The zref-clever package∗
Code documentation

Gustavo Barros†
2022-01-10

EXPERIMENTAL

Contents
1 Initial setup 2
2 Dependencies 3
3 zref setup 3
4 Plumbing 7
  4.1 Auxiliary 7
  4.2 Messages 8
  4.3 Data extraction 11
  4.4 Reference format 11
  4.5 Languages 13
  4.6 Language files 17
  4.7 Options 26
5 Configuration 42
  5.1 \zcsetup 42
  5.2 \zcRefTypeSetup 42
  5.3 \zcLanguageSetup 45
6 User interface 50
  6.1 \zcref 50
  6.2 \zcpageref 52
7 Sorting 52
8 Typesetting 59

∗This file describes v0.1.2-alpha, released 2022-01-10.
†https://github.com/gusbrs/zref-clever

1
1 Initial setup

Start the \texttt{DocStrip} guards.

\begin{verbatim}
1 \langle∗package\rangle
2 \langle@@=zrefclever\rangle
3 Taking a stance on backward compatibility of the package. During initial development, we have used freely recent features of the kernel (albeit refraining from \texttt{l3candidates}, even though I’d have loved to have used \texttt{\textbackslash boole_case_true}:...). We presume \texttt{xparse} (which made to the kernel in the 2020-10-01 release), and \texttt{expl3} as well (which made to the kernel in the 2020-02-02 release). We also just use UTF-8 for the language files (which became the default input encoding in the 2018-04-01 release). Finally, a couple of changes came with the 2021-11-15 kernel release, which are important here. First, a fix was made to the new hook management system (\texttt{ltcmdhooks}), with implications to the hook we add to \texttt{\textbackslash appendix} (by Phelype Oleinik at \url{https://tex.stackexchange.com/q/617905} and \url{https://github.com/latex3/latex2e/pull/699}). Second, the support for \texttt{\textbackslash@currentcounter} has been improved, including \texttt{\textbackslash footnote} and \texttt{amsmath} (by Frank Mittelbach and Ulrike Fischer at \url{https://github.com/latex3/latex2e/issues/687}). Hence, since we would not be able to go much backwards without special handling anyway, we make the cut at the 2021-11-15 kernel release.
4 \providecommand\IfFormatAtLeastTF\{\0\1\0\1\0\0\textbackslash fmtversion\}
5 \IfFormatAtLeastTF\{2021-11-15\}
6 \{%
7 \PackageError{zref-clever}{LaTeX kernel too old}{%
8 \%
\end{verbatim}
'zref-clever' requires a LaTeX kernel 2021-11-15 or newer.%
\MessageBreak Loading will abort!%
}\%
\endinput

Identify the package.
\ProvidesExplPackage {zref-clever} {2022-01-10} {0.1.2-alpha}
\{Clever LaTeX cross-references based on zref\}

2 Dependencies

Required packages. Besides these, zref-hyperref, zref-titleref, and zref-check may also be
loaded depending on user options.
\RequirePackage { zref-base }
\RequirePackage { zref-user }
\RequirePackage { zref-abspage }
\RequirePackage { l3keys2e }
\RequirePackage { ifdraft }

3 zref setup

For the purposes of the package, we need to store some information with the labels, some
of it standard, some of it not so much. So, we have to setup zref to do so.

Some basic properties are handled by zref itself, or some of its modules. The default
and page properties are provided by zref-base, while zref-abspage provides the abspage
property which gives us a safe and easy way to sort labels for page references.

The counter property, in most cases, will be just the kernel’s \@currentcounter,
set by \refstepcounter. However, not everywhere is it assured that \@currentcounter
gets updated as it should, so we need to have some means to manually tell zref-clever
what the current counter actually is. This is done with the currentcounter option, and
stored in \_zrefclever\_current\_counter\_tl, whose default is \@currentcounter.
\zref@newprop { zc@counter } \l__zrefclever_current_counter_tl
\zref@addprop \ZREF@mainlist { zc@counter }

The reference itself, stored by zref-base in the default property, is somewhat a
disputed real estate. In particular, the use of \labelformat (previously from varioref,
now in the kernel) will include there the reference “prefix” and complicate the job we are
trying to do here. Hence, we isolate \the(counter) and store it “clean” in thecounter
for reserved use. Since \currentlabel, which populates the default property, is more
reliable than \currentcounter, thecounter is meant to be kept as an option (ref
option), in case there’s need to use zref-clever together with \labelformat. Based on
the definition of \currentlabel done inside \refstepcounter in texdoc source2e,
section ltxref.dtx. We just drop the \p@... prefix.
\zref@newprop { thecounter }
\cs_if_exist:cTF { c@ \_zrefclever\_current\_counter\_tl }
\{ \use:c \{ the \_zrefclever\_current\_counter\_tl \} \}
\cs_if_exist:cTF { c@ \@currentcounter }
}
Much of the work of zref-clever relies on the association between a label’s “counter” and its “type” (see the User manual section on “Reference types”). Superficially examined, one might think this relation could just be stored in a global property list, rather than in the label itself. However, there are cases in which we want to distinguish different types for the same counter, depending on the document context. Hence, we need to store the “type” of the “counter” for each “label”. In setting this, the presumption is that the label’s type has the same name as its counter, unless it is specified otherwise by the countertype option, as stored in \l__zrefclever_counter_type_prop.

Since the default/thecounter and page properties store the “printed representation” of their respective counters, for sorting and compressing purposes, we are also interested in their numeric values. So we store them in \zc@cntval and \zc@pgval. For this, we use \texttt{c@\langle counter\rangle}, which contains the counter’s numerical value (see ‘texdoc source2e’, section ‘ltcounts.dtx’).

However, since many counters (may) get reset along the document, we require more than just their numeric values. We need to know the reset chain of a given counter, in order to sort and compress a group of references. Also here, the “printed representation” is not enough, not only because it is easier to work with the numeric values but, given we occasionally group multiple counters within a single type, sorting this group requires to know the actual counter reset chain.

Furthermore, even if it is true that most of the definitions of counters, and hence of their reset behavior, is likely to be defined in the preamble, this is not necessarily true. Users can create counters, newtheorems mid-document, and alter their reset behavior along the way. Was that not the case, we could just store the desired information at
begin document in a variable and retrieve it when needed. But since it is, we need to store the information with the label, with the values as current when the label is set.

Though counters can be reset at any time, and in different ways at that, the most important use case is the automatic resetting of counters when some other counter is stepped, as performed by the standard mechanisms of the kernel (optional argument of \newcounter, \@addtoreset, \counterwithin, and related infrastructure). The canonical optional argument of \newcounter establishes that the counter being created (the mandatory argument) gets reset every time the “enclosing counter” gets stepped (this is called in the usual sources “within-counter”, “old counter”, “super-counter”, “parent counter” etc.). This information is somewhat tricky to get. For starters, the counters which may reset the current counter are not retrievable from the counter itself, because this information is stored with the counter that does the resetting, not with the one that gets reset (the list is stored in \cl⟨counter⟩ with format \@elt{countera}\@elt{counterb}\@elt{counterc}, see ltcounts.dtx in texdoc source2e). Besides, there may be a chain of resetting counters, which must be taken into account: if \texttt{counterC} gets reset by \texttt{counterB}, and \texttt{counterB} gets reset by \texttt{counterA}, stepping the latter affects all three of them.

The procedure below examines a set of counters, those in \l__zrefclever-counter_resetters_seq, and for each of them retrieves the set of counters it resets, as stored in \cl⟨counter⟩, looking for the counter for which we are trying to set a label (\l__zrefclever_current_counter_tl, by default \@currentcounter, passed as an argument to the functions). There is one relevant caveat to this procedure: \l__zrefclever-counter_resetters_seq is populated by hand with the “usual suspects”, there is no way (that I know of) to ensure it is exhaustive. However, it is not that difficult to create a reasonable “usual suspects” list which, of course, should include the counters for the sectioning commands to start with, and it is easy to add more counters to this list if needed, with the option \texttt{counterresetters}. Unfortunately, not all counters are created alike, or reset alike. Some counters, even some kernel ones, get reset by other mechanisms (notably, the enumerate environment counters do not use the regular counter machinery for resetting on each level, but are nested nevertheless by other means). Therefore, inspecting \cl⟨counter⟩ cannot possibly fully account for all of the automatic counter resetting which takes place in the document. And there’s also no other “general rule” we could grab on for this, as far as I know. So we provide a way to manually tell zref-clever of these cases, by means of the \texttt{counterresetby} option, whose information is stored in \l__zrefclever-counter_resetby_prop. This manual specification has precedence over the search through \l__zrefclever-counter_resetters_seq, and should be handled with care, since there is no possible verification mechanism for this.

Recursively generate a sequence of “enclosing counters” values, for a given \texttt{(counter)} and leave it in the input stream. This function must be expandable, since it gets called from \texttt{zref@newprop} and is the one responsible for generating the desired information when the label is being set. Note that the order in which we are getting this information is reversed, since we are navigating the counter reset chain bottom-up. But it is very hard to do otherwise here where we need expandable functions, and easy to handle at the reading side.

\begin{verbatim}
\__zrefclever_get_enclosing_counters_value:n \{\__zrefclever_get_enclosing_counters_value:n \}
\end{verbatim}
Both $e$ and $f$ expansions work for this particular recursive call. I’ll stay with the $e$ variant, since conceptually it is what I want ($z$ itself is not expandable), and this package is anyway not compatible with older kernels for which the performance penalty of the $e$ expansion would ensue (helpful comment by Enrico Gregorio, aka ‘egreg’ at https://tex.stackexchange.com/q/611370/#comment1529282_611385).

\cs_generate_variant:Nn \__zrefclever_get_enclosing_counters_value:n { e }

(End definition for $\__zrefclever_get_enclosing_counters_value:n$)

$\__zrefclever_counter_reset_by:n$

Auxiliary function for $\__zrefclever_get_enclosing_counters_value:n$, and useful on its own standing. It is broken in parts to be able to use the expandable mapping functions. $\__zrefclever_counter_reset_by:n$ leaves in the stream the “enclosing counter” which resets (counter).

\cs_new:Npn \__zrefclever_counter_reset_by:n #1
{ \bool_if:nTF { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} } { \prop_item:Nn \l__zrefclever_counter_resetby_prop {#1} } { \seq_map_tokens:Nn \l__zrefclever_counter_resetters_seq { \__zrefclever_counter_reset_by_aux:nn {#1} } } }

\cs_new:Npn \__zrefclever_counter_reset_by_aux:nn #1#2
{ \cs_if_exist:cT { c@ #2 } { \tl_if_empty:cF { cl@ #2 } { \tl_map_tokens:cn { cl@ #2 } { \__zrefclever_counter_reset_by_auxi:nnn {#2} {#1} } } } }

\cs_new:Npn \__zrefclever_counter_reset_by_auxi:nnn #1#2#3
{ \str_if_eq:nnT {#2} {#3} { \tl_map_break:n { \seq_map_break:n {#1} } } }

(End definition for $\__zrefclever_counter_reset_by:n$)

Finally, we create the $zc\@enclval$ property, and add it to the $main$ property list.

\zref@neuprop { zc@enclval }

{ }
Another piece of information we need is the page numbering format being used by \thepage, so that we know when we can (or not) group a set of page references in a range. Unfortunately, page is not a typical counter in ways which complicates things. First, it does commonly get reset along the document, not necessarily by the usual counter reset chains, but rather with \pagenumbering or variations thereof. Second, the format of the page number commonly changes in the document (roman, arabic, etc.), not necessarily, though usually, together with a reset. Trying to “parse” \thepage to retrieve such information is bound to go wrong: we don’t know, and can’t know, what is within that macro, and that’s the business of the user, or of the documentclass, or of the loaded packages. The technique used by cleveref, which we borrow here, is simple and smart: store with the label what \thepage would return, if the counter \c@page was “1”. That does not allow us to sort the references, luckily however, we have \abspage which solves this problem. But we can decide whether two labels can be compressed into a range or not based on this format: if they are identical, we can compress them, otherwise, we can’t. To do so, we locally redefine \c@page to return “1”, thus avoiding any global spillovers of this trick. Since this operation is not expandable we cannot run it directly from the property definition. Hence, we use a shipout hook, and set \g__zrefclever_page_format_tl, which can then be retrieved by the starred definition of \zref@newprop*{zc@pgfmt}.

\tl_new:N \g__zrefclever_page_format_tl
\cs_new_protected:Npx \__zrefclever_page_format_aux: { \int_eval:n { 1 } }
\AddToHook { shipout / before }
{ \group_begin: \cs_set_eq:NN \c@page \__zrefclever_page_format_aux: \tl_gset:Nx \g__zrefclever_page_format_tl { \thepage } \group_end: }
\zref@newprop* { zc@pgfmt } \g__zrefclever_page_format_tl
\zref@addprop \ZREF@mainlist { zc@pgfmt }

Still some other properties which we don’t need to handle at the data provision side, but need to cater for at the retrieval side, are the ones from the zref-xr module, which are added to the labels imported from external documents, and needed to construct hyperlinks to them and to distinguish them from the current document ones at sorting and compressing: urluse, url and externaldocument.

4 Plumbing

4.1 Auxiliary

Just a convenience, since sometimes we just need one of the branches, and it is particularly easy to miss the empty F branch after a long T one.
4.2 Messages

\msg_new:nnn { zref-clever } { option-not-type-specific }
\{
  Option-’#1’-is-not-type-specific-\msg_line_context:.~
  Set-it-in-\iochar:N\zcLanguageSetup’-before-first-’type’-
  switch-or-as-package-option.
\}

\msg_new:nnn { zref-clever } { option-only-type-specific }
\{
  No-type-specified-for-option-’#1’-\msg_line_context:.~
  Set-it-after-’type’-switch.
\}

\msg_new:nnn { zref-clever } { key-requires-value }
\{
  The-’#1’-key-’#2’-requires-a-value-\msg_line_context:. }

\msg_new:nnn { zref-clever } { key-boolean-or-empty }
\{
  The-key-’#1’-only-accepts-the-values-’true’,-’false’-
  or-an-empty-value-\msg_line_context:. }

\msg_new:nnn { zref-clever } { language-declared }
\{
  Language-’#1’-is-already-declared-\msg_line_context:.-Nothing-to-do. }

\msg_new:nnn { zref-clever } { unknown-language-alias }
\{
  Language-’#1’-is-unknown-\msg_line_context:.-Can’t-alias-to-it.-
  See-documentation-for-’\iochar:N\zcDeclareLanguage’-and-
  ’\iochar:N\zcDeclareLanguageAlias’.
\}

\msg_new:nnn { zref-clever } { unknown-language-setup }
\{
  Language-’#1’-is-unknown-\msg_line_context:.-Can’t-set-it-up.-
  See-documentation-for-’\iochar:N\zcDeclareLanguage’-and-
  ’\iochar:N\zcDeclareLanguageAlias’.
\}

\msg_new:nnn { zref-clever } { unknown-language-opt }
\{
  Language-’#1’-is-unknown-\msg_line_context:.-Using-default.-
  See-documentation-for-’\iochar:N\zcDeclareLanguage’-and-
  ’\iochar:N\zcDeclareLanguageAlias’.
\}

\msg_new:nnn { zref-clever } { unknown-language-decl }
\{
  Can’t-set-declension-’#1’-for-unknown-language-’#2’-\msg_line_context:.~
  See-documentation-for-’\iochar:N\zcDeclareLanguage’-and-
  ’\iochar:N\zcDeclareLanguageAlias’.
\}

