The zref-clever package[*]

Gustavo Barros[†]

2021-09-29

Abstract

zref-clever provides an user interface for making LaTeX cross-references which automates some of their typical requirements, thus easing their input in the document and improving the consistency of typeset results. A reference made with `\zcref` includes a "name" according to its "type" and lists of multiple labels can be automatically sorted and compressed into ranges when due. The reference format is highly and easily customizable, both globally and locally. zref-clever is based on zref's extensible referencing system.

Contents

[*]This file describes v0.1.0-alpha, released 2021-09-29.

[†]`https://github.com/gusbrs/zref-clever`

1 Introduction

2 Loading the package

As usual:

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

3 Dependencies

zref-clever requires zref, and LaTeX kernel 2021-06-01, or newer. It also needs l3keys2e. Some packages are leveraged by zref-clever if they are present, but are not loaded or required by it, namely: hyperref, zref-titleref (zref's module), and zref-check.

4 User interface

`\zcref`

`\zcref⟨*⟩[⟨options⟩]{⟨labels⟩}`

Typesets references to ⟨*labels*⟩, given as a comma separated list. When hyperref support is enabled, references will be hyperlinked to their respective anchors, according to options. The starred version of the command does the same as the plain one, just does not form links. The ⟨*options*⟩ are (mostly) the same as those of the package, and can be given to local effect.

`\zcpageref`

`\zcpageref⟨*⟩[⟨options⟩]{⟨labels⟩}`

Typesets page references to ⟨*labels*⟩, given as a comma separated list. It is equivalent to calling `\zcref` with the `ref=page` option: `\zcref⟨*⟩[⟨options⟩, ref=page]{⟨labels⟩}`.

`\zcsetup`

`\zcsetup{⟨options⟩}`

Sets zref-clever's general options (see Section 5).

`\zcRefTypeSetup`

`\zcRefTypeSetup {⟨type⟩} {⟨options⟩}`

Sets type-specific reference format options (see Section 7).

Besides these, user facing commands related to internationalization are presented in Section 8.

5 Options

zref-clever is highly configurable, offering a lot of flexibility in typeset results of the references, but it also tries to keep these "handles" as convenient and user friendly as possible. To this end, most of what one can do with zref-clever (pretty much all of it), can be achieved directly through the standard and familiar "comma separated list of `key=value` options".

There are two main groups of options in zref-clever: "general options", which affect the overall behavior of the package, or the reference as a whole; and "reference format options", which control the detail of reference formatting, including type-specific and language-specific settings.

This section covers the first group (for the second one, see Section 7). General options can be set globally either as package options at load-time (see Section 2) or by means of `\zcsetup` in the preamble (see Section 4). They can also be set locally with `\zcsetup` along the document or through the optional argument of `\zcref` (see Section 4). Most general options can be used in any of these contexts, but that is not necessarily true for all cases, some restrictions may apply, as described in each option's documentation.

`ref` `page` The `ref` option controls the label property to which `\zcref` refers to. It can receive values `zc@thecnt` and `page`. If zref-titleref is loaded, `ref` also accepts the value `title`. The package's default is `zc@thecnt`, which is an internal property equivalent to zref's `default` property, except that it is not affected by the kernel's `\labelformat`. In sum, just what you'd expect from a regular reference. By default, sorting and compression is done according to the information of the counter underlying `zc@thecnt`. Special treatment in these areas is provided for `page`, but not for `title`. The `page` option is a convenience alias for `ref=page`.

`typeset` `noname` When `\zcref` typesets a set of references, each group of references of the same type can be, and by default are, preceded by the type's "name", and this is indeed an important feature of zref-clever. This is optional however, and the `typeset` option controls this behavior. It can receive values `ref`, in which case it typesets only the reference(s), `name`, in which case it typesets only the name(s), or `both`, in which case it typesets, well, both of them. Note that, when value `name` is used, the name is still typeset according to the set of references given to `\zcref`. For example, for multiple references, the plural form is used, capitalization options are honored, etc. Also hyperlinking behaves just *as if* the references were present and, depending on the corresponding options, the name may be linked to the first reference of the type group. The `noname` option is a convenience alias for `typeset=ref`.

