

The zref-check package*

Gustavo Barros[†]

2021-07-27

Contents

1	Introduction	2
1.1	Hard vs. soft cross-references	3
2	Loading the package	4
3	Dependencies	4
4	User interface	4
5	Checks	5
6	Options	6
7	Label names	6
8	Technique and limitations	7
8.1	Page number checks	7
8.2	Within page checks	7
8.3	Sectioning checks	8
I	zref-check implementation	8
1	Initial setup	8
2	Dependencies	9
3	zref setup	9

*This file describes v0.1.0-alpha, last revised 2021-07-27.

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

4	Plumbing	10
4.1	Messages	10
4.2	Options	10
4.3	Position on page	13
4.4	Counter	16
4.5	Label formats	16
4.6	Property values	16
5	\zrcheck	19
6	Targets	20
7	Checks	21
7.1	Running	21
7.2	Check conditionals	23
7.2.1	This page	23
7.2.2	On page	24
7.2.3	Before / After	25
7.2.4	Pages	25
7.2.5	Close / Far	27
7.2.6	Chapter	28
7.2.7	Section	30
	Index	32

1 Introduction

`zref-check` provides an user interface for making L^AT_EX 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 `zcregion`

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 premiss 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 premiss 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-abspace` are loaded by default. `ifdraft` is also loaded by default. A recent L^AT_EX kernel is required, since we rely on the new hook system from `ltxcmds` 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

<code>\zrcheck</code>	<code>\zrcheck[<i><options></i>]{<i><labels></i>}[<i><checks></i>]{<i><text></i>}</code>
-----------------------	--

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) `zlabels`, 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 it for building hyperlinks, besides there is no need for it to be.

<code>\zrctarget</code>	<code>\zrctarget{<label>}{<text>}</code>
-------------------------	--

Typesets `{<text>}`, as given, and places a pair of `zlabels`, one at the start of `{<text>}`, using `{<label>}` as label name, another one (internal) at the end of `{<text>}`.

<code>zrcregion</code>	<code>\begin{zrcregion}{<label>}</code> ... <code>\end{zrcregion}</code>
------------------------	--

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 `\zrctargets` (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.

<code>\zrchecksetup</code>	<code>\zrchecksetup{<options>}</code>
----------------------------	---

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:

<code>thispage</code>	<code>{<label>}</code> occurs on the same page as <code>\zrcheck</code> .
<code>prevpage</code>	<code>{<label>}</code> occurs on the previous page relative to <code>\zrcheck</code> .
<code>nextpage</code>	<code>{<label>}</code> occurs on the next page relative to <code>\zrcheck</code> .
<code>facing</code>	On a <code>twoside</code> document, both <code>{<label>}</code> and <code>\zrcheck</code> fall onto a double spread, each on one of the two facing pages.
<code>above</code>	<code>{<label>}</code> and <code>\zrcheck</code> are both on the same page, and <code>{<label>}</code> occurs “above” <code>\zrcheck</code> (for how this is inferred, see Section 8.2).
<code>below</code>	<code>{<label>}</code> and <code>\zrcheck</code> are both on the same page, and <code>{<label>}</code> occurs “below” <code>\zrcheck</code> .
<code>pagesbefore</code>	<code>{<label>}</code> occurs on any page before the one of <code>\zrcheck</code> .
<code>ppbefore</code>	Convenience alias for <code>pagesbefore</code> .
<code>pagesafter</code>	<code>{<label>}</code> occurs on any page after the one of <code>\zrcheck</code> .
<code>ppafter</code>	Convenience alias for <code>pagesafter</code> .
<code>before</code>	Either <code>above</code> or <code>pagesbefore</code> .
<code>after</code>	Either <code>below</code> or <code>pagesafter</code> .
<code>thischap</code>	<code>{<label>}</code> occurs on the same chapter as <code>\zrcheck</code> .
<code>prevchap</code>	<code>{<label>}</code> occurs on the previous chapter relative to the one of <code>\zrcheck</code> .
<code>nextchap</code>	<code>{<label>}</code> occurs on the next chapter relative to the one of <code>\zrcheck</code> .
<code>chapsbefore</code>	<code>{<label>}</code> occurs on any chapter before the one of <code>\zrcheck</code> .
<code>chapsafter</code>	<code>{<label>}</code> occurs on any chapter after the one of <code>\zrcheck</code> .
<code>thissec</code>	<code>{<label>}</code> occurs on the same section as <code>\zrcheck</code> .
<code>prevsec</code>	<code>{<label>}</code> occurs on the previous section (of the same chapter) relative to the one of <code>\zrcheck</code> .
<code>nextsec</code>	<code>{<label>}</code> occurs on the next section (of the same chapter) relative to the one of <code>\zrcheck</code> .

- `secsbefore` $\{\langle label \rangle\}$ occurs on any section (of the same chapter) before the one of `\zrcheck`.
- `secsafter` $\{\langle label \rangle\}$ occurs on any section (of the same chapter) after the one of `\zrcheck`.
- `close` $\{\langle label \rangle\}$ 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 $\{\langle labels \rangle\}$. `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 L^AT_EX 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 $\{\langle label \rangle\}$ 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 of 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 to 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 through 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 `ltxcmdhooks`, 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.

File I

zref-check implementation

Start the DocStrip guards.

```

1 <*package>
   Identify the internal prefix ( $\text{\LaTeX}$ 3 DocStrip convention).
2 <@@=zrefcheck>
```

1 Initial setup

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

```

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



```

12     \endinput
13 }%

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

```

2 Dependencies

```

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

```

3 zref setup

`\g__zrefcheck_abschap_int` 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.

