The zref-check package*

Gustavo Barros†

2021-07-27

Contents

1 Introduction

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

The basic idea is that, instead of trying to provide the text to be typeset based on the contextual information (as varioref does), zref-check lets the user supply an arbitrary text and specify a number of checks to be done on the label(s) being referred to. If the checks fail, a warning is issued upon compilation, so that the user can go back to that cross-reference and correct it as needed. In a way, this shares the spirit of widows-and-orphans: instead of trying to fix it for you automatically (as nowidow does), it just

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

provides a warning so that the problem can be identified (and fixed) without having to rely on burdensome and error prone manual proof-reading.

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

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

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

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

1.1 Hard vs. soft cross-references

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

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

Granted, the need and usage of this type of document self reference certainly depend on the kind of document, on the area of knowledge, etc. However, they do tend to be employed in a number of places, particularly in longer documents. And that's precisely the scenario in which they may become problematic. If your document is short (say, a paper/article) and it was made in a reasonably short spurt, you'll probably won't bother with this kind of references. In this case, and to use varioref's expression, you "usually know (!)" them to be correct. However, if you are preparing one of those long, complex,

and "long ground" documents, with several rounds of editing and content rearranging, this kind of references will likely bring you trouble. They are not only hard to check and find, but they are also hard to fix. After all, if you are making one such reference, you are taking that statement as a premisse at the current point in the text. So, if that reference is missing, or relocated, you may need to bring in the support to the premisse for your argument to close, rather than just "adjust the reference text".

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

2 Loading the package

As usual:

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

3 Dependencies

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

4 User interface

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

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

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

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

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

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

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

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

5 Checks

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

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

The available checks are the following:

`thispage` `{`⟨*label*⟩`}` occurs on the same page as `\zrcheck`.

`prevpage` `{`⟨*label*⟩`}` occurs on the previous page relative to `\zrcheck`.

`nextpage` `{`⟨*label*⟩`}` occurs on the next page relative to `\zrcheck`.

`facing` On a `twoside` document, both `{`⟨*label*⟩`}` and `\zrcheck` fall onto a double spread, each on one of the two facing pages.

`above` `{`⟨*label*⟩`}` and `\zrcheck` are both on the same page, and `{`⟨*label*⟩`}` occurs "above" `\zrcheck` (for how this is inferred, see Section 8.2).

`below` `{`⟨*label*⟩`}` and `\zrcheck` are both on the same page, and `{`⟨*label*⟩`}` occurs "below" `\zrcheck`.

`pagesbefore` `{`⟨*label*⟩`}` occurs on any page before the one of `\zrcheck`.

`ppbefore` Convenience alias for `pagesbefore`.

`pagesafter` `{`⟨*label*⟩`}` occurs on any page after the one of `\zrcheck`.

`ppafter` Convenience alias for `pagesafter`.

`before` Either `above` or `pagesbefore`.

`after` Either `below` or `pagesafter`.

`thischap` `{`⟨*label*⟩`}` occurs on the same chapter as `\zrcheck`.

`prevchap` `{`⟨*label*⟩`}` occurs on the previous chapter relative to the one of `\zrcheck`.

`nextchap` `{`⟨*label*⟩`}` occurs on the next chapter relative to the one of `\zrcheck`.

`chapsbefore` `{`⟨*label*⟩`}` occurs on any chapter before the one of `\zrcheck`.

`chapsafter` `{`⟨*label*⟩`}` occurs on any chapter after the one of `\zrcheck`.

`thissec` `{`⟨*label*⟩`}` occurs on the same section as `\zrcheck`.

`prevsec` `{`⟨*label*⟩`}` occurs on the previous section (of the same chapter) relative to the one of `\zrcheck`.

`nextsec` `{`⟨*label*⟩`}` occurs on the next section (of the same chapter) relative to the one of `\zrcheck`.

`secsbefore` `{`⟨*label*⟩`}` occurs on any section (of the same chapter) before the one of `\zrcheck`.

`secsafter` `{`⟨*label*⟩`}` occurs on any section (of the same chapter) after the one of `\zrcheck`.

`close` `{`⟨*label*⟩`}` occurs within a page range from `closerange` pages before the one of `\zrcheck` to `closerange` pages after it (about `closerange`, see Section 6).

`far` Not `close`.

6 Options

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

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

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

`onpage` Allows to control the messaging style for "within page checks", and takes values `labelseq`, `msg`, `obeydraft`, `obeyfinal`. The default, `labelseq` uses the labels' shipout sequence, as retrieved from the `.aux` file, to infer relative position within the page. `msg` also uses the same method for checking relative position, but issues a (different) message **even if the check passes**, to provide a simple workflow for robust checking of "false negatives" at a final typesetting stage of the document, considering the label sequence is

not fool proof (for details, see Section 8.2). `msg` also issues its messages at the same level defined in `msglevel`. `obeydraft` corresponds to `labelseq` if option `draft` is passed to `\documentclass` and to `msg` otherwise. `obeyfinal` corresponds to `msg` if option `final` is (explicitly) passed to `\documentclass`, and to `labelseq` otherwise.

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

`labelcmd` Defines the command used to set the user labels in `\zrctarget` and `zrcregion`. Takes a control sequence *name* as value, and the default sets labels with the minimal required properties, those of the `zrefcheck` property list. This is a preamble only option. The specified control sequence must receive one mandatory argument (the `{`⟨*label*⟩`}`) and must generate a `zref label` with at least the properties in the `zrefcheck` property list. The intended use case is that of the user creating a convenience macro which calls both `\label` and `\zlabel`, as suggested in Section 1, so that the same labels are accessible either from the standard reference system or from zref. For example:

```
\NewDocumentCommand\mybothlabels{m}{\label{#1}\zlabel{#1}}
\zrchecksetup{labelcmd=mybothlabels}
```

Note that the value of the underlying counter used for labels in `\zrctarget` and `zrcregion` – what you'd get with a plain `\ref` here – is not really meaningful. But you get to use `\pageref{`⟨*label*⟩`}`, or hyperref's `\hyperref[`⟨*label*⟩`]{`⟨*text*⟩`}` on the labels used in `\zrctarget` and `zrcregion` with this procedure.

7 Label names

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

8 Technique and limitations

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

8.1 Page number checks

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

This is why the user commands of this package set always a pair or labels around `{`⟨*text*⟩`}`. So, when checking `\zrcheck` against a regular `zlabel` both the start and the end of the `{`⟨*text*⟩`}` are checked against the label, and the check fails if either of them fails. When checking `\zrcheck` against a `\zrctarget` or a `zrcregion`, both beginnings and ends are checked against each other two by two, and if any of them fails, the check

fails. In other words, if a page number checks passes, we know that the entire `{`⟨*text*⟩`}` arguments pass it.

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

8.2 Within page checks

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

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

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

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

8.3 Sectioning checks

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

The caveat is that the package does this by hooking into `\chapter` and `\section`, which poses two restrictions for the proper working of these checks. First, we are using the new hook system for this, as provided by ltcmdhooks, which means a recent LaTeX kernel is required. Second, since we are hooking into `\chapter` and `\section`, these checks presume these commands are being used by the document class for this purpose (either directly, or internally as, for example, KOMA-Script's `\addchap` and `\addsec`

do). If that's not the case, additional setup may be required for these checks to work as expected.