The zref-check package*

Gustavo Barros†

2021-08-17

Abstract

zref-check provides an user interface for making LaTeX cross-references flexibly, while allowing to have them checked for consistency with the document structure as typeset. Statements such as "above", "on the next page", "previously", "as will be discussed", "on the previous chapter" and so on can be given to `\zcheck` in free-form, and a set of "checks" can be specified to be run against a given `label`, which will result in a warning at compilation time if any of these checks fail. `\zctarget` and the `zcregion` environment are also defined as a means to easily set label targets to arbitrary places in the text which can be referred to by `\zcheck`.

Contents

*This file describes v0.2.0, last revised 2021-08-17.

†https://github.com/gusbrs/zref-check

1 Introduction

The zref-check package provides an user interface for making LaTeX cross-references exploiting document contextual information to enrich the way the reference can be rendered, but at the same time ensuring the means that these cross-references can be done consistently with the document structure.

The usual LaTeX cross-reference is done by referring to a `label`, associated with one or another document structural element, and this reference will typeset for you some content based on the information which is stored in that label. `\zcheck`, the main user command of zref-check, has a somewhat different concept. Instead of trying to provide the text to be typeset based on the contextual information, `\zcheck` lets the user supply an arbitrary text and specify one or more checks to be done on the label(s) being referred to. If any of the checks fails, a warning is issued upon compilation, so that the user can go back to that cross-reference and correct it as needed, without having to rely on burdensome and error prone manual proof-reading.

This grants a much increased flexibility for the cross-reference text, which means in practice that the writing style, the variety of expressions you may use for similar situations, does not need to be sacrificed for the convenience. `\zcheck`'s cross-references do not need to "feel" automated to be consistently checked. Localization is also not an issue, since the cross-reference text is provided directly by the user. Separating "typesetting" from "checking" also means there is a lot of document context we can leverage for this purpose (see Section 5).

A standard LaTeX cross-reference is made to 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").

For references to these usual specific document elements, zref-check caters for a particular kind of cross-reference which is common: *relational* statements based on them. `\zcheck` can typeset and meaningfully check cross-references such as "above", "on the next page", "on the facing page", "on the previous section", "later on this chapter" and so on. After all, if your reference is being made on page 2 and refers to something on the same page, "on this page" reads much better than "on page 2". If you are writing chapter 4, "on the previous chapter" sounds nicer than "on chapter 3".

However, there is yet another kind of "looser" cross-reference we routinely do in our documents. Expressions such as "previously", "as mentioned before", "as will be discussed", and so on, are a powerful discursive instrument, which enriches the text, by offering hints to the arguments' threads, without necessarily pressing them too hard onto the reader. So, we might not want to say "on footnote 57, pag. 34", but prefer "previously", not "on Section 3.4", but rather "below", or "later on". Besides, we also may refer to certain passages in the text in this way, rather than to numbered document elements. And this kind of reference is not only hard to check and find, but also 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". zref-check also provides support for this kind of cross-reference, allowing for them to be consistently verified.

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 (from the oberdiek bundle) is also loaded by default. A LaTeX kernel later than 2021-06-01 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

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

Typesets ⟨*text*⟩, as given, while performing a list of ⟨*checks*⟩ on 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. ⟨*checks*⟩ and ⟨*options*⟩ can be given side by side as a comma separated list in the optional argument. ⟨*labels*⟩ is also a comma separated list.

`\zctarget` `\zctarget{⟨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*⟩.

`zcregion`
```
\begin{zcregion}{⟨label⟩}
  ...
\end{zcregion}
```
An environment that does the same as `\zctarget`, for cases of longer stretches of text.

`\zrefchecksetup` `\zrefchecksetup{⟨options⟩}`

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

5 Checks

zref-check provides several "checks" to be used with `\zcheck`. The checks may be combined in a `\zcheck` 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 `\zcheck`.

The use of checks which perform "within the page" comparisons – namely `above` and `below` and, through them, `before` and `after` – comes with some caveats you should

be acquainted with. Section 8.2 discusses their limitations and expands on the expected workflow for their use to ensure reliable results.

Note that the naming convention of the checks adopts the perspective of `\zcheck`. That is, the name of the check describes the position of the label being referred to, relative to the `\zcheck` call being made. For example, the `before` check should issue no message if `\ztarget{mylabel}{...}` occurs before `\zcheck[before]{mylabel}{...}`.

The available checks are the following:

`thispage` ⟨*label*⟩ occurs on the same page as `\zcheck`.

`prevpage` ⟨*label*⟩ occurs on the previous page relative to `\zcheck`.

`nextpage` ⟨*label*⟩ occurs on the next page relative to `\zcheck`.

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

`above` ⟨*label*⟩ and `\zcheck` are both on the same page, and ⟨*label*⟩ occurs "above" `\zcheck`.

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

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

`ppbefore` Convenience alias for `pagesbefore`.

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

`ppafter` Convenience alias for `pagesafter`.

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

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

`thischap` ⟨*label*⟩ occurs on the same chapter as `\zcheck`.

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

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

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

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

`thissec` ⟨*label*⟩ occurs on the same section as `\zcheck`.

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

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

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

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

`close` ⟨*label*⟩ occurs within a page range from `closerange` pages before the one of `\zcheck` to `closerange` pages after it (about the `closerange` option, 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 `\zrefchecksetup` (see Section 4) in the preamble. Most options can also be used with local effects, through the optional argument of `\zcheck`.

`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 `\zcheck` 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 of these cases, 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 `\zcheck` provides granular control of hyperlinking by means of its starred version.

`msglevel` Sets the level of messages issued by `\zcheck` 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 `\zcheck`, 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", considering the label sequence is not fool proof (for details and workflow recommendations, 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 the range of pages, relative to the reference, that are considered "close" by the `close` check. Takes a positive integer as value, with default 5.

`labelcmd` Defines the command used to set the user labels in `\zctarget` and `zcregion`. 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 7, 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}}
\zrefchecksetup{labelcmd=mybothlabels}
```

Note that the value of the underlying counter used for labels in `\zctarget` and `zcregion` – 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 `\zctarget` and `zcregion` with this procedure.

7 Labels

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 `\zctarget` 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 may need to add extra labels for this. That's true. But zref-check is not meant to replace the existing functionality of the kernel or of the traditional packages in this area (to my knowledge, it only intersects directly with varioref and, even so, it is quite different in scope). Indeed, it is easy to see the use in tandem with standard references, for example:

```
... Figure~\ref{fig:figure-1}, \zcheck*[nextpage]{fig:figure-1}{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.

All user commands of zref-check have their `{`⟨*label*⟩`}` arguments protected for babel active characters using zref's `\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 Limitations

There are three qualitatively different kinds of checks being used by `\zcheck`, 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`, `pagesbefore`, `pagesafter`, `facing` – 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 `\zcheck`, when typeset, crosses page boundaries, starting in one page, and finishing in another. The same can happen with the text in `\zctarget` and the `zcregion` environment.

This is why the user commands of this package set a pair or labels around ⟨*text*⟩. So, when checking `\zcheck` 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 `\zcheck` against a `\zctarget` or a `zcregion`, 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 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` at a final typesetting stage (or just set it to `obeydraft` or `obeyfinal` if that's part of your workflow) 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 LaTeX kernel later than 2021-06-01 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 needed for these checks to work as expected.