\msg_new:nnn { zref-clever } { language-no-decl-ref }
\{
  Language-’#1’-has-no-declared-declension-cases-\msg_line_context:.~
  Nothing-to-do-with-option-’d-#2’.
\}

\msg_new:nnn { zref-clever } { language-no-gender }

(End definition for \_zrefclever_if_package_loaded:n and \_zrefclever_if_class_loaded:n.)
\msg_new:nnn { zref-clever } { language-no-decl-setup }
{ Language~'#1'~has~no~declared~gender~\msg_line_context:.~ Nothing~to~do~with~option~'#2'=#3'. }
\msg_new:nnn { zref-clever } { unknown-decl-case }
{ Declension-case~'#1'~unknown~for~language~'#2'.~ Using~default-declension-case. }
\msg_new:nnn { zref-clever } { nudge-multitype }
{ Reference~with~multiple~types~\msg_line_context:.~ You~may~wish~to~separate~them~or~review~language~around~it. }
\msg_new:nnn { zref-clever } { nudge-comptosing }
{ Multiple~labels~have~been~compressed~into~singular~type-name~ for~type~'#1'.~\msg_line_context:. }
\msg_new:nnn { zref-clever } { nudge-plural-when-sg }
{ Option~'sg'~signals~that~a~singular~type-name~was~expected~\msg_line_context:.~ But~type~'#1'~has~plural~type-name. }
\msg_new:nnn { zref-clever } { gender-not-declared }
{ Language~'#1'~has~no~'#2'~gender~declared~\msg_line_context:. }
\msg_new:nnn { zref-clever } { nudge-gender-mismatch }
{ Gender~mismatch~for~type~'#1'.~\msg_line_context:.~ You've~specified~'g=#2'~but~type-name~is~'#3'~for~language~'#4'. }
\msg_new:nnn { zref-clever } { nudge-gender-not-declared-for-type }
{ You've~specified~'g=#1'~\msg_line_context:.~ But~gender~for~type~'#2'~is~not~declared~for~language~'#3'. }
\msg_new:nnn { zref-clever } { nudgeif-unknown-value }
{ Unknown~value~'#1'~for~'nudgeif'~option~\msg_line_context:. }
\msg_new:nnn { zref-clever } { option-document-only }
{ Option~'#1'~is~only~available~after~\ioe_char:N\begin\{document\}. }
\msg_new:nnn { zref-clever } { langfile-loaded }
{ Loaded~'#1'~language~file. }
\msg_new:nnn { zref-clever } { langfile-not-available }
{ Language~file~for~'#1'~not~available~\msg_line_context:. }
\msg_new:nnn { zref-clever } { unknown-language-load }
{ Language~'#1'~is~unknown~\msg_line_context:.~ Unable~to~load~language~file.~See~documentation~for~ \ioe_char:N\zcDeclareLanguage'.~and~}
\msg_new:nnn { zref-clever } { zref-property-undefined }
{
  Option-'ref=#1'-requested\msg_line_context:.-
  But-the-property-'#1'-is-not-declared,-falling-back-to-'default'.
}
\msg_new:nnn { zref-clever } { hyperref-preamble-only }
{
  Option-'hyperref'-only-available-in-the-preamble\msg_line_context:.-
  To-inhibit-hyperlinking-locally,-you-can-use-the-starred-version-of-
  \textbackslash{iow_char:N}\zcref'.
}
\msg_new:nnn { zref-clever } { missing-hyperref }
{ Missing-'hyperref'-package.-Setting-'hyperref=false'. }
\msg_new:nnn { zref-clever } { titleref-preamble-only }
{
  Option-'titleref'-only-available-in-the-preamble\msg_line_context:.-
  Did-you-mean-'ref=title'?.
}
\msg_new:nnn { zref-clever } { option-preamble-only }
{ Option-'#1'-only-available-in-the-preamble\msg_line_context:.
}
\msg_new:nnn { zref-clever } { unknown-compat-module }
{
  Unknown-compatibility-module-'#1'-given-to-option-'nocompat'.-
  Nothing-to-do.
}
\msg_new:nnn { zref-clever } { missing-zref-check }
{
  Option-'check'-requested\msg_line_context:.-
  But-package-'zref-check'-is-not-loaded,-can't-run-the-checks.
}
\msg_new:nnn { zref-clever } { missing-type }
{ Reference-type-undefined-for-label-'#1'\msg_line_context:.}
\msg_new:nnn { zref-clever } { missing-property }
{ Reference-property-'#1'-undefined-for-label-'#2'\msg_line_context:.}
\msg_new:nnn { zref-clever } { missing-name }
{ Reference-format-option-'#1'-undefined-for-type-'#2'\msg_line_context:.}
\msg_new:nnn { zref-clever } { missing-string }
{
  We-couldn't-find-a-value-for-reference-option-'#1'-\msg_line_context:.-
  But-we-should-have:-throw-a-rock-at-the-maintainer.
}
\msg_new:nnn { zref-clever } { single-element-range }
{ Range-for-type-'#1'-resulted-in-single-element\msg_line_context:.}
\msg_new:nnn { zref-clever } { compat-package }
{ Loaded-support-for-'#1'-package. }
\msg_new:nnn { zref-clever } { compat-class }
{ Loaded-support-for-'#1'-documentclass. }
\msg_new:nnn { zref-clever } { option-deprecated }
{ Option-'#1'-has-been-deprecated\msg_line_context:.\textbackslash{iow_newline:}
  Use-'#2'-instead. }
4.3 Data extraction

\_\_zrefclever_extract_default:Nnnn

Extract property \(\langle prop\rangle\) from \(\langle label\rangle\) and sets variable \(\langle tl\ var\rangle\) with extracted value. Ensure \texttt{\_\_zref@extractdefault} is expanded exactly twice, but no further to retrieve the proper value. In case the property is not found, set \(\langle tl\ var\rangle\) with \(\langle default\rangle\).

\begin{verbatim}
\cs_new_protected:Npn \_\_zrefclever_extract_default:Nnnn #1#2#3#4
\exp_args:NNNo \exp_args:NNo \tl_set:Nn #1 \{ #2 \} \tl_set:Nn #3 \{ #4 \}
\end{verbatim}

Extract property \(\langle prop\rangle\) from \(\langle label\rangle\). Ensure that, in the context of an x expansion, \texttt{\_\_zref@extractdefault} is expanded exactly twice, but no further to retrieve the proper value. Thus, this is meant to be use in an x expansion context, not in other situations. In case the property is not found, leave \(\langle default\rangle\) in the stream.

\begin{verbatim}
\cs_new:Npn \_\_zrefclever_extract_unexp:nnn #1#2#3
\exp_args:NNo \exp_args:No \exp_not:n { \zref@extractdefault {#1} {#2} {#3} }
\end{verbatim}

An internal version for \texttt{\_\_zref@extractdefault}.

\begin{verbatim}
\cs_new:Npn \_\_zrefclever_extract:nnn #1#2#3
\zref@extractdefault {#1} {#2} {#3}
\end{verbatim}

4.4 Reference format

For a general discussion on the precedence rules for reference format options, see Section “Reference format” in the User manual. Internally, these precedence rules are handled / enforced in \texttt{\_\_zrefclever_get_ref_opt_typeset:nN}, \texttt{\_\_zrefclever_get_ref_opt_font:nN}, and \texttt{\_\_zrefclever_type_name_setup:} which are the basic functions to retrieve proper values for reference format settings. The “fallback” settings are stored in \texttt{\_\_zrefclever_fallback_unknown_lang_prop}. 11
Store “current” type, language, and declension cases in different places for type-specific and language-specific options handling, notably in `\__zrefclever_provide_langfile:n`, `\zcRefTypeSetup`, and `\zclanguageSetup`. But also for language options retrieval, in `\__zrefclever_get_lang_opt_type:nnN` and `\__zrefclever_get_lang_opt_default:nnN`.

Lists of reference format related options in “categories”. Since these options are set in different scopes, and at different places, storing the actual lists in centralized variables makes the job not only easier later on, but also keeps things consistent.

Only “type names” are “necessarily type-specific”, which makes them somewhat special on the retrieval side of things. In short, they don’t have their values queried by `\__zrefclever_get_ref_opt_typeset:nN`, but by `\__zrefclever_type_name_setup:`.

}`
And, finally, some combined groups of the above variables, for convenience.

(End definition for \c__zrefclever_ref_options_necessarily_not_type_specific_seq and others.)

4.5 Languages

\g__zrefclever_languages_prop

Stores the names of known languages and the mapping from “language name” to “base
language name”. Whether or not a language or alias is known to zref-clever is decided
by its presence in this property list. A “base language” (loose concept here, meaning
just “the name we gave for the language file in that particular language”) is just like any
other one, the only difference is that the “language name” happens to be the same as the
“base language name”, in other words, it is an “alias to itself”.

\prop_new:N \g__zrefclever_languages_prop

(End definition for \g__zrefclever_languages_prop.)

\zcDeclareLanguage

Declare a new language for use with zref-clever. \langle language \rangle is taken to be both the “lan-
guage name” and the “base language name”. [[\langle options\rangle]] receive a k=v set of options,
with three valid options. The first, declension, takes the noun declension cases prefixes
for \langle language \rangle as a comma separated list, whose first element is taken to be the default
case. The second, gender, receives the genders for \langle language \rangle as comma separated list.
The third, allcaps, receives no value, and indicates that for \langle language \rangle all nouns must
be capitalized for grammatical reasons, in which case, the cap option is disregarded for
\langle language \rangle. If \langle language \rangle is already known, just warn. This implies a particular restric-
tion regarding [[\langle options\rangle]], namely that these options, when defined by the package,
cannot be redefined by the user. This is deliberate, otherwise the built-in language files
would become much too sensitive to this particular user input, and unnecessarily so.
\zcDeclareLanguage is preamble only.

\zcDeclareLanguage [[\langle options\rangle]] \langle \langle language \rangle \rangle
\NewDocumentCommand \zcDeclareLanguage { O { } m } {
\group_begin:
\tl_if_empty:nF {#2} {
    \prop_if_in:NnTF \g__zrefclever_languages_prop {#2} {
        { \msg_warning:nnn { zref-clever } { language-declared } {#2} }
        {
            \prop_gput:Nnn \g__zrefclever_languages_prop {#2} {#2}
            \prop_new:c { g__zrefclever_lang_ #2 _prop }
            \tl_set:Nn \l__zrefclever_base_language_tl {#2}
            \keys_set:nn { zref-clever / declarelang } {#1}
        }
    }
    \group_end:
}
\@onlypreamble \zcDeclareLanguage
(End definition for \zcDeclareLanguage.)
\zcDeclareLanguageAlias
\zcDeclareLanguageAlias
\NewDocumentCommand \zcDeclareLanguageAlias { m m } {
\group_begin:
\tl_if_empty:nF {#1} {
    \prop_if_in:NnTF \g__zrefclever_languages_prop {#2} {
        { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
        {
            \exp_args:NNnx \prop_gput:Nnn \g__zrefclever_languages_prop {#1} \prop_item:Nn \g__zrefclever_languages_prop {#2}
        }
    }
    { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
}
\group_end:
\@onlypreamble \zcDeclareLanguageAlias
(End definition for \zcDeclareLanguageAlias.)
\keys_define:nn { zref-clever / declarelang } {
\declension .code:n = 
\prop_gput:cn { g__zrefclever_lang_ \l__zrefclever_base_language_tl _prop } \declension } {#1}
\declension .value_required:n = true ,
\gender .code:n = 
\prop_gput:cn { g__zrefclever_lang_ \l__zrefclever_base_language_tl _prop }
}
Auxiliary function for \_zrefclever_zcref:nnn, responsible for processing options from \zcDeclareLanguage. It is necessary to separate them from the reference options machinery because their behavior is language dependent, but the language itself can also be set as an option (lang, value stored in \_zrefclever_ref_language_tl). Hence, we must validate these options after the reference options have been set. It is expected to be called right (or soon) after \keys_set:nnn in \_zrefclever_zcref:nnn, where current values for \_zrefclever_ref_language_tl and \_zrefclever_ref_decl_case_tl are in place.

Validate the declension case (d) option against the declared cases for the reference language. If the user value for the latter does not match the declension cases declared for the former, the function sets an appropriate value for \_zrefclever_ref_decl_case_tl, either using the default case, or clearing the variable, depending on the language setup. And also issues a warning about it.
Validate the gender \((g)\) option against the declared genders for the reference language. If the user value for the latter does not match the genders declared for the former, clear \l__zrefclever_ref_gender_tl\ and warn.

\begin{verbatim}
\exp_args:NNx \seq_set_from_clist:Nn \l__zrefclever_lang_gender_seq { \prop_item:cn { g__zrefclever_lang__l__zrefclever_base_language_tl_prop } } \tl_clear:N \l__zrefclever_ref_gender_tl
\end{verbatim}
Ensure the cap in \l__zrefclever_ref_options_prop is set to true when the language was declared with allcaps option.

\str_if_eq:eeT
{ \prop_item:cn
  { g__zrefclever_lang_
    \l__zrefclever_base_language_tl_prop
  } { true }
} { \prop_put:Nnn \l__zrefclever_ref_options_prop { cap } { true } }

If the language itself is not declared, we still have to issue declension and gender warnings, if d or g options were used.

\tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
{ \msg_warning:nnxx { zref-clever } { unknown-language-decl }
  \l__zrefclever_ref_decl_case_tl
}{ \l__zrefclever_ref_language_tl }
\tl_clear:N \l__zrefclever_ref_decl_case_tl
\tl_if_empty:NF \l__zrefclever_ref_gender_tl
{ \msg_warning:nnxxx { zref-clever } { language-no-gender } { g } \l__zrefclever_ref_gender_tl
  \l__zrefclever_ref_language_tl }
\tl_clear:N \l__zrefclever_ref_gender_tl
}

(End definition for \__zrefclever_process_language_options:)

4.6 Language files

Contrary to general options and type options, which are always local, language-specific settings are always global. Hence, the loading of built-in language files, as well as settings done with \zcLanguageSetup, should set the relevant variables globally.

The built-in language files and their related infrastructure are designed to perform “on the fly” loading of the language files, “lazily” as needed. Much like babel does for languages not declared in the preamble, but used in the document. This offers some convenience, of course, and that’s one reason to do it. But it also has the purpose of
parsimony, of “loading the least possible”. Therefore, we load at `begindocument` one single language (see `lang` option), as specified by the user in the preamble with the `lang` option or, failing any specification, the current language of the document, which is the default. Anything else is lazily loaded, on the fly, along the document.

This design decision has also implications to the form the language files assumed. As far as my somewhat impressionistic sampling goes, dictionary or localization files of the most common packages in this area of functionality, are usually a set of commands which perform the relevant definitions and assignments in the preamble or at `begindocument`. This includes `translator`, `translations`, but also `babel`’s `.ldf` files, and `biblatex`’s `.lbx` files.

I’m not really well acquainted with this machinery, but as far as I grasp, they all rely on some variation of `\ProvidesFile` and `\input`. And they can be safely `\input` without generating spurious content, because they rely on being loaded before the document has actually started. As far as I can tell, `babel`’s “on the fly” functionality is not based on the `.ldf` files, but on the `.ini` files, and on `\babelprovide`. And the `.ini` files are not in this form, but actually resemble “configuration files” of sorts, which means they are read and processed somehow else than with just `\input`. So we do the more or less the same here. It seems a reasonable way to ensure we can load language files on the fly robustly mid-document, without getting paranoid with the last bit of white-space in them, and without introducing any undue content on the stream when we cannot afford to do it.