`sort` `nosort` The `sort` option controls whether the list of ⟨*labels*⟩ received as argument by `\zcref` should be sorted or not. It is a boolean option, and defaults to `true`. The `nosort` option is a convenience alias for `sort=false`.

`typesort` `notypesort` Sorting references of the same type can be done with well defined logical criteria. They either have the same counter or their counters share a clear hierarchical relation (in the resetting behavior), such that a definite sorting rule can be inferred from the label's data. The same is not true for sorting of references of different types. Should "tables" come before or after "figures"? The typesort option allows to specify the sorting priority of different reference types. It receives as value a comma separated list of reference types, specifying that their sorting is to be done in the order of that list. But `typesort` does not need to receive *all* possible reference types. The special value `{othertypes}` (yes, braced) can be placed anywhere along the list, to specify the sort priority of any type not included explicitly in the list. If `{othertypes}` is not present in the list, it is presumed to be at the end of it. Any unspecified types (that is, those falling implicitly or explicitly into the `{othertypes}` category) get sorted between themselves in the order of their first appearance in the label list given as argument to `\zcref`. I presume the common use cases will not need to specify `{othertypes}` at all but, for the sake of example, if you just really dislike equations, you could use `typesort={{othertypes}, equation}`. `typesort`'s default value is `{part, chapter, section, paragraph}`, which places the sectioning reference types first in the list, in their hierarchical order, and leaves everything else to the order of appearance of the labels. The `notypesort` option behaves like `typesort={{othertypes}}` would do, that is, it sorts all types in the order of the first appearance in the labels' list.

`comp` `nocomp` `\zcref` can automatically compress a set of references of the same type into a range, when they occur in immediate sequence. The `comp` controls whether this compression should take place or not. It is a boolean option, and defaults to `true`. The `nocomp` option is a convenience alias for `comp=false`. Of course, for better compression results the `sort` is recommended, but the two options are technically independent.

`range` By default (that is, when the `range` option is not given), `\zcref` typesets a complete list of references according to the ⟨*labels*⟩ it received as argument, and only compresses some of them into ranges if the `comp` option is enabled and if references of the same type occur in immediate sequence. The `range` option makes `\zcref` behave differently. Sorting is implied by this option (the `sort` option is disregarded) and, for each reference type group in ⟨*labels*⟩, `\zcref` builds a range from the first to the last reference in it, even if references in between do not occur in immediate sequence. `\zcref` is smart enough, though, to recognize when the first and last references of a type do happen to be contiguous, in which case it typesets a "pair", instead of a "range". It is a boolean option, and the package's default is `range=false`. The option given without a value is equivalent to `range=true` (in the l3keys' jargon, the *option*'s default is `true`).

`nameinlink` The `nameinlink` option controls whether the type name should be included in the reference hyperlink or not (provided there is a link, of course). Naturally, the name can only be included in the link of the *first* reference of each type block. `nameinlink` can receive values `true`, `false`, `single`, and `tsingle`. When the value is `true` the type name is always included in the hyperlink. When it is `false` the type name is never included in the link. When the value is `single`, the type name is included in the link only if `\zcref` is typesetting a single reference (not necessarily having received a single label as argument, as they may have been compressed), otherwise, the name is left out of the link. When the value is `tsingle`, the type name is included in the link for each type block with a single reference, otherwise, it isn't. An example: suppose you make a couple of references to something like `\zcref{chap:chapter1}` and `\zcref{chap:chapter1, sec:section1, fig:figure1, fig:figure2}`. The "figure" type name will only be included in the hyperlink if `nameinlink` option is set to `true`. If it is set to `tsingle`, the first reference will include the name in the link for "chapter", as expected, but also in the second reference the "chapter" and "section" names will be included in their respective links, while that of "figure" will not. If the option is set to `single`, only the name for "chapter" in the first reference will be included in the link, while in the second reference none of them will. The package's default is `nameinlink=tsingle`, and the option given without a value is equivalent to `nameinlink=true`.