```

19 \int_new:N \g__zrefcheck_abschap_int
20 \int_new:N \g__zrefcheck_abssec_int

```

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

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

```

21 \AddToHook { cmd / chapter / before }
22 {
23     \int_gincr:N \g__zrefcheck_abschap_int
24     \int_zero:N \g__zrefcheck_abssec_int
25 }
26 \zref@newprop { abschap } [0] { \int_use:N \g__zrefcheck_abschap_int }
27 \zref@addprop \ZREF@mainlist { abschap }

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

```

32 \zref@newlist { zrefcheck }
33 \zref@addprops { zrefcheck }
34 {
35     abspage ,
36     abschap ,
37     abssec ,
38     page
39 }

```

4 Plumbing

4.1 Messages

```

\__zrefcheck_message:nnnn
\__zrefcheck_message:nnnx
40 \cs_new:Npn \__zrefcheck_message:nnnn #1#2#3#4
41 {
42   \use:c { msg_ \l__zrefcheck_msglevel_tl :nnnnn }
43   { zref-check } {#1} {#2} {#3} {#4}
44 }
45 \cs_generate_variant:Nn \__zrefcheck_message:nnnn { nnnx }

(End definition for \__zrefcheck_message:nnnn.)

46 \msg_new:nnn { zref-check } { check-failed }
47 {
48   Failed-check~'~#1'~for-label~'~#2' \iow_newline:
49   on-page~#3~on-input~line~\msg_line_number:.
50 }
51 \msg_new:nnn { zref-check } { double-check }
52 {
53   Double-check~'~#1'~for-label~'~#2' \iow_newline:
54   on-page~#3~on-input~line~\msg_line_number:.
55 }

56 \msg_new:nnn { zref-check } { check-missing }
57 { Check~'~#1'~not-defined-on-input~line~\msg_line_number:. }
58 \msg_new:nnn { zref-check } { property-undefined }
59 { Property~'~#1'~not-defined-on-input~line~\msg_line_number:. }
60 \msg_new:nnn { zref-check } { property-not-in-label }
61 { Label~'~#1'~has-no-property~'~#2'~on-input~line~\msg_line_number:. }
62 \msg_new:nnn { zref-check } { property-not-integer }
63 {
64   Property~'~#1'~for-label~'~#2'~not-an-integer \iow_newline:
65   on-input~line~\msg_line_number:.
66 }

67 \msg_new:nnn { zref-check } { hyperref-preamble-only }
68 {
69   Option~'hyperref'~only-available-in-the-preamble. \iow_newline:
70   Use-the-starred-version-of~'\noexpand\zrcheck'~instead.
71 }
72 \msg_new:nnn { zref-check } { missing-hyperref }
73 { Missing~'hyperref'~package. \iow_newline: Setting~'hyperref=false'. }
74 \msg_new:nnn { zref-check } { ignore-document-only }
75 {
76   Option~'ignore'~only-available-in-the-document. \iow_newline:
77   Use-option~'msglevel'~instead.
78 }

```

4.2 Options

hyperref option

```

\l_zrefcheck_use_hyperref_bool
\l_zrefcheck_warn_hyperref_bool
79 \bool_new:N \l__zrefcheck_use_hyperref_bool

```

```

80 \bool_new:N \l__zrefcheck_warn_hyperref_bool
81 \keys_define:nn { zref-check }
82 {
83   hyperref .choice: ,
84   hyperref / auto .code:n =
85   {
86     \bool_set_true:N \l__zrefcheck_use_hyperref_bool
87     \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
88   } ,
89   hyperref / true .code:n =
90   {
91     \bool_set_true:N \l__zrefcheck_use_hyperref_bool
92     \bool_set_true:N \l__zrefcheck_warn_hyperref_bool
93   } ,
94   hyperref / false .code:n =
95   {
96     \bool_set_false:N \l__zrefcheck_use_hyperref_bool
97     \bool_set_false:N \l__zrefcheck_warn_hyperref_bool
98   } ,
99   hyperref .default:n = auto
100 }

```

(End definition for \l__zrefcheck_use_hyperref_bool and \l__zrefcheck_warn_hyperref_bool.)

```

101 \AtBeginDocument
102 {
103   \@ifpackageloaded { hyperref }
104   {
105     \bool_if:NT \l__zrefcheck_use_hyperref_bool
106     {
107       \RequirePackage { zref-hyperref }
108       \zref@addprop { zrefcheck } { anchor }
109     }
110   }
111   {
112     \bool_if:NT \l__zrefcheck_warn_hyperref_bool
113     { \msg_warning:nn { zref-check } { missing-hyperref } }
114     \bool_set_false:N \l__zrefcheck_use_hyperref_bool
115   }
116   \keys_define:nn { zref-check }
117   {
118     hyperref .code:n =
119     { \msg_warning:nn { zref-check } { hyperref-preamble-only } }
120   }
121 }

```

msglevel option

\l__zrefcheck_msglevel_tl

```

122 \tl_new:N \l__zrefcheck_msglevel_tl
123 \keys_define:nn { zref-check }
124 {
125   msglevel .choice: ,
126   msglevel / warn .code:n =
127   { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } } ,
128   msglevel / info .code:n =

```

```

129     { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } } ,
130     msglevel / none .code:n =
131     { \tl_set:Nn \l__zrefcheck_msglevel_tl { none } } ,
132     msglevel / obeydraft .code:n =
133     {
134         \ifdraft
135         { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
136         { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
137     } ,
138     msglevel / obeyfinal .code:n =
139     {
140         \ifoptionfinal
141         { \tl_set:Nn \l__zrefcheck_msglevel_tl { warning } }
142         { \tl_set:Nn \l__zrefcheck_msglevel_tl { info } }
143     } ,
144 ignore: alias for msglevel=none
145     ignore .code:n =
146     { \msg_warning:nn { zref-check } { ignore-document-only } }
147 }

```

(End definition for \l__zrefcheck_msglevel_tl.)

```

147 \AtBeginDocument
148 {
149     \keys_define:nn { zref-check }
150     {
151         ignore .meta:n =
152         { msglevel = none }
153     }
154 }

```

onpage option

\l__zrefcheck_msgonpage_bool

```

155 \bool_new:N \l__zrefcheck_msgonpage_bool
156 \keys_define:nn { zref-check }
157 {
158     onpage .choice: ,
159     onpage / labelseq .code:n =
160     {
161         \bool_set_false:N \l__zrefcheck_msgonpage_bool
162     } ,
163     onpage / msg .code:n =
164     {
165         \bool_set_true:N \l__zrefcheck_msgonpage_bool
166     } ,
167     onpage / obeydraft .code:n =
168     {
169         \ifdraft
170         { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
171         { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
172     } ,
173     onpage / obeyfinal .code:n =
174     {
175         \ifoptionfinal

```

```

176         { \bool_set_true:N \l__zrefcheck_msgonpage_bool }
177         { \bool_set_false:N \l__zrefcheck_msgonpage_bool }
178     }
179 }

```

(End definition for `\l__zrefcheck_msgonpage_bool`.)

`closerange` option

`\l__zrefcheck_close_range_int`

```

180 \int_new:N \l__zrefcheck_close_range_int
181 \keys_define:nn { zref-check }
182 {
183     closerange .int_set:N = \l__zrefcheck_close_range_int ,
184 }

```

(End definition for `\l__zrefcheck_close_range_int`.)