Hence, `zref-clever`’s built-in language files are a set of key-value options which are read from the file, and fed to `\keys_set:nn{zref-clever/langfile}` by `\__zrefclever_provide_langfile:n`. And they use the same syntax and options as `\zcLanguageSetup` does. The language file itself is read with `\ExplSyntaxOn` with the usual implications for white-space and catcodes.

`\__zrefclever_provide_langfile:n` is only meant to load the built-in language files. For languages declared by the user, or for any settings to a known language made with `\zcLanguageSetup`, values are populated directly to a variable `\g__zrefclever_lang_⟨language⟩_prop`. Hence, there is no need to “load” anything in this case: definitions and assignments made by the user are performed immediately.

### Provide

- `\g__zrefclever_loaded_langfiles_seq` Used to keep track of whether a language file has already been loaded or not.

- `\__zrefclever_load_langfile_verbose_bool` Controls whether `\__zrefclever_provide_langfile:n` fails silently or verbosely in case of unknown languages or not found language files.

- `\__zrefclever_provide_langfile:n ⟨language⟩` Load language file for known ⟨language⟩ if it is available and if it has not already been loaded.

```latex
\__zrefclever_provide_langfile:n {(language)}
```

- `\cs_new_protected:Npn \__zrefclever_provide_langfile:n #1`
Even if we don’t have the actual language file, we register it as “loaded”. At this point, it is a known language, properly declared. There is no point in trying to load it multiple
times, because users cannot really provide the language files (well, technically they could, but we are working so they don’t need to, and have better ways to do what they want). And if the users had provided some language-specific options themselves, by means of \zrefLanguageSetup, everything would be in place, and they could use the \texttt{lang} option multiple times, and the \texttt{langfile-not-available} warning would never go away.

\begin{verbatim}
\seq_gput_right:NV \g__zrefclever_loaded_langfiles_seq
  \l__zrefclever_base_language_tl
\}

{ \bool_if:NT \l__zrefclever_load_langfile_verbose_bool
  { \msg_warning:nnn { zref-clever } { unknown-language-load } {#1} }
}
\esphack
\group_end:
\cs_generate_variant:Nn \__zrefclever_provide_langfile:n { x }
\end{verbatim}

\texttt{\__zrefclever_provide_langfile_verbose:n} Does the same as \texttt{\__zrefclever_provide_langfile:n}, but warns if the loading of the language file has failed.

\begin{verbatim}
\__zrefclever_provide_langfile_verbose:n \langle language\rangle
\cs_new_protected:Npn \__zrefclever_provide_langfile_verbose:n \#1
  { \group_begin:
    \bool_set_true:N \l__zrefclever_load_langfile_verbose_bool
    \__zrefclever_provide_langfile:n {#1}
    \group_end:
  }
\cs_generate_variant:Nn \__zrefclever_provide_langfile_verbose:n { x }
\end{verbatim}

\texttt{\__zrefclever_provide_langfile_verbose:n} Does exactly the same, but warns if the loading of the language file has failed.

\begin{verbatim}
\__zrefclever_provide_langfile_verbose:n \langle language\rangle
\cs_new_protected:Npn \__zrefclever_provide_langfile_verbose:n \#1
  { \group_begin:
    \bool_set_true:N \l__zrefclever_load_langfile_verbose_bool
    \__zrefclever_provide_langfile:n {#1}
    \group_end:
  }
\cs_generate_variant:Nn \__zrefclever_provide_langfile_verbose:n { x }
\end{verbatim}

\texttt{\__zrefclever_provide_langfile_verbose:n} Does exactly the same, but warns if the loading of the language file has failed.

\begin{verbatim}
\_zrefclever_provide_lang_file:nn
\_zrefclever_provide_lang_opt_type:nn
\_zrefclever_provide_lang_opt_default:nn
\end{verbatim}

\texttt{\_zrefclever_provide_lang_file:nn} A couple of auxiliary functions for the of \texttt{\_zrefclever/langfile} keys set in \texttt{\_zrefclever-_provide_langfile:n}. They respectively “provide” (i.e. set if it value does not exist, do nothing if it already does) “type-specific” and “default” language options. Both receive \langle key\rangle and \langle value\rangle as arguments, but \texttt{\_zrefclever_provide_lang_opt_type:nn} relies on the current value of \texttt{\_zrefclever_setup_type_tl}, as set by the type key.

\begin{verbatim}
\_zrefclever_provide_lang_opt_type:nn \langle key\rangle \langle value\rangle
\_zrefclever_provide_lang_opt_default:nn \langle key\rangle \langle value\rangle
\cs_new_protected:Npn \_zrefclever_provide_lang_opt_type:nn \#1\#2
  { \exp_args:Nnx \prop_gput_if_new:cnn
    { g__zrefclever_lang \l__zrefclever_base_language_tl _prop }
    { type- \l__zrefclever_setup_type_tl - #1 } \#2
  }
\cs_generate_variant:Nn \_zrefclever_provide_lang_opt_type:nn { nV }
\cs_new_protected:Npn \_zrefclever_provide_lang_opt_default:nn \#1\#2
  { }
\end{verbatim}
The set of keys for `zref-clever/langfile`, which is used to process the language files in `__zrefclever_provide_langfile:n`. The no-op cases for each category have their messages sent to “info”. These messages should not occur, as long as the language files are well formed, but they’re placed there nevertheless, and can be leveraged in regression tests.

\keys_define:nn { zref-clever / langfile }
{
  type .code:n =
  {
    \tl_if_empty:nTF {#1}
    { \tl_clear:N \l__zrefclever_setup_type_tl }
    { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
  },
  case .code:n =
  {
    \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
    { \msg_info:nxx { zref-clever } { language-no-decl-setup }
      { \l__zrefclever_base_language_tl } {#1}
    }
    { \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
      { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
      { \msg_info:nxx { zref-clever } { unknown-decl-case } {#1} }
      { \l__zrefclever_base_language_tl }
      \seq_get_left:NN \l__zrefclever_lang_declension_seq
      \l__zrefclever_lang_decl_case_tl
    }
  },
  case .value_required:n = true ,
  gender .code:n =
  {
    \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
    { \msg_info:nxxx { zref-clever } { language-no-gender }
      { \l__zrefclever_base_language_tl } { gender } {#1}
    }
    { \tl_if_empty:NTF \l__zrefclever_setup_type_tl
      { \msg_info:nnn { zref-clever }
        { option-only-type-specific } { gender }
      }
      { \msg_info:nxx { zref-clever } { unknown-gender } {#1} }
      { \l__zrefclever_base_language_tl }
      \seq_get_left:NN \l__zrefclever_lang_gender_seq
      \l__zrefclever_lang_gender_case_tl
    }
  },
\clist_clear:N \l_tmpa_clist
\clist_map_inline:nn {#1}
{\seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
{ \clist_put_right:Nn \l_tmpa_clist {##1} }
{ \msg_info:nnx { zref-clever }
  \{ gender-not-declared \}
  \{ \l__zrefclever_base_language_tl \} {##1} }
}
\clist_if_empty:NF \l_tmpa_clist
{ \exp_args:Nnx \__zrefclever_provide_lang_opt_type:nn
  \{ gender \} \{ \clist_use:Nn \l_tmpa_clist { , } \} }
}
}
,
gender .value_required:n = true ,
cap .choices:nn =
{ true , false }
{ \tl_if_empty:NTF \l__zrefclever_setup_type_tl
  \{ \l__zrefclever_provide_lang_opt_default:nV
    \{ cap \} \l_keys_choice_tl
  \}
  \{ \l__zrefclever_provide_lang_opt_type:nV
    \{ cap \} \l_keys_choice_tl
  \}
  },
cap .default:n = true ,
nocap .meta:n = \{ cap = false \},
nocap .value_forbidden:n = true ,
abbrev .choices:nn =
{ true , false }
{ \tl_if_empty:NTF \l__zrefclever_setup_type_tl
  \{ \l__zrefclever_provide_lang_opt_default:nV
    \{ abbrev \} \l_keys_choice_tl
  \}
  \{ \l__zrefclever_provide_lang_opt_type:nV
    \{ abbrev \} \l_keys_choice_tl
  \}
  },
abbrev .default:n = true ,
noabbrev .meta:n = { abbrev = false },
noabbrev .value_forbidden:n = true ,

\seq_map_inline:Nn \c__zrefclever_ref_options_necessarily_not_type_specific_seq
{ \keys_define:nn { zref-clever / langfile }
  { #1 .value_required:n = true ,
    #1 .code:n =
    { \tl_if_empty:NTF \l__zrefclever_setup_type_tl
      { \__zrefclever_provide_lang_opt_default:nn {#1} {##1} }
      { \msg_info:nmm { zref-clever }
        { option-not-type-specific } {#1}
      }
    },
  }
}

\seq_map_inline:Nn \c__zrefclever_ref_options_possibly_type_specific_seq
{ \keys_define:nn { zref-clever / langfile }
  { #1 .value_required:n = true ,
    #1 .code:n =
    { \tl_if_empty:NTF \l__zrefclever_setup_type_tl
      { \__zrefclever_provide_lang_opt_default:nn {#1} {##1} }
      { \__zrefclever_provide_lang_opt_type:nn {#1} {##1} }
    },
  }
}

\seq_map_inline:Nn \c__zrefclever_ref_options_type_names_seq
{ \keys_define:nn { zref-clever / langfile }
  { #1 .value_required:n = true ,
    #1 .code:n =
    { \tl_if_empty:NTF \l__zrefclever_setup_type_tl
      { \msg_info:nmm { zref-clever }
        { option-only-type-specific } {#1}
      }
    },
    { \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
      { \__zrefclever_provide_lang_opt_type:nn {#1} {##1} }
      { \__zrefclever_provide_lang_opt_type:nn
        { \l__zrefclever_lang_decl_case_tl - #1 } {##1} }
    }
  }
}

23
Fallback

All “strings” queried with \__zrefclever_get_ref_opt_typeset:nN – in practice, those in either \c__zrefclever_ref_options_necessarily_not_type_specific_seq or \c__zrefclever_ref_options_possibly_type_specific_seq – must have their values set for “fallback”, even if to empty ones, since this is what will be retrieved in the absence of a proper language-specific option, which will be the case if babel or polyglossia is loaded and sets a language which zref-clever does not know. On the other hand, “type names” are not looked for in “fallback”, since it is indeed impossible to provide any reasonable value for them for a “specified but unknown language”. Also “font” options – those in \c__zrefclever_ref_options_font_seq, and queried with \__zrefclever_get_ref_opt_font:nN – do not need to be provided here, since the later function sets an empty value if the option is not found.

\prop_new:N \g__zrefclever_fallback_unknown_lang_prop
\prop_gset_from_keyval:Nn \g__zrefclever_fallback_unknown_lang_prop
{ tpairsep = {,~} ,
  tlistsep = {,~} ,
  tlastsep = {,~} ,
  notesep = {~} ,
  namesep = {
obreakspace} ,
  pairsep = {,~} ,
  listsep = {,~} ,
  lastsep = {,~} ,
  rangesep = {\textendash} ,
  preref = {} ,
  postref = {} ,
}

Get language options

Get type-specific language option of \langle key \rangle for \langle type \rangle and \langle language \rangle, and store it in \langle tl variable \rangle if found. If not found, leave the \langle false code \rangle on the stream, in which case the value of \langle tl variable \rangle should not be relied upon.

\__zrefclever_get_lang_opt_type:nnnN {\langle language \rangle} {\langle type \rangle} {\langle key \rangle}
\langle tl variable \rangle \langle false code \rangle

\prg_new_protected_conditional:Nnn
\__zrefclever_get_lang_opt_type:nnnN #1#2#3#4 { F }
{ \prop_get:NnNTF \g__zrefclever_languages_prop #1 \l__zrefclever_base_language_tl
  \prop_get:cnNTF
  \l__zrefclever_base_language_tl_prop
  \l__zrefclever_base_language_tl _prop
  \l__zrefclever_base_language_tl
  \prg_return_true: }

24
\_\_zrefclever_get_lang_opt_default:nnNF

Get default language option of \textit{(key)} for \textit{(language)}, and store it in \textit{(tl variable)} if found. If not found, leave the \textit{(false code)} on the stream, in which case the value of \textit{(tl variable)} should not be relied upon.

\_\_zrefclever_get_fallback_unknown_lang_opt:nNF

Get fallback language option of \textit{(key)}, and store it in \textit{(tl variable)} if found. If not found, leave the \textit{(false code)} on the stream, in which case the value of \textit{(tl variable)} should not be relied upon.
4.7 Options
Auxiliary

If \langle value \rangle is empty, remove \langle key \rangle from \langle property list \rangle. Otherwise, add \langle key \rangle = \langle value \rangle to \langle property list \rangle.

\_zrefclever\_prop\_put\_non\_empty:Nnn \langle property list \rangle \{ \langle key \rangle \} \{ \langle value \rangle \}

\cs\_new\_protected:Npn \_zrefclever\_prop\_put\_non\_empty:Nnn \#1\#2\#3
\{
\tl\_if\_empty:nTF \{\#3\}
\{ \prop\_remove:Nn \#1 \{\#2\} \}
\{ \prop\_put:Nnn \#1 \{\#2\} \{\#3\} \}
\}

(End definition for \_zrefclever\_prop\_put\_non\_empty:Nnn.)

ref option

\_l\_zrefclever\_ref\_property\_tl stores the property to which the reference is being made. Note that one thing must be handled at this point: the existence of the property itself, as far as zref is concerned. This because typesetting relies on the check \zref\_if\_ref\_contains\_prop, which presumes the property is defined and silently expands the true branch if it is not (insightful comments by Ulrike Fischer at https://github.com/ho-tex/zref/issues/13). Therefore, before adding anything to \_l\_zrefclever\_ref\_property\_tl, check if first here with \zref\_if\_prop\_undefined: close it at the door.