`cap`
`nocap`
`capfirst`
`abbrev`
`noabbrevfirst`
`C`
`hyperref`
`lang`
`font`
`note`
`check`
`countertype`
`counterresetters`
`counterresetby`

6 Reference types

A “reference type” is the basic zref-clever setup unit for specifying how a cross-reference group of a certain kind is to be typeset. Though, usually, it will have the same name as the underlying LaTeX *counter*, they are conceptually different. zref-clever sets up *reference types* and an association between each *counter* and its *type*, it does not define the counters themselves, which are defined by your document. One *reference type* can be associated with one or more *counters*, and a *counter* can be associated with different *types* at different points in your document. But each label is stored with only one *type*, as specified by the counter-type association at the moment it is set, and that determines how the reference to that label is typeset. References to different *counters* of the same *type* are grouped together, and treated alike by `\zcref`. A *reference type* may be known to zref-clever when the *counter* it is associated with is not actually defined, and this inconsequential. In practice, the contrary may also happen, a *counter* may be defined but we have no *type* for it, but this must be handled by zref-clever as an error (at least, if we try to refer to it), usually a “missing name” error.

The association of a *counter* to its *type* is controlled by the `countertype` option. As seen in its documentation, zref-clever presumes the *type* to be the same as the *counter* unless instructed otherwise by that option. This association, as determined by the local value of the option, affects how the *label* is set, which stores the type among its properties. However, when it comes to typesetting, that is from the perspective of `\zcref`, only the *type* matters. In other words, how the reference is supposed to be typeset is determined at the point the *label* gets set. In sum, they may be namesakes (or not), but “type” is *type* and “counter” is *counter*.

A *reference type* can be associated with multiple counters because we may want to refer to different document elements, with different *counters*, as a single *type*, with a single name. One prominent case of this are sectioning commands. `\section`, `\subsection`, and `\subsubsection` have each their counter, but we’d like to refer to all of them by “sections” and group them together. The same for `\paragraph` and `\subparagraph`.

There are also cases in which we may want to use different *reference types* to refer to document objects sharing the same *counter*. Notably, the environments created with LaTeX’s `\newtheorem` command and the `\appendix`.

One more observation about “reference types” is due here. A *type* is not really “defined” in the sense a variable or a function is. It is more of a “string” which zref-clever uses to look for a whole set of type-specific reference format options (see Section 7). Each of these options individually may be “set” or not, “defined” or not. And, depending on the setup and the relevant precedence rules for this, some of them may be required and some not. In practice, zref-clever uses the *type* to look for these options when it needs one, and issues a compilation warning when it cannot find a suitable value.

7 Reference format

8 Internationalization

`\zcDeclareLanguage`

`\zcDeclareLanguage {⟨language⟩}`

Declare a new language for use with zref-clever. If ⟨*language*⟩ is already known, just warn. `\zcDeclareLanguage` is preamble only.

`\zcDeclareLanguageAlias` `\zcDeclareLanguageAlias {⟨language alias⟩} {⟨aliased language⟩}`

Declare ⟨*language alias*⟩ to be an alias of ⟨*aliased language*⟩. ⟨*aliased language*⟩ must be already known to zref-clever. `\zcDeclareLanguageAlias` is preamble only.

`\zcDeclareTranslations` `\zcDeclareTranslations {⟨language⟩} {⟨options⟩}`

Sets language-specific reference format options for ⟨*language*⟩ (see Section 7). ⟨*language*⟩ must be already known to zref-clever. `\zcDeclareTranslations` is preamble only.

9 Usage examples

10 Limitations

11 Acknowledgments

12 Change history

A change log with relevant changes for each version, eventual upgrade instructions, and upcoming changes, is maintained in the package's repository, at `https://github.com/gusbrs/zref-clever/blob/main/CHANGELOG.md`.