```

(*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`

```
\prg_new_protected_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_protected_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 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`

```
\prg_new_protected_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_protected_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 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

```
\prg_new_protected_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_protected_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_protected_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_protected_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_protected_conditional:Npnn \__zrefcheck_check_pagegap: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
          { \int_abs:n { \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_protected_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 of definition for* `\__zrefcheck_check_nextpage:nn` *and others.*)

6.4.5 Close / Far

`\__zrefcheck_check_close:nn`
`\__zrefcheck_check_far:nn`

```
\prg_new_protected_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_protected_conditional:Npnn \__zrefcheck_check_far:nn #1#2 { F }
  {
    \__zrefcheck_check_close:nnTF {#1} {#2}
      { \prg_return_false: }
      { \prg_return_true:  }
  }
```

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

6.4.6 Chapter

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

```
\prg_new_protected_conditional:Npnn \__zrefcheck_check_thischap:nn #1#2 { F }
  {
    \group_begin:
    \bool_set_true:N \l__zrefcheck_integer_bool
    \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
    \zrefcheck_get_asint:nnn {#2} { zc@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 `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.

```
        ! \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_protected_conditional:Npnn \__zrefcheck_check_nextchap:nn #1#2 { F }
  {
    \group_begin:
    \bool_set_true:N \l__zrefcheck_integer_bool
    \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
    \zrefcheck_get_asint:nnn {#2} { zc@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_protected_conditional:Npnn \__zrefcheck_check_prevchap:nn #1#2 { F }
  {
    \group_begin:
    \bool_set_true:N \l__zrefcheck_integer_bool
    \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
    \zrefcheck_get_asint:nnn {#2} { zc@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_protected_conditional:Npnn \__zrefcheck_check_chapsafter:nn #1#2 { F }
  {
    \group_begin:
    \bool_set_true:N \l__zrefcheck_integer_bool
    \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
    \zrefcheck_get_asint:nnn {#2} { zc@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_protected_conditional:Npnn \__zrefcheck_check_chapsbefore:nn #1#2 { F }
  {
    \group_begin:
    \bool_set_true:N \l__zrefcheck_integer_bool
    \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_int }
    \zrefcheck_get_asint:nnn {#2} { zc@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 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

```
\prg_new_protected_conditional:Npnn \__zrefcheck_check_thissec:nn #1#2 { F }
  {
    \group_begin:
    \bool_set_true:N \l__zrefcheck_integer_bool
    \zrefcheck_get_asint:nnn {#1} { zc@abssec  } { \l__zrefcheck_lbl_int }
    \zrefcheck_get_asint:nnn {#2} { zc@abssec  } { \l__zrefcheck_ref_int }
    \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
    \zrefcheck_get_asint:nnn {#2} { zc@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 `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.

```
        ! \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_protected_conditional:Npnn \__zrefcheck_check_nextsec:nn #1#2 { F }
  {
    \group_begin:
    \bool_set_true:N \l__zrefcheck_integer_bool
    \zrefcheck_get_asint:nnn {#1} { zc@abssec  } { \l__zrefcheck_lbl_int }
    \zrefcheck_get_asint:nnn {#2} { zc@abssec  } { \l__zrefcheck_ref_int }
    \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
    \zrefcheck_get_asint:nnn {#2} { zc@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_protected_conditional:Npnn \__zrefcheck_check_prevsec:nn #1#2 { F }
  {
    \group_begin:
    \bool_set_true:N \l__zrefcheck_integer_bool
    \zrefcheck_get_asint:nnn {#1} { zc@abssec  } { \l__zrefcheck_lbl_int }
    \zrefcheck_get_asint:nnn {#2} { zc@abssec  } { \l__zrefcheck_ref_int }
    \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
    \zrefcheck_get_asint:nnn {#2} { zc@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_protected_conditional:Npnn \__zrefcheck_check_secsafter:nn #1#2 { F }
  {
    \group_begin:
    \bool_set_true:N \l__zrefcheck_integer_bool
    \zrefcheck_get_asint:nnn {#1} { zc@abssec  } { \l__zrefcheck_lbl_int }
    \zrefcheck_get_asint:nnn {#2} { zc@abssec  } { \l__zrefcheck_ref_int }
    \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
    \zrefcheck_get_asint:nnn {#2} { zc@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_protected_conditional:Npnn \__zrefcheck_check_secsbefore:nn #1#2 { F }
  {
    \group_begin:
    \bool_set_true:N \l__zrefcheck_integer_bool
    \zrefcheck_get_asint:nnn {#1} { zc@abssec  } { \l__zrefcheck_lbl_int }
    \zrefcheck_get_asint:nnn {#2} { zc@abssec  } { \l__zrefcheck_ref_int }
    \zrefcheck_get_asint:nnn {#1} { zc@abschap } { \l__zrefcheck_lbl_b_int }
    \zrefcheck_get_asint:nnn {#2} { zc@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 of definition for* `\__zrefcheck_check_thissec:nn` *and others.*)

6.4.8 Manual

`\__zrefcheck_check_manual:nn`

```
\prg_new_protected_conditional:Npnn \__zrefcheck_check_manual:nn #1#2 { F }
  { \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:`
`\zrefcheck_zcref_run_checks_on_labels:n`

```
\cs_new_protected:Npn \zrefcheck_zcref_beg_label:
  {
    \int_gincr:N \g__zrefcheck_id_int
    \tl_set:Nx \l__zrefcheck_checkbeg_tl
      { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
    \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrefcheck-check }
  }
\cs_new_protected:Npn \zrefcheck_zcref_end_label_maybe:
  {
    \bool_if:NT \l__zrefcheck_zcheck_end_label_bool
```

```
      {
        \zref@labelbylist
          { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
          { zrefcheck-end }
      }
  }
\cs_new_protected:Npn \zrefcheck_zcref_run_checks_on_labels:n #1
  {
    \__zrefcheck_run_checks:nnx
      { \l__zrefcheck_zcheck_checks_seq } {#1} { \l__zrefcheck_checkbeg_tl }
  }
```

(*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`

```
\cs_new_protected:Npn \zrefcheck_zrefvario_label:
  {
    \int_gincr:N \g__zrefcheck_id_int
    \tl_set:Nx \l__zrefcheck_checkbeg_tl
      { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
    \zref@labelbylist { \l__zrefcheck_checkbeg_tl } { zrefcheck-zrefvario }
  }
\cs_new_protected:Npn \zrefcheck_zrefvario_run_check_on_label:nn #1#2
  { \__zrefcheck_do_check:nnV {#1} {#2} \l__zrefcheck_checkbeg_tl }
\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* **??**.)

```
⟨/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.