\tl\_new:N \l\_zrefclever\_ref\_property\_tl
\keys\_define:nn \{ zref-clever / reference \}
\{
ref .code:n =
\{
\zref\_if\_prop\_undefined \#1\}
\{ \msg\_warning:nnn \{ zref-clever \} \{ zref\_property\_undefined \} \#1\}
\tl\_set:Nn \_l\_zrefclever\_ref\_property\_tl \{ default \}
\}
\{ \tl\_set:Nn \_l\_zrefclever\_ref\_property\_tl \#1 \}
\}
ref .initial:n = default ,
ref .value\_required:n = true ,
page .meta:n = \{ ref = page \},
page .value\_forbidden:n = true ,
\}

typeset option

\bool\_new:N \_l\_zrefclever\_typeset\_ref\_bool
\bool\_new:N \_l\_zrefclever\_typeset\_name\_bool
\keys\_define:nn \{ zref-clever / reference \}
\{
typeset .choice: ,
typeset / both .code:n =
\begin{verbatim}
{ \bool_set_true:N \l__zrefclever_typeset_ref_bool 
  \bool_set_true:N \l__zrefclever_typeset_name_bool }
, typeset / ref .code:n =
{ \bool_set_true:N \l__zrefclever_typeset_ref_bool 
  \bool_set_false:N \l__zrefclever_typeset_name_bool }
, typeset / name .code:n =
{ \bool_set_false:N \l__zrefclever_typeset_ref_bool 
  \bool_set_true:N \l__zrefclever_typeset_name_bool }
, typeset .initial:n = both ,
typeset .value_required:n = true ,
noname .meta:n = { typeset = ref },
noname .value_forbidden:n = true ,
noref .meta:n = { typeset = name },
noref .value_forbidden:n = true ,
}

\bool_new:N \l__zrefclever_typeset_sort_bool
\keys_define:nn { zref-clever / reference }
{ sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
  sort .initial:n = true ,
  sort .default:n = true ,
  nosort .meta:n = { sort = false },
  nosort .value_forbidden:n = true ,
}

\bool_new:N \l__zrefclever_typeset_sort_seq
\keys_define:nn { zref-clever / reference }
{ typesort .code:n =
  \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
  \seq_reverse:N \l__zrefclever_typesort_seq }
  \seq_new:N \l__zrefclever_typesort_seq
\keys_define:nn { zref-clever / reference }
{ typesort .code:n =
  \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
  \seq_reverse:N \l__zrefclever_typesort_seq }
  \seq_new:N \l__zrefclever_typesort_seq
\keys_define:nn { zref-clever / reference }
{ typesort .code:n =
  \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
  \seq_reverse:N \l__zrefclever_typesort_seq }
\end{verbatim}
comp option
\bool_new:N \l__zrefclever_typeset_compress_bool
\keys_define:nn { zref-clever / reference }
{
  comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
  comp .initial:n = true ,
  comp .default:n = true ,
  nocomp .meta:n = { comp = false },
  nocomp .value_forbidden:n = true ,
}
range option
\bool_new:N \l__zrefclever_typeset_range_bool
\keys_define:nn { zref-clever / reference }
{
  range .bool_set:N = \l__zrefclever_typeset_range_bool ,
  range .initial:n = false ,
  range .default:n = true ,
}
cap and capfirst options
\bool_new:N \l__zrefclever_capitalize_first_bool
\keys_define:nn { zref-clever / reference }
{
  cap .code:n =
  {
\tl_if_empty:nTF {#1}
  { \prop_remove:Nn \l__zrefclever_ref_options_prop { cap } }
  {
    \bool_lazy_or:nnTF
    { \str_if_eq_p:nn {#1} { true } }
    { \str_if_eq_p:nn {#1} { false } }
    { \prop_put:Nnn \l__zrefclever_ref_options_prop { cap } {#1} }
    { \msg_warning:nnn { zref-clever } { key-boolean-or-empty } {#1} }
  }
  }
  ,
  cap .default:n = true ,
  nocap .meta:n = { cap = false },
  nocap .value_forbidden:n = true ,
  capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,
  capfirst .initial:n = false ,
  capfirst .default:n = true ,
}
abbrev and noabbrevfirst options
\bool_new:N \l__zrefclever_noabbrev_first_bool
\keys_define:nn { zref-clever / reference }

\{ abbrev .code:n =
\{ \tl_if_empty:nTF {#1}
\{ \prop_remove:Nn \l__zrefclever_ref_options_prop { abbrev } \}
\{ \bool_lazy_or:nnTF
\{ \str_if_eq_p:nn {#1} { true } \}
\{ \str_if_eq_p:nn {#1} { false } \}
\{ \prop_put:Nnn \l__zrefclever_ref_options_prop { abbrev } {#1} \}
\}
\}
\}
abbrev .default:n = true ,
noabbrev .meta:n = { abbrev = false },
noabbrev .value_forbidden:n = true ,
noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
noabbrevfirst .initial:n = false ,
noabbrevfirst .default:n = true ,
\}
\keys_define:nn { zref-clever / reference }
\{ S .meta:n =
\{ \capfirst = true , noabbrevfirst = true \},
S .value_forbidden:n = true ,
\}
\bool_new:N \l__zrefclever_use_hyperref_bool
\bool_new:N \l__zrefclever_warn_hyperref_bool
\keys_define:nn { zref-clever / reference }
\{ hyperref .choice: ,
hyperref / auto .code:n =
\{ \bool_set_true:N \l__zrefclever_use_hyperref_bool
\bool_set_false:N \l__zrefclever_warn_hyperref_bool
\}
hyperref / true .code:n =
\{ \bool_set_true:N \l__zrefclever_use_hyperref_bool
\bool_set_true:N \l__zrefclever_warn_hyperref_bool
\}
hyperref / false .code:n =
\{ \bool_set_false:N \l__zrefclever_use_hyperref_bool
\bool_set_false:N \l__zrefclever_warn_hyperref_bool
\}
\AddToHook { begindocument }
{ \_\_zrefclever_if_package_loaded:nTF { hyperref }
  { \bool_if:NT \l\_\_zrefclever_use_hyperref_bool
    { \RequirePackage { zref-hyperref } }
  }
  { \bool_if:NT \l\_\_zrefclever_warn_hyperref_bool
    { \msg_warning:nn { zref-clever } { missing-hyperref } }
  \bool_set_false:N \l\_\_zrefclever_use_hyperref_bool
  }
\keys_define:nn { zref-clever / reference }
  { hyperref .code:n =
    { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }
  }
}
\str_new:N \l\_\_zrefclever_nameinlink_str
\keys_define:nn { zref-clever / reference }
  { nameinlink .choice:, nameinlink / true .code:n =
    { \str_set:Nn \l\_\_zrefclever_nameinlink_str { true } }
  , nameinlink / false .code:n =
    { \str_set:Nn \l\_\_zrefclever_nameinlink_str { false } }
  , nameinlink / single .code:n =
    { \str_set:Nn \l\_\_zrefclever_nameinlink_str { single } }
  , nameinlink / tsingle .code:n =
    { \str_set:Nn \l\_\_zrefclever_nameinlink_str { tsingle } }
  , nameinlink .initial:n = tsingle ,
  nameinlink .default:n = true ,
}
\bool_new:N \l\_\_zrefclever_preposinlink_bool
\keys_define:nn { zref-clever / reference }
  { preposinlink .bool_set:N = \l\_\_zrefclever_preposinlink_bool ,
    preposinlink .initial:n = false ,
    preposinlink .default:n = true ,
}
\str_new:N \l\_\_zrefclever_current_language_tl
\str_new:N \l\_\_zrefclever_main_language_tl
\l\_\_zrefclever_current_language_tl is an internal alias for babel’s \language or polyglossia’s \mainlanguage and, if none of them is loaded, we set it to english. \l\_\_zrefclever_main_language_tl is an internal alias for babel’s \bbl\mainlanguage or for polyglossia’s \mainlanguage, as the case may be. Note that for polyglossia we
get babel’s language names, so that we only need to handle those internally. \l__zrefclever_ref_language_tl is the internal variable which stores the language in which the reference is to be made.

The overall setup here seems a little roundabout, but this is actually required. In the preamble, we (potentially) don’t yet have values for the “current” and “main” document languages, this must be retrieved at a begindocument hook. The begindocument hook is responsible to get values for \l__zrefclever_current_language_tl and \l__zrefclever_main_language_tl, and to set the default for \l__zrefclever_ref_language_tl. Package options, or preamble calls to \zcsetup are also hooked at begindocument, but come after the first hook, so that the pertinent variables have been set when they are executed. Finally, we set a third begindocument hook, at begindocument/before, so that it runs after any options set in the preamble. This hook redefines the lang option for immediate execution in the document body, and ensures the current language’s language file gets loaded, if it hadn’t been already.

For the babel and polyglossia variables which store the “current” and “main” languages, see https://tex.stackexchange.com/a/233178, including comments, particularly the one by Javier Bezos. For the babel and polyglossia variables which store the list of loaded languages, see https://tex.stackexchange.com/a/281220, including comments, particularly PLK’s. Note, however, that languages loaded by \babelprovide, either directly, “on the fly”, or with the provide option, do not get included in \bbl@loaded.

\tl_new:N \l__zrefclever_ref_language_tl
\tl_new:N \l__zrefclever_current_language_tl
\tl_new:N \l__zrefclever_main_language_tl
\AddToHook { begindocument }{
  \__zrefclever_if_package_loaded:nTF { babel }{
    \tl_set:Nn \l__zrefclever_current_language_tl { \languagename }
    \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
  }
  \__zrefclever_if_package_loaded:nTF { polyglossia }{
    \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
    \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
  }
  \tl_set:Nn \l__zrefclever_ref_language_tl \l__zrefclever_current_language_tl
}

Provide default value for \l__zrefclever_ref_language_tl corresponding to option current, but do so outside of the \l3keys machinery (that is, instead of using .initial:n), so that we are able to distinguish when the user actually gave the option, in which case the language file loading is done verbosely, from when we are setting the default value (here), in which case the language file loading is done silently.

\tl_set:Nn \l__zrefclever_ref_language_tl{ \l__zrefclever_current_language_tl }
If any lang option has been given by the user, the corresponding language is already loaded, otherwise, ensure the default one (current) gets loaded early, but not verbosely.

Redefinition of the lang key option for the document body. Also, drop the verbose language file loading in the document body, as it can become intrusive depending on the use case, and does not provide much “juice” anyway: in \zcref missing names warnings will already ensue.

\keys_define:nn { zref-clever / reference }

\option{d}

For setting the declension case. Short for convenience and for not polluting the markup too much given that, for languages that need it, it may get to be used frequently.

@samcarter and Alan Munn provided useful comments about declension on the TeX.SX chat. Also, Florent Rougon’s efforts in this area, with the xcref package (https://github.com/frougon/xcref), have been an insightful source to frame the problem in general terms.
We just store the value at this point, which is validated by \_\_zrefclever_process_language_options: after \keys_set:n.

d .tl_set:N = \l__zrefclever_ref_decl_case_tl ,
d .value_required:n = true ,
}
}

\keys_define:nn { zref-clever / reference }

\bool_new:N \l__zrefclever_nudge_enabled_bool
\bool_new:N \l__zrefclever_nudge_multitype_bool
\bool_new:N \l__zrefclever_nudge_comptosing_bool
\bool_new:N \l__zrefclever_nudge_singular_bool
\bool_new:N \l__zrefclever_nudge_gender_bool
\tl_new:N \l__zrefclever_ref_gender_tl
\keys_define:nn { zref-clever / reference }

\keys_define:nn { zref-clever / reference }

{ nudge .choice: ,
  nudge / true .code:n =
  { \bool_set_true:N \l__zrefclever_nudge_enabled_bool } ,
  nudge / false .code:n =
  { \bool_set_false:N \l__zrefclever_nudge_enabled_bool } ,
  nudge / ifdraft .code:n =
  { \ifdraft
    { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
    { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
  } ,
  nudge / iffinal .code:n =
  { \ifoptionfinal
    { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
    { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
  } ,
  nudge .initial:n = false ,
  nudge .default:n = true ,
  nonudge .meta:n = { nudge = false } ,
  nonudge .value_forbidden:n = true ,
  nudgeif .code:n =
  { \bool_set_false:N \l__zrefclever_nudge_multitype_bool
    \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
    \bool_set_false:N \l__zrefclever_nudge_gender_bool
    \clist_map_inline:nn {#1}
    { \str_case:nnF (#1)
      { multitype }
      { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
    }
  }
\begin{verbatim}
{ comptosing }
{ \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
{ gender }
{ \bool_set_true:N \l__zrefclever_nudge_gender_bool }
{ all }

{ \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
{ \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
{ \bool_set_true:N \l__zrefclever_nudge_gender_bool }

{\msg_warning:nnn { zref-clever } { nudgeif-unknown-value } {##1} }

\msg_warning:nnn { zref-clever } { option-document-only } { g }

\AddToHook { begindocument }
{

\keys_define:nn { zref-clever / reference }
{

We just store the value at this point, which is validated by \_\_zrefclever_process_language_options: after \keys_set:nn.

\keys_define:nn { zref-clever / reference }
{

g .tl_set:N = \l__zrefclever_ref_gender_tl ,
g .value_required:n = true ,
}

\keys_define:nn { zref-clever / reference }
{

font option

\textit{font can’t be used as a package option, since the options get expanded by \LaTeX before being passed to the package (see \url{https://tex.stackexchange.com/a/489570}). It can be set in \zcref and, for global settings, with \zcssetup. Note that, technically, the “raw” options are already available as \_\_zrefclever_process_language_options: after \keys_set:nn.}

\keys_define:nn { zref-clever / reference }
{

\keys_define:nn { zref-clever / reference }
{

titleref .code:n = \{ \RequirePackage { zref-titleref } \} ,
titleref .value_forbidden:n = true ,
}

\keys_define:nn { zref-clever / reference }
{

titleref .code:n = \{ \RequirePackage { zref-titleref } \} ,
titleref .value_forbidden:n = true ,
}

35
\end{verbatim}
\AddToHook{begindocument}
{
  \keys_define:nn { zref-clever / reference }
  {
    titleref .code:n =
    { \msg_warning:nn { zref-clever } { titleref-preamble-only } }
  }
}

\tl_new:N \l__zrefclever_zcref_note_tl
\keys_define:nn { zref-clever / reference }
{
  note .tl_set:N = \l__zrefclever_zcref_note_tl ,
  note .value_required:n = true ,
}

\check option
Integration with zref-check.
\bool_new:N \l__zrefclever_zrefcheck_available_bool
\bool_new:N \l__zrefclever_zcref_with_check_bool
\keys_define:nn { zref-clever / reference }
{
  check .code:n = { \RequirePackage{zref-check} } ,
  check .value_required:n = true ,
}
\AddToHook{begindocument}
{
  \__zrefclever_if_package_loaded:nTF { zref-check }
  {
    \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
    \keys_define:nn { zref-clever / reference }
    {
      check .code:n =
      { \bool_set_true:N \l__zrefclever_zcref_with_check_bool
        \keys_set:nn { zref-check / zcheck } {#1}
      } ,
      check .value_required:n = true ,
    }
  }
  {
    \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
    \keys_define:nn { zref-clever / reference }
    {
      check .value_forbidden:n = false ,
      check .code:n =
      { \msg_warning:nn { zref-clever } { missing-zref-check } }
    }
  }
}
\endinput
countertype option

\l__zrefclever_counter_type_prop is used by zc@type property, and stores a mapping from “counter” to “reference type”. Only those counters whose type name is different from that of the counter need to be specified, since zc@type presumes the counter as the type if the counter is not found in \l__zrefclever_counter_type_prop.

\prop_new:N \l__zrefclever_counter_type_prop
\keys_define:nn { zref-clever / label } {
  countertype .code:n =
  {
    \keyval_parse:nnn
    {
      \msg_warning:nnnn { zref-clever } { key-requires-value } { countertype }
    }
    {
      \l__zrefclever_prop_put_non_empty:Nnn \l__zrefclever_counter_type_prop {#1}
    }
  },
  countertype .value_required:n = true ,
  countertype .initial:n =
  {
    subsection = section ,
    subsubsection = section ,
    subparagraph = paragraph ,
    enumi = item ,
    enumii = item ,
    enumiii = item ,
    enumiv = item ,
    mpfootnote = footnote ,
  } ,

One interesting comment I received (by Denis Bitouzé, at issue #1) about the most appropriate type for paragraph and subparagraph counters was that the reader of the document does not care whether that particular document structure element has been introduced by \paragraph or, e.g. by the \subsubsection command. This is a difference the author knows, as they’re using \LaTeX, but to the reader the difference between them is not really relevant, and it may be just confusing to refer to them by different names. In this case the type for paragraph and subparagraph should just be section. I don’t have a strong opinion about this, and the matter was not pursued further. Besides, I presume not many people would set secnumdepth so high to start with. But, for the time being, I left the \paragraph type for them, since there is actually a visual difference to the reader between the \subsubsection and \paragraph in the standard classes: up to the former, the sectioning commands break a line before the following text, while, from the later on, the sectioning commands and the following text are part of the same line. So, \paragraph is actually different from “just a shorter way to write \subsubsection”.