Set load-time default values

```

185 \keys_set:nn { zref-check }
186 {
187     hyperref      = auto ,
188     msglevel      = warn ,
189     onpage        = labelseq ,
190     closerange    = 5
191 }

```

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

```

192 \RequirePackage { l3keys2e }
193 \ProcessKeysOptions { zref-check }

```

`\zrchecksetup` Provide `\zrchecksetup`.

```

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

```

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

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

```

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

```

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

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

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

For reference: About what are valid characters for use in labels: <https://tex.stackexchange.com/a/18312>. About some problems with active colons: <https://tex.stackexchange.com/a/89470>. About the difference between L3 strings and token lists, see <https://tex.stackexchange.com/a/446381>, in particular Joseph Wright’s comment: “Strings are for data that will never be typeset, for example file names, identifiers, etc.: if the material may be used in typesetting, it should be a token list.” See also moewe’s (CW) answer in the same lines. Which suggests using L3 strings for the reference labels might be a good catch all approach, and possibly more robust. David Carlisle’s comment about `inputenc` 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`

```

196 \prop_new:N \g__zrefcheck_auxfile_lblseq_prop
(End definition for \g__zrefcheck_auxfile_lblseq_prop.)
197 \tl_set:Nn \g_tmpa_tl { \c_sys_jobname_str .aux }
198 \file_if_exist:nT { \g_tmpa_tl }
199   {

```

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.

```

200 \ior_open:Nn \g_tmpa_ior { \g_tmpa_tl }
201 \group_begin:
202 \int_zero:N \l_tmpa_int
203 \tl_clear:N \l_tmpa_tl
204 \tl_clear:N \l_tmpb_tl
205 \bool_set_false:N \l_tmpa_bool
206 \ior_map_variable:NNn \g_tmpa_ior \l_tmpa_tl
207 {
208   \tl_map_variable:NNn \l_tmpa_tl \l_tmpb_tl
209   {
210     \tl_if_eq:NnTF \l_tmpb_tl { \zref@newlabel }
211     {

```

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

```

212       \bool_set_true:N \l_tmpa_bool
213     }
214   {
215     \bool_if:NNTF \l_tmpa_bool
216     {
217       \bool_set_false:N \l_tmpa_bool
218       \int_incr:N \l_tmpa_int
219       \prop_gput:Nxx \g__zrefcheck_auxfile_lblseq_prop
220       { \l_tmpb_tl } { \int_use:N \l_tmpa_int }
221     }
222   }

```

If there is not a match of the first token with `\zref@newlabel`, break the loop and discard the rest of the line, to ensure no `\catcode` 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.

```

223       \tl_map_break:
224     }
225   }
226 }
227 }
228 \group_end:
229 \ior_close:N \g_tmpa_ior
230 }

```

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.

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.

```

231 \begingroup
232   \let \addtoreset \ltx@gobbletwo
233   \newcounter { zrefcheck }
234 \endgroup
235 \setcounter { zrefcheck } { 0 }

```

4.5 Label formats

```

\__zrefcheck_check_lblfmt:n      \__zrefcheck_check_lblfmt:n {<check id int>}
236 \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>}
237 \cs_new:Npn \__zrefcheck_end_lblfmt:n #1 { #1 @zrefcheck }

(End definition for \__zrefcheck_end_lblfmt:n.)

```

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>}

238 \cs_new:Npn \zrefcheck_get_astl:nnn #1#2#3
239 {
240   \tl_clear:N #3
241   \tl_if_eq:nnTF {#2} { lblseq }
242   {
243     \prop_get:NnNF \g__zrefcheck_auxfile_lblseq_prop {#1} #3
244     {
245       \msg_warning:nnnn { zref-check }
246       { property-not-in-label } {#1} {#2}

```



```

247     }
248   }
249   {

```

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

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

```

250     \zref@ifrefundefined {#1}
251     {}
252     {
253       \zref@ifpropundefined {#2}
254       { \msg_warning:nnnn { zref-check } { property-undefined } {#2} }
255       {
256         \zref@ifrefcontainsprop {#1} {#2}
257         {
258           \tl_set:Nx #3
259           { \zref@extractdefault {#1} {#2} { \c_empty_tl } }
260         }
261         {
262           \msg_warning:nnnn
263           { zref-check } { property-not-in-label } {#1} {#2}
264         }
265       }
266     }
267   }
268 }

```

(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).

```

269 \bool_new:N \l__zrefcheck_integer_bool

```

(End definition for `\l__zrefcheck_integer_bool`.)

`\l__zrefcheck_propval_tl`

270 `\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>}`

271 `\cs_new:Npn \zrefcheck_get_asint:nnn #1#2#3`

272 `{`

273 `\zrefcheck_get_astl:nnn {#1} {#2} { \l__zrefcheck_propval_tl }`

274 `__zrefcheck_is_integer:nTF { \l__zrefcheck_propval_tl }`

275 `{`

Make it an integer data type.

276 `\int_set:Nn #3 { \int_eval:n { \l__zrefcheck_propval_tl } }`

277 `}`

278 `{`

279 `\bool_set_false:N \l__zrefcheck_integer_bool`

280 `\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`.

281 `{ }`

282 `{`

283 `\msg_warning:nnnn { zref-check }`

284 `{ property-not-integer } {#2} {#1}`

285 `}`

286 `}`

287 `}`

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

`__zrefcheck_is_integer:n`

288 `\prg_new_conditional:Npnn __zrefcheck_is_integer:n #1 { p, T, F, TF }`

289 `{`

290 `\tl_if_empty:oTF {#1}`

Empty tl is also not an integer.

291 `{ \prg_return_false: }`

292 `{`

Thanks egreg: <https://tex.stackexchange.com/a/244405>. FIXME This, however,

makes `l3build doc` complain that we're using an internal function of the `int` module,

`__int_to_roman:w`. Which, of course, is true, but I don't know how to replace this.

293 `\tl_if_blank:oTF { __int_to_roman:w -0#1 }`

294 `{ \prg_return_true: }`

295 `{ \prg_return_false: }`

296 `}`

297 `}`

(End definition for `__zrefcheck_is_integer:n`.)

5 \zrcheck

\zrcheck The $\langle text \rangle$ 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{*}[\langle options \rangle]{\langle labels \rangle}[\langle checks \rangle]{\langle text \rangle}
```

```
298 \NewDocumentCommand \zrcheck
299   { s O { } } > { \SplitList { , } } m > { \SplitList { , } } O { } m }
300   { \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 301 \int_new:N \g__zrefcheck_id_int
\l__zrefcheck_checkend_tl 302 \tl_new:N \l__zrefcheck_checkbeg_tl
\l__zrefcheck_link_label_tl 303 \tl_new:N \l__zrefcheck_checkend_tl
\l__zrefcheck_link_anchor_tl 304 \tl_new:N \l__zrefcheck_link_label_tl
\l__zrefcheck_link_star_tl 305 \tl_new:N \l__zrefcheck_link_anchor_tl
306 \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 $\langle labels \rangle$ 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 {\langle labels \rangle} {\langle * \rangle} {\langle options \rangle} {\langle checks \rangle} {\langle text \rangle}
```

```
307 \cs_new:Npn \__zrefcheck_zrcheck:nnnnn #1#2#3#4#5
308   {
309     \group_begin:
```

Process local options.

