The zref-clever package[*]

Gustavo Barros[†]

2021-09-13

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 2021-09-13.

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

1 Introduction

2 Loading the package

3 Dependencies

4 User interface

5 Options

6 Reference Types

A bit of terminology, to avoid confusion. 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 defines *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 zref-clever. A *reference type* may exist even 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 a "missing type" error (at least, if we try to refer to it).

A *reference type* can be associated with multiple counters because we may want to refer to different document elements, with different *counters*, with a single name, as a single *type*. 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 "section". The same for `\paragraph` and `\subparagraph`. There is one relevant subtlety to grouping multiple counters under the same type: in order for us to be able to meaningfully sort and compress this group, the set of counters contained therein cannot be arbitrary. Indeed, all of the *counters* grouped in the same *type* must belong to the same counter reset chain, and must be nested within each other (they cannot even just share the same parent).

There are also cases in which we may want to use different *reference types* to refer to document objects sharing the same *counter*. Prominently, the environments created with the kernel's `\newtheorem` command and the `\appendix`, but we'll try to consider, and handle, the case generally.

Another relevant use case of the same general problem of different types for the same counter is the `\appendix` which in some document classes, including the standard ones, change the sectioning commands looks but, of course, keep using the same counter (`book.cls` and `report.cls` reset counters `chapter` and `section` to 0, change `\@chapapp` to use `\appendixname` and use `\@Alph` for `\thechapter`; `article.cls` resets counters `section` and `subsection` to 0, and uses `\@Alph` for `\thesection`; `memoir.cls`, `scrbook.cls` and `scrarticle.cls` do the same as their corresponding standard classes, and sometimes a little more, but what interests us here is pretty much the same; see also the appendix package).

All in all, and since zref spares us of the need to redefine such core commands, I think a more general approach, even if a little less automatic, is the best for us here. zref's data flexibility also helps us in this. As it turns out, we can also use `\l_@@_counter_type_prop` for this purpose (hence it was made locally scoped). And we do so by storing, with the label, the "type" value of the "counter" key in `\l_@@_counter_type_prop` when the label is set. If it was not for the need to distinguish different *types* of the same *counter* this information could be kept in the variable alone, but since we need to leverage other document information in the process, storing it with the label is not a bad idea. And it makes some things simpler even for the general case, since we don't have to control whether there is a type property in the label or not. (The property would have to be included anyway, since the `\appendix` case offers little in terms of hooks or grouping, the only choice is whether to populate this property for every label or just for the ones we'd like to "override"). With that in hand, `\l_@@_counter_type_prop` can be set at appropriate times, and the information gets stored in the label. For environments, it is trivial with a hook to `env/⟨env⟩/begin`. This can be used for `\newtheorem` environments to start with. In principle, with a recent kernel, a hook to `\appendix` could also be used, otherwise some (simple) user intervention may be required.

7 Limitations

8 Acknowledgments

9 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.