37
counterresetters option

\l__zrefclever_counter_resetters_seq is used by \l__zrefclever_counter_resetby:n to populate the zc@enclval property, and stores the list of counters which are potential “enclosing counters” for other counters. This option is constructed such that users can only _add_ items to the variable. There would be little gain and some risk in allowing removal, and the syntax of the option would become unnecessarily more complicated. Besides, users can already override, for any particular counter, the search done from the set in \l__zrefclever_counter_resetters_seq with the counterresetby option.

\seq_new:N \l__zrefclever_counter_resetters_seq
\keys_define:nn { zref-clever / label }
{ counterresetters .code:n =

  \clist_map_inline:nn {#1}
  { \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1} 
    \seq_put_right:Nn \l__zrefclever_counter_resetters_seq {##1} }
  \l__zrefclever_counter_resetters_seq {#1}
}
}
counterresetters .initial:n =
{ part ,
  chapter ,
  section ,
  subsection ,
  subsubsection ,
  paragraph ,
  subparagraph ,
},
counterresetters .value_required:n = true ,

counterresetby option

\l__zrefclever_counter_resetby_prop is used by \l__zrefclever_counter_resetby:n to populate the zc@enclval property, and stores a mapping from counters to the counter which resets each of them. This mapping has precedence in \l__zrefclever_counter_resetby:n over the search through \l__zrefclever_counter_resetters_seq.

\prop_new:N \l__zrefclever_counter_resetby_prop
\keys_define:nn { zref-clever / label }
{ counterresetby .code:n =
  \keyval_parse:nnn
  { \msg_warning:nnn { zref-clever } { key-requires-value } { counterresetby } }
}

38
The counters for the \texttt{enumerate} environment do not use the regular counter machinery for resetting on each level, but are nested nevertheless by other means, treat them as exception.

\begin{verbatim}
enumii = enumi ,
enumiii = enumii ,
enumiv = enumiii ,
\end{verbatim}

\texttt{currentcounter} option

\begin{verbatim}
\tl_new:N \l__zrefclever_current_counter_tl
\keys_define:nn { zref-clever / label }
{ currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
currentcounter .value_required:n = true ,
currentcounter .initial:n = \@currentcounter ,
}
\end{verbatim}

\texttt{nocompat} option

\begin{verbatim}
\bool_new:N \g__zrefclever_nocompat_bool
\seq_new:N \g__zrefclever_nocompat_modules_seq
\keys_define:nn { zref-clever / reference }
{ nocompat .code:n =
\begin{verbatim}
{ \tl_if_empty:nTF {#1}
  { \bool_gset_true:N \g__zrefclever_nocompat_bool }
  { \clist_map_inline:n {#1}
    { \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {##1}
      { \seq_gput_right:Nn \g__zrefclever_nocompat_modules_seq {##1} }
    }
  }
}
\end{verbatim}
\end{verbatim}

39
\AddToHook { begindocument }
{
\keys_define:nn { zref-clever / reference }
{
  nocompat .code:n =
{
  \msg_warning:nnn { zref-clever }
  { option-preamble-only } { nocompat }
}
}
}
\AtEndOfPackage
{
\AddToHook { begindocument }
{
\seq_map_inline:Nn \g__zrefclever_nocompat_modules_seq

\__zrefclever_compat_module:nn ⟨module⟩ ⟨code⟩

(End definition for \__zrefclever_compat_module:nn.)

Reference options

This is a set of options related to reference typesetting which receive equal treatment and, hence, are handled in batch. Since we are dealing with options to be passed to \zcref or to \zcsetup or at load time, only “not necessarily type-specific” options are pertinent here. However, they may either be type-specific or language-specific, and thus must be stored in a property list, \l__zrefclever_ref_options_prop, in order to
be retrieved from the option name by \texttt{\_zrefclever\_get\_ref\_opt\_typeset:nN} and \texttt{\_zrefclever\_get\_ref\_opt\_font:nN} according to context and precedence rules.

The keys are set so that any value, including an empty one, is added to \texttt{\_zrefclever\_ref\_options\_prop}, while a key with no value removes the property from the list, so that these options can then fall back to lower precedence levels settings. For discussion about the used technique, see Section 5.2.

\begin{verbatim}
\prop_new:N \l__zrefclever_ref_options_prop
\seq_map_inline:Nn \c__zrefclever_ref_options_reference_seq
{ \keys_define:nn { zref-clever / reference } { #1 .default:V = \c_novalue_tl , #1 .code:n = { \tl_if_novalue:nTF {##1} { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} } { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} } } , } \keys_define:nn { zref-clever / reference } { refpre .code:n = { % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha. \msg_warning:nnnn { zref-clever }{ option-deprecated } { refpre } { preref } } , refpos .code:n = { % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha. \msg_warning:nnnn { zref-clever }{ option-deprecated } { refpos } { postref } } } }
\keys_define:nn { } { zref-clever / zcsetup .inherit:n = { zref-clever / label , zref-clever / reference , } }
\end{verbatim}

Package options

The options have been separated in two different groups, so that we can potentially apply them selectively to different contexts: label and reference. Currently, the only use of this selection is the ability to exclude label related options from \texttt{\zcsetup}'s options. Anyway, for load-time package options and for \texttt{\zcsetup} we want the whole set, so we aggregate the two into \texttt{zref-clever/zcsetup}, and use that here.

\begin{verbatim}
\keys_define:nn { } { zref-clever / zcsetup .inherit:n = { zref-clever / label , zref-clever / reference , } }
\end{verbatim}
\ProcessKeysOptions { zref-clever / zcsetup }

5 Configuration

5.1 \zcsetup

\zcsetup { Provide \zcsetup. }
\zcsetup{(options)}
\NewDocumentCommand \zcsetup { m }
\{ \_\_zrefclever_zcsetup:n \{#1\} \}
(End definition for \zcsetup.)
\_\_zrefclever_zcsetup:n { A version of \zcsetup for internal use with variant. }
\_\_zrefclever_zcsetup:n{(options)}
\cs_new_protected:Npn \_\_zrefclever_zcsetup:n #1
\{ \keys_set:nn { zref-clever / zcsetup } \{#1\} \}
\cs_generate_variant:Nn \_\_zrefclever_zcsetup:n { x }
(End definition for \_\_zrefclever_zcsetup:n.)

5.2 \zcRefTypeSetup

\zcRefTypeSetup { is the main user interface for “type-specific” reference formatting. Settings done by this command have a higher precedence than any language-specific setting, either done at \zcLanguageSetup or by the package’s language files. On the other hand, they have a lower precedence than non type-specific general options. The ⟨options⟩ should be given in the usual key=val format. The ⟨type⟩ does not need to pre-exist, the property list variable to store the properties for the type gets created if need be. }
\zcRefTypeSetup {⟨type⟩} {⟨options⟩}
\NewDocumentCommand \zcRefTypeSetup { m m }
\{ \prop_if_exist:cF { l\_\_zrefclever_type_ #1 _options_prop } 
\{ \prop_new:c { l\_\_zrefclever_type_ #1 _options_prop } \}
\\tl_set:Nn \l\_\_zrefclever_setup_type_tl \{#1\}
\\keys_set:nn { zref-clever / typesetup } \{#2\}
\}
(End definition for \zcRefTypeSetup.)

Inside \zcRefTypeSetup any of the options can receive empty values, and those values, if they exist in the property list, will override language-specific options, regardless of their emptiness. In principle, we could live with the situation of, once a setting has been made in \l\_\_zrefclever_type_<type>_options_prop or in \l\_\_zrefclever-_ref_options_prop it stays there forever, and can only be overridden by a new value at the same precedence level or a higher one. But it would be nice if an user can “unset” an option at either of those scopes to go back to the lower precedence level of the language-specific options at any given point. So both in \zcRefTypeSetup and
in setting reference options (see Section 4.7), we leverage the distinction of an “empty
valued key” (key= or key={}) from a “key with no value” (key). This distinction is
captured internally by the lower-level key parsing, but must be made explicit at \keys-
set:nn by means of the .default:V property of the key in \keys_define:nn. For the
technique, by Jonathan P. Spratte, aka ‘Skillmon’, and some discussion about it, including
further insights by Phelype Oleinik, see https://tex.stackexchange.com/q/614690

\keys_define:nn { zref-clever / typesetup }
\{
cap .code:n =
{ \tl_if_empty:nTF {#1}
{ \prop_remove:cn
{ l__zrefclever_type_
\l__zrefclever_setup_type_tl _options_prop
 }{ cap }
}
{ bool_lazy_or:nnTF
{ \str_if_eq_p:nn {#1} { true } }
{ \str_if_eq_p:nn {#1} { false } }
{ \prop_put:cnn
{ l__zrefclever_type_
\l__zrefclever_setup_type_tl _options_prop
 }{ cap } {#1}
}
{ \msg_warning:nnn { zref-clever } { key-boolean-or-empty } {#1}
}
},
cap .default:n = true ,
nocap .meta:n = { cap = false } ,
nocap .value_forbidden:n = true ,
abbrev .code:n =
{ \tl_if_empty:nTF {#1}
{ \prop_remove:cn
{ l__zrefclever_type_
\l__zrefclever_setup_type_tl _options_prop
 }{ abbrev }
}
}
\begin{verbatim}
{ \bool_lazy_or:nnTF
  \str_if_eq_p:nn {#1} { true } }
{ \str_if_eq_p:nn {#1} { false } }
{ \prop_put:cnn
  { l__zrefclever_type_
    \l__zrefclever_setup_type_tl _options_prop
  } {#1}
}
{ \msg_warning:nnn { zref-clever } { key-boolean-or-empty } {#1}
}
{ \prop_remove:cn
  { l__zrefclever_type_
    \l__zrefclever_setup_type_tl _options_prop
  }
}
{ abbrev } {#1}
{ noabbrev .meta:n = { abbrev = false },
  noabbrev .value_forbidden:n = true ,
}
abbrev .default:n = true ,
noabbrev .meta:n = { abbrev = false },
noabbrev .value_forbidden:n = true ,
{ \keys_define:nn { zref-clever / typesetup }
  { #1 .code:n =
  { \msg_warning:nnn { zref-clever } { option-not-type-specific } {#1}
    ,
  }
}
{ \keys_define:nn { zref-clever / typesetup }
  { #1 .default:V = \c_novalue_tl ,
    #1 .code:n =
    { \tl_if_novalue:nTF {##1}
      { \prop_remove:cn
        { l__zrefclever_type_
          \l__zrefclever_setup_type_tl _options_prop
        }
      }
    }
  }
}
\end{verbatim}
5.3 \zcLanguageSetup

\zcLanguageSetup is the main user interface for “language-specific” reference formatting, be it “type-specific” or not. The difference between the two cases is captured by the type key, which works as a sort of a “switch”. Inside the (options) argument of \zcLanguageSetup, any options made before the first type key declare “default” (non type-specific) language options. When the type key is given with a value, the options following it will set “type-specific” language options for that type. The current type can be switched off by an empty type key. \zcLanguageSetup is preamble only.
A couple of auxiliary functions for the of `zref-clever/langsetup` keys set in `zcLanguageSetup`. They respectively declare (unconditionally set) “type-specific” and “default” language-specific options.

\_\_zrefclever\_declare\_lang\_opt\_type:nnnn \{\langle language\rangle\} \{\langle type\rangle\} \{\langle key\rangle\} \{\langle value\rangle\}
\_\_zrefclever\_declare\_lang\_opt\_default:nnn \{\langle language\rangle\} \{\langle key\rangle\} \{\langle value\rangle\}

(End definition for `zcLanguageSetup`.)

The set of keys for `zref-clever/langsetup`, which is used to set language-specific options in `zcLanguageSetup`.

\keys_define:nn \{ zref-clever / langsetup \}
{type = \dottedline{46}
\begin{verbatim}
\{
  \tl_if_empty:nTF {
    \tl_clear:N \l__zrefclever_setup_type_tl
  }
  \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
\},
\}
\end{verbatim}
\l__zrefclever_base_language_tl
\l__zrefclever_setup_type_tl
{ gender } { \clist_use:Nn \l_tmpa_clist { , } }
\}
\}
\)
\}
\)
\}
\}
\)
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\l_if_empty:NTF \l__zrefclever_setup_type_tl
{ \l__zrefclever_declare_lang_opt_default:Vnn
 \l__zrefclever_setup_type_tl
{#1} {##1}
}
{ msg_warning:nn \zref-clever
 { option-not-type-specific } {#1}
}
,\l__zrefclever_base_language_tl
{#1} {##1}
}
{\msg_warning:nnn { zref-clever }
 { option-not-type-specific } {#1}
}

\seq_map_inline:Nn \c__zrefclever_ref_options_possibly_type_specific_seq
{\keys_define:nn { zref-clever / langsetup }
{#1 .value_required:n = true ,
#1 .code:n =
{
 \l_if_empty:NTF \l__zrefclever_setup_type_tl
{ \l__zrefclever_declare_lang_opt_default:Vnn
 \l__zrefclever_setup_type_tl
{#1} {##1}
}
{ \l__zrefclever_declare_lang_opt_type:VVnn
 \l__zrefclever_setup_type_tl
{#1} {##1}
}
}
,\l__zrefclever_base_language_tl
{#1} {##1}
}
,\keys_define:nn { zref-clever / langsetup }
{ refpre .code:n =
{ \msg_warning:nnn { zref-clever }
 { option-deprecated } { preref }
},
refpos .code:n =
{ \msg_warning:nnn { zref-clever }
 { option-deprecated } { postref }
} ,\keys_define:nn { zref-clever / langsetup }
{ refpre .code:n =
{ % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
 \msg_warning:nnn { zref-clever }
 { option-deprecated } { preref }
},
refpos .code:n =
{ % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
 \msg_warning:nnn { zref-clever }
 { option-deprecated } { postref }
} ,
6 User interface

6.1 \zcref

\zcref The main user command of the package.

\zcref*\{\textsf{options}\}\{\textsf{labels}\}

\NewDocumentCommand \zcref \s O { } m
{ \zref@wrapper@babel \__zrefclever_zcref:nnn {#3} {#1} {#2} }

(End definition for \zcref.)

\__zrefclever_zcref:nnn An intermediate internal function, which does the actual heavy lifting, and places
\{\textsf{labels}\} as first argument, so that it can be protected by \zref@wrapper@babel in \zcref.