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

Names of the labels for this `zrefcheck` call.

```
311     \int_gincr:N \g__zrefcheck_id_int
312     \tl_set:Nx \l__zrefcheck_checkbeg_tl
313       { \__zrefcheck_check_lblfmt:n { \g__zrefcheck_id_int } }
314     \tl_set:Nx \l__zrefcheck_checkend_tl
315       { \__zrefcheck_end_lblfmt:n { \l__zrefcheck_checkbeg_tl } }
```

Set `checkbeg` label.

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

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

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

If the reference is undefined, just typeset.

```

320     {#5}
321     {
322         \bool_if:nTF
323         {
324             \l__zrefcheck_use_hyperref_bool &&
325             ! \l__zrefcheck_link_star_tl
326         }
327         {
328             \exp_args:Nx \zrefcheck_get_astl:nnn
329             { \l__zrefcheck_link_label_tl }
330             { anchor } { \l__zrefcheck_link_anchor_tl }
331             \hyperlink { \l__zrefcheck_link_anchor_tl } {#5}
332         }
333     } {#5}
334 }

```

Set checkend label.

```

335     \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.

```

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

```

Run the checks.

```

337     \__zrefcheck_run_checks:nnV {#4} {#1} { \l__zrefcheck_checkbeg_tl }
338     \group_end:
339 }

```

(End definition for __zrefcheck_zrcheck:nnnnn.)

6 Targets

```

\zrctarget      \zrctarget{<label>}{<text>}

340 \NewDocumentCommand \zrctarget { m +m }
341 {
342     \refstepcounter { zrefcheck }
343     \zref@wrapper@babel \zref@labelbylist {#1} { zrefcheck }
344     #2
345     \zref@wrapper@babel
346     \zref@labelbylist { \__zrefcheck_end_lblfmt:n {#1} } { zrefcheck }
347 }

```

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

```

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

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

348 \NewDocumentEnvironment {zrcregion} { m }
349 {
350     \refstepcounter { zrefcheck }
351     \zref@wrapper@babel \zref@labelbylist {#1} { zrefcheck }
352 }
353 {

```

```

354 \zref@wrapper@babel
355 \zref@labelbylist { \__zrefcheck_end_lblfmt:n {#1} } { zrefcheck }
356 }

```

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

7 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.

7.1 Running

```

\__zrefcheck_run_checks:nnn \__zrefcheck_run_checks:nnn {<checks>} {<labels>} {<reference>}
\__zrefcheck_run_checks:nnV
357 \cs_new:Npn \__zrefcheck_run_checks:nnn #1#2#3
358 {
359   \group_begin:
360     \tl_map_inline:nn {#2}
361     {
362       \tl_map_inline:nn {#1}
363       { \__zrefcheck_do_check:nnn {####1} {##1} {#3} }
364     }
365   \group_end:
366 }
367 \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
368 \bool_new:N \l__zrefcheck_passedcheck_bool
369 \bool_new:N \l__zrefcheck_onpage_bool
370 \seq_new:N \c__zrefcheck_onpage_checks_seq
371 \seq_set_from_clist:Nn \c__zrefcheck_onpage_checks_seq
372 { 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.

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

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

```
374 \cs_new:Npn \__zrefcheck_do_check:nnn #1#2#3
```

```
375 {
```

```
376   \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.

```
377   \zref@ifrefundefined {#2}
```

```
378   {}
```

```
379   {
```

```
380     \bool_set_true:N \l__zrefcheck_passedcheck_bool
```

```
381     \bool_set_false:N \l__zrefcheck_onpage_bool
```

```
382     \cs_if_exist:cTF { __zrefcheck_check_ #1 :nnF }
```

```
383     {
```

“label beg” vs “reference beg”.

```
384       \use:c { __zrefcheck_check_ #1 :nnF }
```

```
385       {#2} {#3}
```

```
386       { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
```

“label beg” vs “reference end”.

```
387       \exp_args:Nnno \use:c { __zrefcheck_check_ #1 :nnF }
```

```
388       {#2} { \__zrefcheck_end_lblfmt:n {#3} }
```

```
389       { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
```

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

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

```
391       {}
```

```
392       {
```

“label end” vs “reference beg”.

```
393       \exp_args:Nno \use:c { __zrefcheck_check_ #1 :nnF }
```

```
394       { \__zrefcheck_end_lblfmt:n {#2} } {#3}
```

```
395       { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
```

“label end” vs “reference end”.

```
396       \exp_args:Nnoo \use:c { __zrefcheck_check_ #1 :nnF }
```

```
397       { \__zrefcheck_end_lblfmt:n {#2} }
```

```
398       { \__zrefcheck_end_lblfmt:n {#3} }
```

```
399       { \bool_set_false:N \l__zrefcheck_passedcheck_bool }
```

```
400     }
```

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.

```
401     \seq_if_in:NnT \c__zrefcheck_onpage_checks_seq {#1}
```

```
402     {
```

```
403       \__zrefcheck_check_thispage:nnT
```

```

404         {#2} {#3}
405         { \bool_set_true:N \l__zrefcheck_onpage_bool }
406     \__zrefcheck_check_thispage:nnT
407     {#2} { \__zrefcheck_end_lblfmt:n {#3} }
408     { \bool_set_true:N \l__zrefcheck_onpage_bool }
409     \zref@ifrefundefined { \__zrefcheck_end_lblfmt:n {#2} }
410     {}
411     {
412         \__zrefcheck_check_thispage:nnT
413         { \__zrefcheck_end_lblfmt:n {#2} } {#3}
414         { \bool_set_true:N \l__zrefcheck_onpage_bool }
415         \__zrefcheck_check_thispage:nnT
416         { \__zrefcheck_end_lblfmt:n {#2} }
417         { \__zrefcheck_end_lblfmt:n {#3} }
418         { \bool_set_true:N \l__zrefcheck_onpage_bool }
419     }
420 }
421 \bool_if:NTF \l__zrefcheck_passedcheck_bool
422 {
423     \bool_if:nT
424     {
425         \l__zrefcheck_msgonpage_bool &&
426         \l__zrefcheck_onpage_bool
427     }
428     {
429         \__zrefcheck_message:nnnx { double-check } {#1} {#2}
430         { \zref@extractdefault {#3} {page} {'unknown'} }
431     }
432 }
433 {
434     \__zrefcheck_message:nnnx { check-failed } {#1} {#2}
435     { \zref@extractdefault {#3} {page} {'unknown'} }
436 }
437 }
438 { \msg_warning:nnn { zref-check } { check-missing } {#1} }
439 }
440 \group_end:
441 }

```

(End definition for __zrefcheck_do_check:nnn.)

7.2 Check conditionals

```

\l__zrefcheck_lbl_int More readable scratch variables for the tests.
\l__zrefcheck_ref_int
\l__zrefcheck_lbl_b_int
\l__zrefcheck_ref_b_int

```

(End definition for \l__zrefcheck_lbl_int and others.)

7.2.1 This page

```

\__zrefcheck_check_thispage:nn

```

```

446 \prg_new_conditional:Npnn \__zrefcheck_check_thispage:nn #1#2 { T, F , TF }
447 {
448   \group_begin:
449     \bool_set_true:N \l__zrefcheck_integer_bool
450     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
451     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
452     \bool_lazy_and:nnTF
453       { \l__zrefcheck_integer_bool }
454       {
455         \int_compare_p:nNn
456           { \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.
457           ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
458           ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
459       }
460       { \group_insert_after:N \prg_return_true: }
461       { \group_insert_after:N \prg_return_false: }
462   \group_end:
463 }

```

(End definition for __zrefcheck_check_thispage:nn.)

7.2.2 On page

```

\__zrefcheck_check_above:nn
\__zrefcheck_check_below:nn
464 \prg_new_conditional:Npnn \__zrefcheck_check_above:nn #1#2 { F , TF }
465 {
466   \group_begin:
467     \__zrefcheck_check_thispage:nnTF {#1} {#2}
468     {
469       \bool_set_true:N \l__zrefcheck_integer_bool
470       \zrefcheck_get_asint:nnn {#1} { lblseq } { \l__zrefcheck_lbl_int }
471       \zrefcheck_get_asint:nnn {#2} { lblseq } { \l__zrefcheck_ref_int }
472       \bool_lazy_and:nnTF
473         { \l__zrefcheck_integer_bool }
474         {
475           \int_compare_p:nNn
476             { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
477             ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
478             ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
479         }
480         { \group_insert_after:N \prg_return_true: }
481         { \group_insert_after:N \prg_return_false: }
482     }
483     { \group_insert_after:N \prg_return_false: }
484   \group_end:
485 }
486 \prg_new_conditional:Npnn \__zrefcheck_check_below:nn #1#2 { F , TF }
487 {
488   \__zrefcheck_check_thispage:nnTF {#1} {#2}

```



```

489     {
490         \__zrefcheck_check_above:nnTF {#1} {#2}
491         { \prg_return_false: }
492         { \prg_return_true:  }
493     }
494     { \prg_return_false: }
495 }

```

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

7.2.3 Before / After

```

\__zrefcheck_check_before:nn
\__zrefcheck_check_after:nn
496 \prg_new_conditional:Npnn \__zrefcheck_check_before:nn #1#2 { F }
497 {
498     \__zrefcheck_check_pagesbefore:nnTF {#1} {#2}
499     { \prg_return_true: }
500     {
501         \__zrefcheck_check_above:nnTF {#1} {#2}
502         { \prg_return_true:  }
503         { \prg_return_false: }
504     }
505 }
506 \prg_new_conditional:Npnn \__zrefcheck_check_after:nn #1#2 { F }
507 {
508     \__zrefcheck_check_pagesafter:nnTF {#1} {#2}
509     { \prg_return_true: }
510     {
511         \__zrefcheck_check_below:nnTF {#1} {#2}
512         { \prg_return_true:  }
513         { \prg_return_false: }
514     }
515 }

```

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

7.2.4 Pages

```

\__zrefcheck_check_nextpage:nn
\__zrefcheck_check_prevpage:nn
516 \prg_new_conditional:Npnn \__zrefcheck_check_nextpage:nn #1#2 { F }
\__zrefcheck_check_pagesbefore:nn
517 {
\__zrefcheck_check_ppbefore:nn
518     \group_begin:
\__zrefcheck_check_pagesafter:nn
519     \bool_set_true:N \l__zrefcheck_integer_bool
\__zrefcheck_check_ppafter:nn
520     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
521     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
\__zrefcheck_check_facing:nn
522     \bool_lazy_and:nnTF
523     { \l__zrefcheck_integer_bool }
524     {
525         \int_compare_p:nNn
526         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
527         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
528         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
529     }
530     { \group_insert_after:N \prg_return_true: }

```

```

531         { \group_insert_after:N \prg_return_false: }
532     \group_end:
533 }
534 \prg_new_conditional:Npnn \__zrefcheck_check_prevpage:nn #1#2 { F }
535 {
536     \group_begin:
537     \bool_set_true:N \l__zrefcheck_integer_bool
538     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
539     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
540     \bool_lazy_and:nnTF
541     { \l__zrefcheck_integer_bool }
542     {
543         \int_compare_p:nNn
544         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
545         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
546         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
547     }
548     { \group_insert_after:N \prg_return_true: }
549     { \group_insert_after:N \prg_return_false: }
550     \group_end:
551 }
552 \prg_new_conditional:Npnn \__zrefcheck_check_pagesbefore:nn #1#2 { F , TF }
553 {
554     \group_begin:
555     \bool_set_true:N \l__zrefcheck_integer_bool
556     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
557     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
558     \bool_lazy_and:nnTF
559     { \l__zrefcheck_integer_bool }
560     {
561         \int_compare_p:nNn
562         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
563         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
564         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
565     }
566     { \group_insert_after:N \prg_return_true: }
567     { \group_insert_after:N \prg_return_false: }
568     \group_end:
569 }
570 \cs_new_eq:NN \__zrefcheck_check_ppbefore:nnF \__zrefcheck_check_pagesbefore:nnF
571 \prg_new_conditional:Npnn \__zrefcheck_check_pagesafter:nn #1#2 { F , TF }
572 {
573     \group_begin:
574     \bool_set_true:N \l__zrefcheck_integer_bool
575     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
576     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
577     \bool_lazy_and:nnTF
578     { \l__zrefcheck_integer_bool }
579     {
580         \int_compare_p:nNn
581         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
582         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
583         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
584     }
585 }

```

```

585     { \group_insert_after:N \prg_return_true: }
586     { \group_insert_after:N \prg_return_false: }
587   \group_end:
588 }
589 \cs_new_eq:NN \__zrefcheck_check_ppafter:nnF \__zrefcheck_check_pagesafter:nnF
590 \prg_new_conditional:Npnn \__zrefcheck_check_facing:nn #1#2 { F }
591 {
592   \group_begin:
593     \bool_set_true:N \l__zrefcheck_integer_bool
594     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
595     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
596     \bool_lazy_and:nnTF
597       { \l__zrefcheck_integer_bool }
598       {

```

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

```

599     \legacy_if_p:n { @twoside } &&

```

Now we test “facing”.