\__zrefclever_zcref:nnn \{\textsf{labels}\} \{\textsf{*}\} \{\textsf{options}\}

\cs_new_protected:Npn \__zrefclever_zcref:nnn \#1\#2\#3
{ \group_begin: }
Set options.
\keys_set:nn { zref-clever / reference } {#3} 

Store arguments values.
\seq_set_from_clist:Nn \__zrefclever_zcref_labels_seq {#1}
\bool_set:Nn \__zrefclever_link_star_bool {#2}

Ensure language file for reference language is loaded, if available. We cannot rely on \keys_set:nn for the task, since if the lang option is set for current, the actual language may have changed outside our control. \__zrefclever_provide_langfile:x does nothing if the language file is already loaded.
\__zrefclever_provide_langfile:x { \l__zrefclever_ref_language_tl }

Process \zcDeclareLanguage options.
\__zrefclever_process_language_options:

Integration with zref-check.
\bool_lazy_and:nnT
\__zrefclever_zrefcheck_available_bool
\__zrefclever_zcref_with_check_bool
\zrefcheck_zcref_beg_label: 

Sort the labels.
\bool_lazy_or:nnT
\__zrefclever_typeset_sort_bool
\__zrefclever_typeset_range_bool
\__zrefclever_sort_labels: 

Typeset the references. Also, set the reference font, and group it, so that it does not leak to the note.
\group_begin:
\__zrefclever_ref_typeset_font_tl
\__zrefclever_typeset_refs:
\group_end:

Typeset note. 
\tl_if_empty:NF \__zrefclever_zcref_note_tl
\__zrefclever_get_ref_opt_typeset:nN { notesep } \l_tmpa_tl
\l_tmpa_tl
\__zrefclever_zcref_note_tl

Integration with zref-check.
\bool_lazy_and:nnT
\__zrefclever_zrefcheck_available_bool
\__zrefclever_zcref_with_check_bool
\zrefcheck_zref_end_label_maybe:
\zrefcheck_zref_run_checks_on_labels:n
\__zrefclever_zcref_labels_seq 


Integration with \texttt{mathtools}.
\begin{verbatim}
\bool_if:NT \l__zrefclever_mathtools_shovonlyrefs_bool
\{ \__zrefclever_mathtools_shovonlyrefs:n
\{ \l__zrefclever_zcref_labels_seq \}
\}
\group_end:
\end{verbatim}

(End definition for \texttt{\_zrefclever_zcref:nnnn}.)

\begin{verbatim}
\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool
\seq_new:N \l__zrefclever_zcref_labels_seq
\bool_new:N \l__zrefclever_link_star_bool
\end{verbatim}

(End definition for \texttt{\_zrefclever_zcref_labels_seq} and \texttt{\_zrefclever_link_star_bool}.)

6.2 \texttt{\zcpageref}

\begin{verbatim}
\zcpageref A \texttt{\pageref} equivalent of \texttt{\zcref}.
\zcpageref\([\ast]\)[\{options\}]{\{labels\}}
\end{verbatim}

\begin{verbatim}
\NewDocumentCommand \zcpageref { s O { } m }
{ \IfBooleanTF {#1}
\{ \zcref\*[#2, ref = page] {#3} \}
\{ \zcref \[#2, ref = page\] \{#3\} \}
}
\end{verbatim}

(End definition for \texttt{\zcpageref}.)

7 Sorting

Sorting is certainly a “big task” for \texttt{zref-clever} but, in the end, it boils down to “carefully done branching”, and quite some of it. The sorting of “page” references is very much lightened by the availability of \texttt{abspage} from the \texttt{zref-abspage} module, which offers “just what we need” for our purposes. The sorting of “default” references falls on two main cases: i) labels of the same type; ii) labels of different types. The first case is sorted according to the priorities set by the \texttt{typesort} option or, if that is silent for the case, by the order in which labels were given by the user in \texttt{\zcref}. The second case is the most involved one, since it is possible for multiple counters to be bundled together in a single reference type. Because of this, sorting must take into account the whole chain of “enclosing counters” for the counters of the labels at hand.

\begin{verbatim}
\l__zrefclever_label_type_a_tl
\l__zrefclever_label_type_b_tl
\l__zrefclever_label_enclval_a_tl
\l__zrefclever_label_enclval_b_tl
\l__zrefclever_label_extdoc_a_tl
\l__zrefclever_label_extdoc_b_tl
\tl_new:N \l__zrefclever_label_type_a_tl
\tl_new:N \l__zrefclever_label_type_b_tl
\tl_new:N \l__zrefclever_label_enclval_a_tl
\tl_new:N \l__zrefclever_label_enclval_b_tl
\tl_new:N \l__zrefclever_label_extdoc_a_tl
\tl_new:N \l__zrefclever_label_extdoc_b_tl
\end{verbatim}

Auxiliary variables, for use in sorting, and some also in typesetting. Used to store reference information – label properties – of the “current” (a) and “next” (b) labels.
\_zrefclever\_sort\_decided\_bool

Auxiliary variable for \_zrefclever\_sort\_default\_same\_type:nn, signals if the sorting between two labels has been decided or not.
\bool\_new:N \_zrefclever\_sort\_decided\_bool

(End definition for \_zrefclever\_sort\_decided\_bool.)

\_zrefclever\_sort\_prior\_a\_int
\_zrefclever\_sort\_prior\_b\_int

Auxiliary variables for \_zrefclever\_sort\_default\_different\_types:nn. Store the sort priority of the “current” and “next” labels.
\int\_new:N \_zrefclever\_sort\_prior\_a\_int
\int\_new:N \_zrefclever\_sort\_prior\_b\_int

(End definition for \_zrefclever\_sort\_prior\_a\_int and \_zrefclever\_sort\_prior\_b\_int.)

\_zrefclever\_label\_types\_seq

Stores the order in which reference types appear in the label list supplied by the user in \zcref. This variable is populated by \_zrefclever\_label\_type\_put\_new\_right:n at the start of \_zrefclever\_sort\_labels:. This order is required as a “last resort” sort criterion between the reference types, for use in \_zrefclever\_sort\_default\_different\_types:nn.
\seq\_new:N \_zrefclever\_label\_types\_seq

(End definition for \_zrefclever\_label\_types\_seq.)

\_zrefclever\_sort\_labels:
The main sorting function. It does not receive arguments, but it is expected to be run inside \_zrefclever\_zcref:mmm where a number of environment variables are to be set appropriately. In particular, \_zrefclever\_zcref\_labels\_seq should contain the labels received as argument to \zcref, and the function performs its task by sorting this variable.
\cs\_new\_protected:Npn \_zrefclever\_sort\_labels:
{S

Store label types sequence.
\seq\_clear:N \_zrefclever\_label\_types\_seq
\tl\_if\_eq:NnF \_zrefclever\_ref\_property\_tl { page }
{S
 \seq\_map\_function:NN \_zrefclever\_zcref\_labels\_seq
 \_zrefclever\_label\_type\_put\_new\_right:n
\}

Sort.
\seq\_sort:Nn \_zrefclever\_zcref\_labels\_seq
{S
 \zref@ifrefundefined {##1}
{S
 \zref@ifrefundefined {##2}
{S
 % Neither label is defined.
 \sort\_return\_same:

 % The second label is defined, but the first isn’t, leave the
 % undefined first (to be more visible).
 \sort\_return\_same:

53
\zref@ifrefundefined{##2}
{
  \% The first label is defined, but the second isn’t, bring the
  \% second forward.
  \sort_return_swapped:
}
*
{
  \% The interesting case: both labels are defined. References
  \% to the "default" property or to the "page" are quite
  \% different with regard to sorting, so we branch them here to
  \% specialized functions.
  \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
    { \__zrefclever_sort_page:nn {##1} {##2} }
    { \__zrefclever_sort_default:nn {##1} {##2} }
}
*
\__zrefclever_label_type_put_new_right:n
Auxiliary function used to store the order in which reference types appear in the label list supplied by the user in \zcref. It is expected to be run inside \__zrefclever_sort_labels:, and stores the types sequence in \l__zrefclever_label_types_seq. I have tried to handle the same task inside \seq_sort:Nn in \__zrefclever_sort_labels: to spare mapping over \l__zrefclever_zcref_labels_seq, but it turned out it not to be easy to rely on the order the labels get processed at that point, since the variable is being sorted there. Besides, the mapping is simple, not a particularly expensive operation. Anyway, this keeps things clean.
\__zrefclever_label_type_put_new_right:n {⟨label⟩}
\cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
  { \__zrefclever_extract_default:Nnnn \l__zrefclever_label_type_a_tl {#1} { zc@type } { \c_empty_tl }
    \seq_if_in:NVF \l__zrefclever_label_types_seq \l__zrefclever_label_type_a_tl
      { \seq_put_right:NV \l__zrefclever_label_types_seq \l__zrefclever_label_type_a_tl }
  }
\__zrefclever_sort_default:nn
The heavy-lifting function for sorting of defined labels for “default” references (that is, a standard reference, not to “page”). This function is expected to be called within the sorting loop of \__zrefclever_sort_labels: and receives the pair of labels being considered for a change of order or not. It should always “return” either \sort_return_same: or \sort_return_swapped:.
\__zrefclever_sort_default:nn {⟨label a⟩} {⟨label b⟩}
\cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
{\__zrefclever_extract_default:Nnnn\l__zrefclever_label_type_a_tl {#1} { zc@type } { zc@missingtype }
\__zrefclever_extract_default:Nnnn\l__zrefclever_label_type_b_tl {#2} { zc@type } { zc@missingtype }
\tl_if_eq:NNTF\l__zrefclever_label_type_a_tl\l__zrefclever_label_type_b_tl
{ \__zrefclever_sort_default_same_type:nn {#1} {#2} }
{ \__zrefclever_sort_default_different_types:nn {#1} {#2} }
}

(End definition for \__zrefclever_sort_default:nn.)
\__zrefclever_sort_default_same_type:nn {⟨label a⟩} {⟨label b⟩}
\cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
{\__zrefclever_extract_default:Nnnn\l__zrefclever_label_enclval_a_tl {#1} { zc@enclval } { \c_empty_tl }
\tl_reverse:N \l__zrefclever_label_enclval_a_tl
\__zrefclever_extract_default:Nnnn\l__zrefclever_label_enclval_b_tl {#2} { zc@enclval } { \c_empty_tl }
\tl_reverse:N \l__zrefclever_label_enclval_b_tl
\__zrefclever_extract_default:Nnnn\l__zrefclever_label_extdoc_a_tl {#1} { externaldocument } { \c_empty_tl }
\__zrefclever_extract_default:Nnnn\l__zrefclever_label_extdoc_b_tl {#2} { externaldocument } { \c_empty_tl }
\bool_set_false:N \l__zrefclever_sort_decided_bool
\% First we check if there’s any “external document” difference (coming
\% from ‘zref-xr’) and, if so, sort based on that.
\tl_if_eq:NNF\l__zrefclever_label_extdoc_a_tl\l__zrefclever_label_extdoc_b_tl
{ \__zrefclever_sort_default_same_type:nn {#1} {#2} }
{ \__zrefclever_sort_default_different_types:nn {#1} {#2} }
\bool_set_false:N \l__zrefclever_sort_decided_bool
\bool_if:nTF\tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
{ \bool_set_true:N \l__zrefclever_sort_decided_bool
\sort_return_same:
}
{ \bool_if:nTF
! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
\tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl

55
\bool_set_true:N \l__zrefclever_sort_decided_bool
\sort_return_swapped:
\bool_set_true:N \l__zrefclever_sort_decided_bool
% Two different "external documents": last resort, sort by the
% document name itself.
\str_compare:eNeTF
{ \l__zrefclever_label_extdoc_b_tl } <
{ \l__zrefclever_label_extdoc_a_tl }
{ \sort_return_swapped: }
{ \sort_return_same: }
\bool_until_do:Nn \l__zrefclever_sort_decided_bool
{ \bool_if:nTF
{ % Both are empty: neither label has any (further) "enclosing
% counters" (left).
\tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
\tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
}
{ \bool_set_true:N \l__zrefclever_sort_decided_bool
\int_compare:nNnTF
{ \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
> { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
{ \sort_return_swapped: }
{ \sort_return_same: }
}
{ \bool_if:nTF
{ % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
\tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
}
{ \bool_set_true:N \l__zrefclever_sort_decided_bool
\int_compare:nNnTF
{ \__zrefclever_extract:nnn {#1} { zc@cntval } { } }
> { \tl_head:N \l__zrefclever_label_enclval_b_tl }
{ \sort_return_swapped: }
{ \sort_return_same: }
}
{ \bool_if:nTF
}
% 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
\tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
\
\{ }

\bool_set_true:N \l__zrefclever_sort_decided_bool
\int_compare:nNnTF
\{ \tl_head:N \l__zrefclever_label_enclval_a_tl \}
\tl_set:Nx \l__zrefclever_label_enclval_a_tl
\tl_set:Nx \l__zrefclever_label_enclval_b_tl
\int_compare:nNnTF
\{ \tl_head:N \l__zrefclever_label_enclval_a_tl \}
\int_zero:N \l__zrefclever_sort_prior_a_int
\int_zero:N \l__zrefclever_sort_prior_b_int
\seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
\End{\l__zrefclever_sort_default_same_type:nn}.

\__zrefclever_sort_default_different_types:nn \langle label a \rangle \langle label b \rangle
\cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn \langle label a \rangle \langle label b \rangle
\{ }

Retrieve sort priorities for \langle label a \rangle and \langle label b \rangle. \l__zrefclever_typesort_seq was stored in reverse sequence, and we compute the sort priorities in the negative range, so that we can implicitly rely on '0' being the “last value”.
\int_zero:N \l__zrefclever_sort_prior_a_int
\int_zero:N \l__zrefclever_sort_prior_b_int
\seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
\__zrefclever_sort_page:nn

The sorting function for sorting of defined labels for references to “page”. This function is expected to be called within the sorting loop of \__zrefclever_sort_labels: and receives the pair of labels being considered for a change of order or not. It should always...
“return” either \texttt{\textbackslash sort\_return\_same}: or \texttt{\textbackslash sort\_return\_swapped}:. Compared to the sorting of default labels, this is a piece of cake (thanks to \texttt{abspage}).

\texttt{\_\_zrefclever\_sort\_page:nn \{\texttt{\textcolor{red}{\textbackslash\textbackslash label\ a}}\} \{\texttt{\textcolor{red}{\textbackslash\textbackslash label\ b}}\}}

\texttt{\cs\_new\_protected:Npn \_\_zrefclever\_sort\_page:nn \#1\#2}
\texttt{\{ \texttt{\textcomp appell:nn \#1} \{ \texttt{\textcolor{red}{\textbackslash\textbackslash abspage}} \} \{ -1 \} \}}
\texttt{\texttt{\textcolor{red}{\textbackslash\textbackslash >}}}
\texttt{\{ \texttt{\_\_zrefclever\_extract\_nnn \#1} \{ \texttt{\textcolor{red}{\textbackslash\textbackslash abspage}} \} \{ -1 \} \}}
\texttt{\texttt{\textcolor{red}{\textbackslash\textbackslash >}}}
\texttt{\{ \texttt{\_\_zrefclever\_extract\_nnn \#2} \{ \texttt{\textcolor{red}{\textbackslash\textbackslash abspage}} \} \{ -1 \} \}}
\texttt{\texttt{\textcolor{red}{\textbackslash\textbackslash >}}}
\texttt{\{ \texttt{\_\_zrefclever\_extract\_nnn \#3} \}}
\texttt{\texttt{\textcolor{red}{\textbackslash\textbackslash >}}}
\texttt{\{ \texttt{\_\_zrefclever\_return\_swapped:} \}}
\texttt{\texttt{\textcolor{red}{\textbackslash\textbackslash >}}}
\texttt{\{ \texttt{\_\_zrefclever\_return\_same:} \}}
\texttt{\}}

(\textit{End definition for \_\_zrefclever\_sort\_page:nn}.)