```

600     (
601       (
602         \int_if_odd_p:n { \l__zrefcheck_ref_int } &&
603         \int_compare_p:nNn
604           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 }
605       ) ||
606       (
607         \int_if_even_p:n { \l__zrefcheck_ref_int } &&
608         \int_compare_p:nNn
609           { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 }
610       )
611     ) &&
612     ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
613     ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
614   }
615   { \group_insert_after:N \prg_return_true: }
616   { \group_insert_after:N \prg_return_false: }
617 \group_end:
618 }

```

(End definition for __zrefcheck_check_nextpage:nn and others.)

7.2.5 Close / Far

```

\__zrefcheck_check_close:nn
\__zrefcheck_check_far:nn
619 \prg_new_conditional:Npnn \__zrefcheck_check_close:nn #1#2 { F , TF }
620 {
621   \group_begin:
622     \bool_set_true:N \l__zrefcheck_integer_bool
623     \zrefcheck_get_asint:nnn {#1} { abspage } { \l__zrefcheck_lbl_int }
624     \zrefcheck_get_asint:nnn {#2} { abspage } { \l__zrefcheck_ref_int }
625     \bool_lazy_and:nnTF
626       { \l__zrefcheck_integer_bool }
627       {
628         \int_compare_p:nNn
629           { \int_abs:n { \l__zrefcheck_lbl_int - \l__zrefcheck_ref_int } }

```

```

630         <
631         { \l__zrefcheck_close_range_int + 1 } &&
632         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
633         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
634     }
635     { \group_insert_after:N \prg_return_true: }
636     { \group_insert_after:N \prg_return_false: }
637 \group_end:
638 }
639 \prg_new_conditional:Npnn \__zrefcheck_check_far:nn #1#2 { F }
640 {
641     \__zrefcheck_check_close:nnTF {#1} {#2}
642     { \prg_return_false: }
643     { \prg_return_true: }
644 }

```

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

7.2.6 Chapter

```

\__zrefcheck_check_thischap:nn
\__zrefcheck_check_nextchap:nn
\__zrefcheck_check_prevchap:nn
\__zrefcheck_check_chapsafter:nn
\__zrefcheck_check_chapsbefore:nn
645 \prg_new_conditional:Npnn \__zrefcheck_check_thischap:nn #1#2 { F }
646 {
647     \group_begin:
648     \bool_set_true:N \l__zrefcheck_integer_bool
649     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
650     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
651     \bool_lazy_and:nnTF
652     { \l__zrefcheck_integer_bool }
653     {
654         \int_compare_p:nNn
655         { \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.

```

656         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
657         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
658     }
659     { \group_insert_after:N \prg_return_true: }
660     { \group_insert_after:N \prg_return_false: }
661 \group_end:
662 }
663 \prg_new_conditional:Npnn \__zrefcheck_check_nextchap:nn #1#2 { F }
664 {
665     \group_begin:
666     \bool_set_true:N \l__zrefcheck_integer_bool
667     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
668     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
669     \bool_lazy_and:nnTF
670     { \l__zrefcheck_integer_bool }
671     {

```

```

672         \int_compare_p:nNn
673         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
674         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
675     }
676     { \group_insert_after:N \prg_return_true: }
677     { \group_insert_after:N \prg_return_false: }
678 \group_end:
679 }
680 \prg_new_conditional:Npnn \__zrefcheck_check_prevchap:nn #1#2 { F }
681 {
682     \group_begin:
683     \bool_set_true:N \l__zrefcheck_integer_bool
684     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
685     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
686     \bool_lazy_and:nnTF
687     { \l__zrefcheck_integer_bool }
688     {
689         \int_compare_p:nNn
690         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
691         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
692         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
693     }
694     { \group_insert_after:N \prg_return_true: }
695     { \group_insert_after:N \prg_return_false: }
696 \group_end:
697 }
698 \prg_new_conditional:Npnn \__zrefcheck_check_chapsafter:nn #1#2 { F }
699 {
700     \group_begin:
701     \bool_set_true:N \l__zrefcheck_integer_bool
702     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
703     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
704     \bool_lazy_and:nnTF
705     { \l__zrefcheck_integer_bool }
706     {
707         \int_compare_p:nNn
708         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
709         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
710     }
711     { \group_insert_after:N \prg_return_true: }
712     { \group_insert_after:N \prg_return_false: }
713 \group_end:
714 }
715 \prg_new_conditional:Npnn \__zrefcheck_check_chapsbefore:nn #1#2 { F }
716 {
717     \group_begin:
718     \bool_set_true:N \l__zrefcheck_integer_bool
719     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_int }
720     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_int }
721     \bool_lazy_and:nnTF
722     { \l__zrefcheck_integer_bool }
723     {
724         \int_compare_p:nNn
725         { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&

```

```

726         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
727         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
728     }
729     { \group_insert_after:N \prg_return_true: }
730     { \group_insert_after:N \prg_return_false: }
731 \group_end:
732 }

```

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

7.2.7 Section

```

\__zrefcheck_check_thissec:nn
\__zrefcheck_check_nextsec:nn
\__zrefcheck_check_prevsec:nn
\__zrefcheck_check_secsafter:nn
\__zrefcheck_check_secsbefore:nn
733 \prg_new_conditional:Npnn \__zrefcheck_check_thissec:nn #1#2 { F }
734 {
735     \group_begin:
736     \bool_set_true:N \l__zrefcheck_integer_bool
737     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
738     \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
739     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
740     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
741     \bool_lazy_and:nnTF
742     { \l__zrefcheck_integer_bool }
743     {
744         \int_compare_p:nNn
745         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
746         \int_compare_p:nNn
747         { \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.

```

748         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
749         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
750     }
751     { \group_insert_after:N \prg_return_true: }
752     { \group_insert_after:N \prg_return_false: }
753 \group_end:
754 }
755 \prg_new_conditional:Npnn \__zrefcheck_check_nextsec:nn #1#2 { F }
756 {
757     \group_begin:
758     \bool_set_true:N \l__zrefcheck_integer_bool
759     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
760     \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
761     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
762     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
763     \bool_lazy_and:nnTF
764     { \l__zrefcheck_integer_bool }
765     {
766         \int_compare_p:nNn
767         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&

```

```

768         \int_compare_p:nNn
769         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int + 1 } &&
770         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
771     }
772     { \group_insert_after:N \prg_return_true: }
773     { \group_insert_after:N \prg_return_false: }
774 \group_end:
775 }
776 \prg_new_conditional:Npnn \__zrefcheck_check_prevsec:nn #1#2 { F }
777 {
778     \group_begin:
779     \bool_set_true:N \l__zrefcheck_integer_bool
780     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
781     \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
782     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
783     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
784     \bool_lazy_and:nnTF
785     { \l__zrefcheck_integer_bool }
786     {
787         \int_compare_p:nNn
788         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
789         \int_compare_p:nNn
790         { \l__zrefcheck_lbl_int } = { \l__zrefcheck_ref_int - 1 } &&
791         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
792         ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
793     }
794     { \group_insert_after:N \prg_return_true: }
795     { \group_insert_after:N \prg_return_false: }
796 \group_end:
797 }
798 \prg_new_conditional:Npnn \__zrefcheck_check_secsafter:nn #1#2 { F }
799 {
800     \group_begin:
801     \bool_set_true:N \l__zrefcheck_integer_bool
802     \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
803     \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
804     \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
805     \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
806     \bool_lazy_and:nnTF
807     { \l__zrefcheck_integer_bool }
808     {
809         \int_compare_p:nNn
810         { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
811         \int_compare_p:nNn
812         { \l__zrefcheck_lbl_int } > { \l__zrefcheck_ref_int } &&
813         ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 }
814     }
815     { \group_insert_after:N \prg_return_true: }
816     { \group_insert_after:N \prg_return_false: }
817 \group_end:
818 }
819 \prg_new_conditional:Npnn \__zrefcheck_check_secsbefore:nn #1#2 { F }
820 {
821     \group_begin:

```

```

822 \bool_set_true:N \l__zrefcheck_integer_bool
823 \zrefcheck_get_asint:nnn {#1} { abssec } { \l__zrefcheck_lbl_int }
824 \zrefcheck_get_asint:nnn {#2} { abssec } { \l__zrefcheck_ref_int }
825 \zrefcheck_get_asint:nnn {#1} { abschap } { \l__zrefcheck_lbl_b_int }
826 \zrefcheck_get_asint:nnn {#2} { abschap } { \l__zrefcheck_ref_b_int }
827 \bool_lazy_and:nnTF
828 { \l__zrefcheck_integer_bool }
829 {
830   \int_compare_p:nNn
831     { \l__zrefcheck_lbl_b_int } = { \l__zrefcheck_ref_b_int } &&
832   \int_compare_p:nNn
833     { \l__zrefcheck_lbl_int } < { \l__zrefcheck_ref_int } &&
834   ! \int_compare_p:nNn { \l__zrefcheck_lbl_int } = { 0 } &&
835   ! \int_compare_p:nNn { \l__zrefcheck_ref_int } = { 0 }
836 }
837 { \group_insert_after:N \prg_return_true: }
838 { \group_insert_after:N \prg_return_false: }
839 \group_end:
840 }

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

841 </package>

```

Index

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

A		405, 408, 414, 418, 449, 469, 519,
\addchap	8	537, 555, 574, 593, 622, 648, 666,
\addsec	8	683, 701, 718, 736, 758, 779, 801, 822
\AddToHook	21, 28	\l_tmpa_bool ... 15, 205, 212, 215, 217
\AtBeginDocument	101, 147	
B		C
\begingroup	231	\catcode ... 15
bool commands:		\chapter ... 8, 9, 28
\bool_if:NTF	105, 112, 215, 421	closerange (option) ... 6
\bool_if:nTF	322, 423	cs commands:
\bool_lazy_and:nnTF	17, 452, 472, 522,	\cs_generate_variant:Nn . 45, 367, 373
	540, 558, 577, 596, 625, 651, 669,	\cs_if_exist:NTF ... 382
	686, 704, 721, 741, 763, 784, 806, 827	\cs_new:Npn ...
\bool_new:N	79, 80, 155, 269, 306, 368, 369	. 40, 236, 237, 238, 271, 307, 357, 374
\bool_set:Nn	318	\cs_new_eq:NN ... 570, 589
\bool_set_false:N	87, 96, 97, 114, 161, 170, 177,	D
	205, 217, 279, 381, 386, 389, 395, 399	\documentclass ... 6
\bool_set_true:N	86,	E
	91, 92, 165, 171, 176, 212, 380,	\endgroup ... 234
		\endinput ... 12
		exp commands:
		\exp_args:Nnno ... 373, 387

\exp_args:Nno	393	\l_tmpa_int	202, 218, 220
\exp_args:Nnoo	396	int internal commands:	
\exp_args:Nx	328	_int_to_roman:w	18, 293
F			
file commands:		ior commands:	
\file_if_exist:nTF	198	\ior_close:N	229
\fmtversion	3	\ior_map_variable:NNn	206
		\ior_open:Nn	200
		\g_tmpa_ior	200, 206, 229
G			
group commands:		iow commands:	
\group_begin:		\iow_newline:	48, 53, 64, 69, 73, 76
. 201, 309, 359, 376, 448, 466, 518,		K	
536, 554, 573, 592, 621, 647, 665,		keys commands:	
682, 700, 717, 735, 757, 778, 800, 821		\keys_define:nn	
\group_end:		. 81, 116, 123, 149, 156, 181	
. 228, 338, 365, 440, 462, 484, 532,		\keys_set:nn	185, 195, 310
550, 568, 587, 617, 637, 661, 678,		L	
696, 713, 731, 753, 774, 796, 817, 839		\label	3
\group_insert_after:N	460,	legacy commands:	
461, 480, 481, 483, 530, 531, 548,		\legacy_if_p:n	599
549, 566, 567, 585, 586, 615, 616,		\let	232
635, 636, 659, 660, 676, 677, 694,		M	
695, 711, 712, 729, 730, 751, 752,		\MessageBreak	10
772, 773, 794, 795, 815, 816, 837, 838		msg commands:	
H			
\hyperlink	19, 331	\msg_line_number:	49, 54, 57, 59, 61, 65
hyperref (option)	6	\msg_new:nnn	
I			
\ifdraft	134, 169	. 46, 51, 56, 58, 60, 62, 67, 72, 74	
\IfFormatAtLeastTF	3, 4	\msg_warning:nn	113, 119, 145
\ifoptionfinal	140, 175	\msg_warning:nnn	438
int commands:		\msg_warning:nnnn	245, 254, 262, 283
\int_abs:n	629	msglevel (option)	6
\int_compare_p:nNn		N	
. 455, 457, 458, 475, 477, 478,		\newcounter	233
525, 527, 528, 543, 545, 546, 561,		\NewDocumentCommand	194, 298, 340
563, 564, 580, 582, 583, 603, 608,		\NewDocumentEnvironment	348
612, 613, 628, 632, 633, 654, 656,		\noexpand	70
657, 672, 674, 689, 691, 692, 707,		O	
709, 724, 726, 727, 744, 746, 748,		onpage (option)	6
749, 766, 768, 770, 787, 789, 791,		options:	
792, 809, 811, 813, 830, 832, 834, 835		closerange	6
\int_eval:n	276	hyperref	6
\int_gincr:N	23, 29, 311	msglevel	6
\int_if_even_p:n	607	onpage	6
\int_if_odd_p:n	602	P	
\int_incr:N	218	\PackageError	7
\int_new:N		prg commands:	
. 19, 20, 180, 301, 442, 443, 444, 445		\prg_new_conditional:Npnn	21, 288,
\int_set:Nn	276	446, 464, 486, 496, 506, 516, 534,	
\int_use:N	26, 30, 220, 236	552, 571, 590, 619, 639, 645, 663,	
\int_zero:N	24, 202	680, 698, 715, 733, 755, 776, 798, 819	

_zrefcheck_check_above:nnTF . . .	490, 501	541, 555, 559, 574, 578, 593, 597,
_zrefcheck_check_after:nn . . .	496	622, 626, 648, 652, 666, 670, 683,
_zrefcheck_check_before:nn . .	496	687, 701, 705, 718, 722, 736, 742,
_zrefcheck_check_below:nn . . .	464	758, 764, 779, 785, 801, 807, 822, 828
_zrefcheck_check_below:nnTF . .	511	_zrefcheck_is_integer:n
_zrefcheck_check_chapsafter:nn	645	288
_zrefcheck_check_chapsbefore:nn	645	_zrefcheck_is_integer:nnTF . . .
_zrefcheck_check_close:nn . . .	619	274
_zrefcheck_check_close:nnTF . .	641	\l_zrefcheck_lbl_b_int
_zrefcheck_check_facing:nn . .	516	442, 739, 745,
_zrefcheck_check_far:nn	619	761, 767, 782, 788, 804, 810, 825, 831
_zrefcheck_check_lblfmt:n	16, 236, 313	\l_zrefcheck_lbl_int
_zrefcheck_check_nextchap:nn .	645	442, 450, 456, 457, 470, 476,
_zrefcheck_check_nextpage:nn .	516	477, 520, 526, 527, 538, 544, 545,
_zrefcheck_check_nextsec:nn . .	733	556, 562, 563, 575, 581, 582, 594,
_zrefcheck_check_pagesafter:nn	516	604, 609, 612, 623, 629, 632, 649,
_zrefcheck_check_pagesafter:nnTF	508, 589	655, 656, 667, 673, 674, 684, 690,
_zrefcheck_check_pagesbefore:nn	516	691, 702, 708, 709, 719, 725, 726,
_zrefcheck_check_pagesbefore:nnTF	498, 570	737, 747, 748, 759, 769, 770, 780,
_zrefcheck_check_ppafter:nn . .	516	790, 791, 802, 812, 813, 823, 833, 834
_zrefcheck_check_ppafter:nnTF	589	\l_zrefcheck_link_anchor_tl . . .
_zrefcheck_check_ppbefore:nn .	516	301, 330, 331
_zrefcheck_check_ppbefore:nnTF	570	\l_zrefcheck_link_label_tl
_zrefcheck_check_prevchap:nn .	645	301, 317, 319, 329
_zrefcheck_check_prevpage:nn .	516	\l_zrefcheck_link_star_tl
_zrefcheck_check_prevsec:nn . .	733	301, 318, 325
_zrefcheck_check_secsafter:nn	733	_zrefcheck_message:nnnn 40, 429, 434
_zrefcheck_check_secsbefore:nn	733	\l_zrefcheck_msglevel_tl . . . 42, 122
_zrefcheck_check_thischap:nn .	645	\l_zrefcheck_msgonpage_bool 155, 425
_zrefcheck_check_thispage:nn .	446	\l_zrefcheck_onpage_bool
_zrefcheck_check_thispage:nnTF	403, 406, 412, 415, 467, 488	368, 381, 405, 408, 414, 418, 426
_zrefcheck_check_thissec:nn . .	733	\c_zrefcheck_onpage_checks_seq .
\l_zrefcheck_checkbeg_tl	301, 312, 315, 316, 337	368, 401
\l_zrefcheck_checkend_tl	301, 314, 335	\l_zrefcheck_passedcheck_bool . .
\l_zrefcheck_close_range_int . . .	180, 631	368, 380, 386, 389, 395, 399, 421
_zrefcheck_do_check:nnn 22, 363, 374		\l_zrefcheck_propval_tl
_zrefcheck_end_lblfmt:n	16, 237, 315, 346, 355, 388, 390,	270, 273, 274, 276
394, 397, 398, 407, 409, 413, 416, 417		\l_zrefcheck_ref_b_int
_zrefcheck_get_asint:nnn	16	442, 740, 745,
_zrefcheck_get_astl:nnn	16	762, 767, 783, 788, 805, 810, 826, 831
\g_zrefcheck_id_int . . 301, 311, 313		\l_zrefcheck_ref_int
\l_zrefcheck_integer_bool	17, 269, 279,	442, 451, 456, 458,
449, 453, 469, 473, 519, 523, 537,		471, 476, 478, 521, 526, 528, 539,
		544, 546, 557, 562, 564, 576, 581,
		583, 595, 602, 604, 607, 609, 613,
		624, 629, 633, 650, 655, 657, 668,
		673, 685, 690, 692, 703, 708, 720,
		725, 727, 738, 747, 749, 760, 769,
		781, 790, 792, 803, 812, 824, 833, 835
		_zrefcheck_run_checks:nnn
		21, 337, 357
		\l_zrefcheck_use_hyperref_bool .
		79, 105, 114, 324
		\l_zrefcheck_warn_hyperref_bool
		79, 112
		_zrefcheck_zrcheck:nnnnn
		17–19, 22, 300, 307