\section{Typesetting}

“Typesetting” the reference, which here includes the parsing of the labels and eventual compression of labels in sequence into ranges, is definitely the “crux” of \texttt{zref-clever}. This because we process the label set as a stack, in a single pass, and hence “parsing”, “compressing”, and “typesetting” must be decided upon at the same time, making it difficult to slice the job into more specific and self-contained tasks. So, do bear this in mind before you curse me for the length of some of the functions below, or before a more orthodox “docstripper” complains about me not sticking to code commenting conventions to keep the code more readable in the \texttt{.dtx} file.

While processing the label stack (kept in \texttt{\_\_zrefclever\_typeset\_labels\_seq}), \texttt{\_\_zrefclever\_typeset\_refs}: “sees” two labels, and two labels only, the “current” one (kept in \texttt{\_\_zrefclever\_label\_a\_tl}), and the “next” one (kept in \texttt{\_\_zrefclever\_label\_b\_tl}). However, the typesetting needs (a lot) more information than just these two immediate labels to make a number of critical decisions. Some examples: i) We cannot know if labels “current” and “next” of the same type are a “pair”, or just “elements in a list”, until we examine the label after “next”; ii) If the “next” label is of the same type as the “current”, and it is in immediate sequence to it, it potentially forms a “range”, but we cannot know if “next” is actually the end of the range until we examined an arbitrary number of labels, and found one which is not in sequence from the previous one; iii) When processing a type block, the “name” comes first, however, we only know if that name should be plural, or if it should be included in the hyperlink, after processing an arbitrary number of labels and find one of a different type. One could naively assume that just examining “next” would be enough for this, since we can know if it is of the same type or not. Alas, “there be ranges”, and a compression operation may boil down to a single element, so we have to process the whole type block to know how its name should be typeset; iv) Similar issues apply to lists of type blocks, each of which is of arbitrary length: we can only know if two type blocks form a “pair” or are “elements in a list” when we finish the block. Etc. etc. etc.

We handle this by storing the reference “pieces” in “queues”, instead of typesetting them immediately upon processing. The “queues” get typeset at the point where all the information needed is available, which usually happens when a type block finishes (we see
something of a different type in “next”, signaled by \_\_zrefclever_last_of_type_bool, or the stack itself finishes (has no more elements, signaled by \_\_zrefclever_typeset_last_bool). And, in processing a type block, the type “name” gets added last (on the left) of the queue. The very first reference of its type always follows the name, since it may form a hyperlink with it (so we keep it stored separately, in \_\_zrefclever_type_first_label_tl, with \_\_zrefclever_type_first_label_type_tl being its type). And, since we may need up to two type blocks in storage before typesetting, we have two of these “queues”: \_\_zrefclever_typeset_queue_curr_tl and \_\_zrefclever_typeset_queue_prev_tl.

Some of the relevant cases (e.g., distinguishing “pair” from “list”) are handled by counters, the main ones are: one for the “type” (\_\_zrefclever_type_count_int) and one for the “label in the current type block” (\_\_zrefclever_label_count_int).

Range compression, in particular, relies heavily on counting to be able to distinguish relevant cases. \_\_zrefclever_range_count_int counts the number of elements in the current sequential “streak”, and \_\_zrefclever_range_same_count_int counts the number of equal elements in that same “streak”. The difference between the two allows us to distinguish the cases in which a range actually “skips” a number in the sequence, in which case we should use a range separator, from when they are after all just contiguous, in which case a pair separator is called for. Since, as usual, we can only know this when a arbitrary long “streak” finishes, we have to store the label which (potentially) begins a range (kept in \_\_zrefclever_range_beg_label_tl). \_\_zrefclever_next_maybe_range_bool signals when “next” is potentially a range with “current”, and \_\_zrefclever_next_is_same_bool when their values are actually equal.

One further thing to discuss here – to keep this “on record” – is inhibition of compression for individual labels. It is not difficult to handle it at the infrastructure side, what gets sloppy is the user facing syntax to signal such inhibition. For some possible alternatives for this, suggested by Enrico Gregorio, Phelype Oleinik, and Steven B. Segletes (and good ones at that) see https://tex.stackexchange.com/q/611370. Yet another alternative would be an option receiving the label(s) not to be compressed, this would be a repetition, but would keep the syntax clean. All in all, probably the best is simply not to allow individual inhibition of compression. We can already control compression of each \zcref call with existing options, this should be enough. I don’t think the small extra flexibility individual label control for this would grant is worth the syntax disruption it would entail. Anyway, it would be easy to deal with this in case the need arose, by just adding another condition (coming from whatever the chosen syntax was) when we check for \_\_zrefclever_labels_in_sequence:nn in \_\_zrefclever_typeset_refs_not_last_of_type:. But I remain unconvinced of the pertinence of doing so.

**Variables**

**Auxiliary variables for \_\_zrefclever_typeset_refs: main stack control.**

\_\_zrefclever_typeset_labels_seq\_\_zrefclever_typeset_last_bool\_\_zrefclever_last_of_type_bool

\_\_zrefclever_type_count_int\_\_zrefclever_label_count_int

(End definition for \_\_zrefclever_typeset_labels_seq, \_\_zrefclever_typeset_last_bool, and \_\_zrefclever_last_of_type_bool.)

**Auxiliary variables for \_\_zrefclever_typeset_refs: main counters.**

\_\_zrefclever_type_count_int\_\_zrefclever_label_count_int

60
\l__zrefclever_label_a_tl \l__zrefclever_label_b_tl
\l__zrefclever_typeset_queue_prev_tl \l__zrefclever_typeset_queue_curr_tl
\l__zrefclever_type_first_label_tl \l__zrefclever_type_first_label_type_tl

Auxiliary variables for \__zrefclever_typeset.refs: main “queue” control and storage.

2151 \tl_new:N \l__zrefclever_label_a_tl
2152 \tl_new:N \l__zrefclever_label_b_tl
2153 \tl_new:N \l__zrefclever_typeset_queue_prev_tl
2154 \tl_new:N \l__zrefclever_typeset_queue_curr_tl
2155 \tl_new:N \l__zrefclever_type_first_label_tl
2156 \tl_new:N \l__zrefclever_type_first_label_type_tl

\l__zrefclever_type_name_tl \l__zrefclever_name_in_link_bool
\l__zrefclever_name_format_tl \l__zrefclever_name_format_fallback_tl
\l__zrefclever_type_name_gender_tl

Auxiliary variables for \__zrefclever_typeset.refs: type name handling.

2157 \tl_new:N \l__zrefclever_type_name_tl
2158 \bool_new:N \l__zrefclever_name_in_link_bool
2159 \tl_new:N \l__zrefclever_name_format_tl
2160 \tl_new:N \l__zrefclever_name_format_fallback_tl
2161 \tl_new:N \l__zrefclever_type_name_gender_tl

\l__zrefclever_range_count_int \l__zrefclever_range_same_count_int
\l__zrefclever_range_beg_label_tl \l__zrefclever_next_maybe_range_bool
\l__zrefclever_next_is_same_bool

Auxiliary variables for \__zrefclever_typeset.refs: range handling.

2162 \int_new:N \l__zrefclever_range_count_int
2163 \int_new:N \l__zrefclever_range_same_count_int
2164 \tl_new:N \l__zrefclever_range_beg_label_tl
2165 \bool_new:N \l__zrefclever_next_maybe_range_bool
2166 \bool_new:N \l__zrefclever_next_is_same_bool

\l__zrefclever_tpairsep_tl \l__zrefclever_tlistsep_tl
\l__zrefclever_tlastsep_tl \l__zrefclever_namesep_tl
\l__zrefclever_pairsep_tl \l__zrefclever_listsep_tl
\l__zrefclever_lastsep_tl \l__zrefclever_rangesep_tl
\l__zrefclever_preref_tl \l__zrefclever_postref_tl
\l__zrefclever_namefont_tl \l__zrefclever_reffont_tl
\bool_new:N \l__zrefclever_capitalize_bool
\bool_new:N \l__zrefclever_abbrev_bool

Auxiliary variables for \__zrefclever_typeset.refs: separators, pre-/postref and font
and other options.

2167 \tl_new:N \l__zrefclever_tpairsep_tl
2168 \tl_new:N \l__zrefclever_tlistsep_tl
2169 \tl_new:N \l__zrefclever_tlastsep_tl
2170 \tl_new:N \l__zrefclever_namesep_tl
2171 \tl_new:N \l__zrefclever_pairsep_tl
2172 \tl_new:N \l__zrefclever_listsep_tl
2173 \tl_new:N \l__zrefclever_lastsep_tl
2174 \tl_new:N \l__zrefclever_rangesep_tl
2175 \tl_new:N \l__zrefclever_preref_tl
2176 \tl_new:N \l__zrefclever_postref_tl
2177 \tl_new:N \l__zrefclever_namefont_tl
2178 \tl_new:N \l__zrefclever_reffont_tl
2179 \bool_new:N \l__zrefclever_capitalize_bool
2180 \bool_new:N \l__zrefclever_abbrev_bool

\l__zrefclever_verbose_testing_bool

Internal variable which enables extra log messaging at points of interest in the code
for purposes of regression testing. Particularly relevant to keep track of expansion control in
\l__zrefclever_typeset_queue_curr_tl.

2181 \bool_new:N \l__zrefclever_verbose_testing_bool

(End definition for \l__zrefclever_verbose_testing_bool.)
Main functions

Main typesetting function for \zcref.

\cs_new_protected:Npn \_\_zrefclever_typeset_refs:
\{% \seq_set_eq:NN \l__zrefclever_typeset_labels_seq \l__zrefclever_zcref_labels_seq \tl_clear:N \l__zrefclever_typeset_queue_prev_tl \tl_clear:N \l__zrefclever_typeset_queue_curr_tl \tl_clear:N \l__zrefclever_type_first_label_tl \tl_clear:N \l__zrefclever_type_first_label_type_tl \tl_clear:N \l__zrefclever_range_beg_label_tl \int_zero:N \l__zrefclever_label_count_int \int_zero:N \l__zrefclever_type_count_int \int_zero:N \l__zrefclever_range_count_int \int_zero:N \l__zrefclever_range_same_count_int \% Get type block options (not type-specific).
\_\_zrefclever_get_ref_opt_typeset:nN \tpairsep \l__zrefclever_tpairsep_tl \_\_zrefclever_get_ref_opt_typeset:nN \tlistsep \l__zrefclever_tlistsep_tl \_\_zrefclever_get_ref_opt_typeset:nN \tlastsep \l__zrefclever_tlastsep_tl \% Process label stack.
\bool_set_false:N \_\_zrefclever_typeset_last_bool \bool_until_do:Nn \_\_zrefclever_typeset_last_bool \{ \seq_pop_left:NN \l__zrefclever_typeset_labels_seq \l__zrefclever_label_a_tl \seq_if_empty:NTF \_\_zrefclever_typeset_labels_seq \{ \tl_clear:N \l__zrefclever_label_b_tl \bool_set_true:N \_\_zrefclever_typeset_last_bool \} \seq_get_left:NN \l__zrefclever_typeset_labels_seq \_\_zrefclever_type_b_tl \} \\tl_if_eq:NnTF \_\_zrefclever_ref_property_tl \page \{ \} \\tl_set:Nn \_\_zrefclever_label_type_a_tl \page \\tl_set:Nn \_\_zrefclever_label_type_b_tl \page \} \_\_zrefclever_extract_default:NVnn \_\_zrefclever_label_type_a_tl \zc@type \zc@missingtype \_\_zrefclever_extract_default:NVnn \_\_zrefclever_label_type_b_tl \zc@type \zc@missingtype \_\_zrefclever_extract_default:NVnn
% First, we establish whether the "current label" (i.e. 'a') is the
% last one of its type. This can happen because the "next label"
% (i.e. 'b') is of a different type (or different definition status),
% or because we are at the end of the list.
\bool_if:NTF \l__zrefclever_typeset_last_bool
  { \bool_set_true:N \l__zrefclever_last_of_type_bool }
  { \zref@ifrefundefined { \l__zrefclever_label_a_tl }
    { \zref@ifrefundefined { \l__zrefclever_label_b_tl }
      { \bool_set_false:N \l__zrefclever_last_of_type_bool }
      { \bool_set_true:N \l__zrefclever_last_of_type_bool }
    }
    { \zref@ifrefundefined { \l__zrefclever_label_b_tl }
      { \zref@ifrefundefined { \l__zrefclever_label_a_tl }
        { \bool_set_true:N \l__zrefclever_last_of_type_bool }
      }
      { \zref@ifrefundefined { \l__zrefclever_label_a_tl }
        { \zref@ifrefundefined { \l__zrefclever_label_b_tl }
          { \bool_set_false:N \l__zrefclever_last_of_type_bool }
          { \bool_set_true:N \l__zrefclever_last_of_type_bool }
        }
        { \zref@ifrefundefined { \l__zrefclever_label_b_tl }
          { \bool_set_true:N \l__zrefclever_last_of_type_bool }
        }
      }
      { \zref@ifrefundefined { \l__zrefclever_label_a_tl }
        { \zref@ifrefundefined { \l__zrefclever_label_b_tl }
          { \bool_set_true:N \l__zrefclever_last_of_type_bool }
        }
        { \zref@ifrefundefined { \l__zrefclever_label_b_tl }
          { \bool_set_true:N \l__zrefclever_last_of_type_bool }
        }
      }
      { \bool_set_true:N \l__zrefclever_last_of_type_bool }
    }
    { \zref@ifrefundefined { \l__zrefclever_label_a_tl }
      { \zref@ifrefundefined { \l__zrefclever_label_b_tl }
        { \bool_set_true:N \l__zrefclever_last_of_type_bool }
      }
      { \zref@ifrefundefined { \l__zrefclever_label_b_tl }
        { \bool_set_true:N \l__zrefclever_last_of_type_bool }
      }
    }
    { \zref@ifrefundefined { \l__zrefclever_label_a_tl }
      { \zref@ifrefundefined { \l__zrefclever_label_b_tl }
        { \bool_set_true:N \l__zrefclever_last_of_type_bool }
      }
      { \zref@ifrefundefined { \l__zrefclever_label_b_tl }
        { \bool_set_true:N \l__zrefclever_last_of_type_bool }
      }
    }
    { \bool_set_true:N \l__zrefclever_last_of_type_bool }
  }

% Handle warnings in case of reference or type undefined.
% Test: 'zc-typeset01.lvt': "Typeset refs: warn ref undefined"
\zref@refused { \l__zrefclever_label_a_tl }
% Test: 'zc-typeset01.lvt': "Typeset refs: warn missing type"
\zref@ifrefundefined { \l__zrefclever_label_a_tl }
  { \int_compare:nNnT \l__zrefclever_label_count_int = { 0 } }
(End definition for \__zrefclever_typeset_refs:.)

This is actually the one meaningful “big branching” we can do while processing the label stack: i) the “current” label is the last of its type block; or ii) the “current” label is not the last of its type block. Indeed, as mentioned above, quite a number of things can only be decided when the type block ends, and we only know this when we look at the “next” label and find something of a different “type” (loose here, maybe different definition status, maybe end of stack). So, though this is not very strict, \__zrefclever_typeset_refs_last_of_type: is more of a “wrapping up” function, and it is indeed the one which does the actual typesetting, while \__zrefclever_typeset_refs_not_last_of_type: is more of an “accumulation” function.

\__zrefclever_typeset_refs_last_of_type: Handles typesetting when the current label is the last of its type.

\cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type: 
{
% Process the current label to the current queue.
\int_case:nnF { \l__zrefclever_label_count_int } 
{
% It is the last label of its type, but also the first one, and that’s % what matters here: just store it.
% Test: ‘zc-typeset01.lvt’: “Last of type: single”
\{ 0 \}
\{ \tl_set:NV \l__zrefclever_type_first_label_tl \l__zrefclever_label_a_tl \tl_set:NV \l__zrefclever_type_first_label_type_tl \l__zrefclever_label_type_a_tl \}

% The last is the second: we have a pair (if not repeated).
% Test: 'zc-typeset01.lvt': "Last of type: pair"
\{ 1 \}
\{ \int_compare:nNnF \{ \l__zrefclever_range_same_count_int \} = \{ 1 \} \{
\tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl \{ \exp_not:V \l__zrefclever_pairsep_tl \l__zrefclever_get_ref:V \l__zrefclever_label_a_tl \}
\} \}

% Last is third or more of its type: without repetition, we'd have the
% last element on a list, but control for possible repetition.
\{ \int_case:nFF \{ \l__zrefclever_range_count_int \} \{
% There was no range going on.
% Test: 'zc-typeset01.lvt': "Last of type: not range"
\{ 0 \}
\{ \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl \{ \exp_not:V \l__zrefclever_lastsep_tl \l__zrefclever_get_ref:V \l__zrefclever_label_a_tl \}
\} \}

% Last in the range is also the second in it.
% Test: 'zc-typeset01.lvt': "Last of type: pair in sequence"
\{ 1 \}
\{ \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl \{ \exp_not:V \l__zrefclever_listsep_tl \l__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl \int_compare:nFF \{ \l__zrefclever_range_same_count_int \} = \{ 1 \} \{
\exp_not:V \l__zrefclever_lastsep_tl \l__zrefclever_get_ref:V \l__zrefclever_label_a_tl \}
\} \}

65
% Last in the range is third or more in it.
\int_case:nnF
{ \l__zrefclever_range_count_int - \l__zrefclever_range_same_count_int }
%
% Repetition, not a range.
% Test: 'zc-typeset01.lvt': "Last of type: range to one"
{ 0 }
%
{ % If 'range_beg_label' is empty, it means it was also the
% first of the type, and hence was already handled.
\tl_if_empty:VF \l__zrefclever_range_beg_label_tl
{ \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
  { \exp_not:V \l__zrefclever_lastsep_tl
    \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
  }
}
%
% A 'range', but with no skipped value, treat as list.
% Test: 'zc-typeset01.lvt': "Last of type: range to pair"
{ 1 }
%
{ \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
  { % Ditto.
    \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
    { \exp_not:V \l__zrefclever_lastsep_tl
      \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
    }
    \exp_not:V \l__zrefclever_lastsep_tl
    \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
  }
}
%
% An actual range.
% Test: 'zc-typeset01.lvt': "Last of type: range"
\tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
{ % Ditto.
  \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
  { \exp_not:V \l__zrefclever_lastsep_tl
  }
}
)

66
% Handle "range" option. The idea is simple: if the queue is not empty,
% we replace it with the end of the range (or pair). We can still
% retrieve the end of the range from 'label_a' since we know to be
% processing the last label of its type at this point.
\bool_if:NT \l__zrefclever_typeset_range_bool
{\tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
 {\zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
 { } 
 { \msg_warning:nxn { zref-clever } { single-element-range }
 { \l__zrefclever_type_first_label_type_tl }
 } 
 }
} {\bool_set_false:N \l__zrefclever_next_maybe_range_bool
 \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
 { } 
 { \__zrefclever_labels_in_sequence:nn 
 { \l__zrefclever_type_first_label_tl }
 { \l__zrefclever_label_a_tl }
 } 
}% Test: 'zc-typeset01.lvt': "Last of type: option range"
% Test: 'zc-typeset01.lvt': "Last of type: option range to pair"
\tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
 { \bool_if:NTF \l__zrefclever_typeset_range_bool
 { \exp_not:V \l__zrefclever_pairsep_tl }
 { \exp_not:V \l__zrefclever_rangesep_tl }
 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
 } 
}% Now that the type block is finished, we can add the name and the first
% ref to the queue. Also, if "typeset" option is not "both", handle it
% here as well.
\__zrefclever_type_name_setup:
\bool_if:nTF
{ \l__zrefclever_typeset_ref_bool \&\& \l__zrefclever_typeset_name_bool }
{ \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
67
\bool_if:NTF \l__zrefclever_typeset_ref_bool \{
\tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
\}
\bool_if:NTF \l__zrefclever_typeset_name_bool \{
\tl_set:Nx \l__zrefclever_typeset_queue_curr_tl { \bool_if:NTF \l__zrefclever_name_in_link_bool \{
\exp_not:N \group_begin:
\exp_not:V \l__zrefclever_namefont_tl
% It's two '@s', but escaped for DocStrip.
\exp_not:N \hyper@@link
\__zrefclever_extract_url_unexp:V \l__zrefclever_type_first_label_tl
\}
\}
\exp_not:N \group_end:
\exp_not:N \group_end:
\}
\bool_if:NTF \l__zrefclever_typeset_none_bool \{
\tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl { \__zrefclever_get_ref_first: }
\}
\}
% Logically, this case would correspond to "typeset=none", but
% it should not occur, given that the options are set up to
% typeset either "ref" or "name". Still, leave here a
% sensible fallback, equal to the behavior of "both".
% Test: 'zc-typeset01.lvt': "Last of type: option typeset none"
\tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl \{ \__zrefclever_get_ref_first: \}
\}
% Typeset the previous type block, if there is one.
\int_compare:nNnT \l__zrefclever_type_count_int > 0 {
  \int_compare:nNnT \l__zrefclever_type_count_int > 1 {
    \l__zrefclever_tlistsep_tl
    \l__zrefclever_typeset_queue_prev_tl
  }
}

% Extra log for testing.
\bool_if:NT \l__zrefclever_verbose_testing_bool {
  \tl_show:N \l__zrefclever_typeset_queue_currl_tl
}

% Wrap up loop, or prepare for next iteration.
\bool_if:NTF \l__zrefclever_typeset_last_bool {
  \int_case:nnF \l__zrefclever_type_count_int {
    0 {
      \l__zrefclever_typeset_queue_currl_tl
    }
    1 {
      \l__zrefclever_tpairsep_tl
      \l__zrefclever_typeset_queue_currl_tl
    }
    \l__zrefclever_tlastsep_tl
    \l__zrefclever_typeset_queue_currl_tl
  }
  \bool_lazy_all:nT {
    \l__zrefclever_nudge_enabled_bool
    \l__zrefclever_nudge_multitype_bool
    \int_compare_p:nNn \l__zrefclever_type_count_int > 0
  }
  \msg_warning:nn { zref-clever } { nudge-multitype }
}

% There are further labels, set variables for next iteration.
\tl_eq:N \l__zrefclever_typeset_queue_prev_tl
\l__zrefclever_typeset_queue_currl_tl
\l__zrefclever_range_beg_label_tl
\int_zero:N \l__zrefclever_label_count_int
\int_incr:N \l__zrefclever_type_count_int

69
\int_zero:N \l__zrefclever_range_count_int
\int_zero:N \l__zrefclever_range_same_count_int
}

(End definition for \__zrefclever_typeset_refs_last_of_type:)

\__zrefclever_typesetRefs_not_last_of_type:
Handles typesetting when the current label is not the last of its type.
\cs_new_protected:Npn \__zrefclever_typesetRefs_not_last_of_type:
{
  \% Signal if next label may form a range with the current one (only
  \% considered if compression is enabled in the first place).
  \bool_set_false:N \l__zrefclever_next_maybe_range_bool
  \bool_set_false:N \l__zrefclever_next_is_same_bool
  \bool_if:NT \l__zrefclever_typeset_compress_bool
  {
    \zref@ifrefundefined { \l__zrefclever_label_a_tl }
    { }
    { \__zrefclever_labels_in_sequence:nn
      { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl } }
  }
  \% Process the current label to the current queue.
  \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
  {
    \% Current label is the first of its type (also not the last, but it
    \% doesn’t matter here): just store the label.
    \tl_set:NV \l__zrefclever_type_first_label_tl \l__zrefclever_label_a_tl
    \tl_set:NV \l__zrefclever_type_first_label_type_tl \l__zrefclever_label_type_a_tl
    \% If the next label may be part of a range, we set ‘range_beg_label’
    \% to "empty" (we deal with it as the "first", and must do it there, to
    \% handle hyperlinking), but also step the range counters.
    \% Test: ‘zc-typeset01.lvt’: "Not last of type: first is range"
    \bool_if:NT \l__zrefclever_typeset_compress_bool
    {
      \tl_clear:N \l__zrefclever_range_beg_label_tl
      \int_incr:N \l__zrefclever_range_count_int
      \bool_if:NT \l__zrefclever_next_is_same_bool
      { \int_incr:N \l__zrefclever_range_same_count_int }
    }
  }
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
  \%
\% There was no range going, we are starting one.
\l_set:NV \l__zrefclever_range_beg_label_tl
  \l__zrefclever_label_a_tl
\int_incr:N \l__zrefclever_range_count_int
\bool_if:NT \l__zrefclever_next_is_same_bool
  { \int_incr:N \l__zrefclever_range_same_count_int }
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
{  
  % Last is third or more in the range: if 'range_count' and 
  % 'range_same_count' are the same, its a repetition (drop it), 
  % if they differ by '1', its a list, if they differ by more, 
  % it is a real range. 
  \int_case:nnF  
  {  
    \l__zrefclever_range_count_int -  
    \l__zrefclever_range_same_count_int  
  }  
  {  
    % Test: 'zc-typeset01.lvt': "Not last of type: range to one"  
    { 0 }  
    {  
      \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl  
      {  
        \tl_if_empty:VF \l__zrefclever_range_beg_label_tl  
        {  
          \exp_not:V \l__zrefclever_listsep_tl  
          \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl  
        }  
      }  
    }  
    {  
      % Test: 'zc-typeset01.lvt': "Not last of type: range to pair"  
      { 1 }  
      {  
        \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl  
        {  
          \tl_if_empty:VF \l__zrefclever_range_beg_label_tl  
          {  
            \exp_not:V \l__zrefclever_listsep_tl  
            \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl  
          }  
          \exp_not:V \l__zrefclever_rangesep_tl  
          \__zrefclever_get_ref:V \l__zrefclever_label_a_tl  
        }  
      }  
    }  
  }  
  {  
    % Test: 'zc-typeset01.lvt': "Not last of type: range"  
    \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl  
    {  
      \tl_if_empty:VF \l__zrefclever_range_beg_label_tl  
      {  
        \exp_not:V \l__zrefclever_listsep_tl  
        \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl  
      }  
      \exp_not:V \l__zrefclever_rangesep_tl  
      \__zrefclever_get_ref:V \l__zrefclever_label_a_tl  
    }  
  }  
}  

72
\__zrefclever_range_count_int \int_zero:N \l__zrefclever_range_count_int
\int_zero:N \l__zrefclever_range_name_count_int
\int_incr:N \l__zrefclever_label_count_int
\int_zero:N \l__zrefclever_range_count_int
\int_zero:N \l__zrefclever_range_name_count_int
\int_incr:N \l__zrefclever_label_count_int
\int_incr:N \l__zrefclever_label_count_int
\int_incr:N \l__zrefclever_label_count_int
\int_zero:N \l__zrefclever_range_count_int
\int_zero:N \l__zrefclever_range_name_count_int
\int_incr:N \l__zrefclever_label_count_int

(End definition for \__zrefclever_typeset_refs_not_last_of_type:)

Auxiliary functions

\__zrefclever_get_ref:n and \__zrefclever_get_ref_first: are the two functions which actually build the reference blocks for typesetting. \__zrefclever_get_ref:n handles all references but the first of its type, and \__zrefclever_get_ref_first: deals with the first reference of a type. Saying they do “typesetting” is imprecise though, they actually prepare material to be accumulated in \l__zrefclever_typeset_queue-_curr_tl inside \__zrefclever_typeset_refs_last_of_type: and \__zrefclever_typeset_refs_not_last_of_type:. And this difference results quite crucial for the \TeXnical requirements of these functions. This because, as we are processing the label stack and accumulating content in the queue, we are using a number of variables which are transient to the current label, the label properties among them, but not only. Hence, these variables must be expanded to their current values to be stored in the queue. Indeed, \__zrefclever_get_ref:n and \__zrefclever_get_ref_first: get called, as they must, in the context of x type expansions. But we don’t want to expand the values of the variables themselves, so we need to get current values, but stop expansion after that. In particular, reference options given by the user should reach the stream for its final typesetting (when the queue itself gets typeset) unmodified (“no manipulation”, to use the \signature jargon). We also need to prevent premature expansion of material that can’t be expanded at this point (e.g. grouping, \zref@default or \hyper@@link).

In a nutshell, the job of these two functions is putting the pieces in place, but with proper expansion control.

\__zrefclever_ref_default: \__zrefclever_name_default: Default values for undefined references and undefined type names, respectively. We are ultimately using \zref@default, but calls to it should be made through these internal functions, according to the case. As a bonus, we don’t need to protect them with \exp:_not:N, as \zref@default would require, since we already define them protected.

(End definition for \__zrefclever_ref_default: and \__zrefclever_name_default:)

\__zrefclever_get_ref:n Handles a complete reference block to be accumulated in the “queue”, including “pre” and “pos” elements, and hyperlinking. For use with all labels, except the first of its type, which is done by \__zrefclever_get_ref_first:.

\__zrefclever_get_ref:n {{\label}}
\cs_new:Npn \_zrefclever_get_ref:n #1
\begin{verbatim}
\{ \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
\{ \bool_if:nTF
\{ \l__zrefclever_use_hyperref_bool &&
\{ \l__zrefclever_link_star_bool
\}
\{ \bool_if:NT \l__zrefclever_preposinlink_bool
\{ \exp_not:V \l__zrefclever_preref_tl
\% It’s two ‘@s’, but escaped for DocStrip.
\exp_not:N \hyper@@link
\{ \__zrefclever_extract_url_unexp:n {#1} \}
\{ \__zrefclever_extract_unexp:nnn {#1} { anchor } { } \}
\}
\{ \bool_if:NT \l__zrefclever_preposinlink_bool
\{ \exp_not:V \l__zrefclever_preref_tl
\exp_not:N \group_begin:
\exp_not:V \l__zrefclever_reffont_tl
\_zrefclever_extract_unexp:nv {#1}
\{ \l__zrefclever_ref_property_tl \}
\exp_not:N \group_end:
\bool_if:NT \l__zrefclever_preposinlink_bool
\{ \exp_not:V \l__zrefclever_postref_tl \}
\}
\bool_if:NT \l__zrefclever_preposinlink_bool
\{ \exp_not:V \l__zrefclever_postref_tl \}
\}
\{ \exp_not:V \l__zrefclever_preref_tl
\exp_not:N \group_begin:
\exp_not:V \l__zrefclever_reffont_tl
\_zrefclever_extract_unexp:nv {#1}
\{ \l__zrefclever_ref_property_tl \}
\exp_not:N \group_end:
\exp_not:V \l__zrefclever_postref_tl
\}
\}
\} \cs_generate_variant:Nn \_zrefclever_get_ref:n { V }
\end{verbatim}
\end{quote}

\_zrefclever_get_ref_first: Handles a complete reference block for the first label of its type to be accumulated in the “queue”, including “pre” and “pos” elements, hyperlinking, and the reference type “name”. It does not receive arguments, but relies on being called in the appropriate place in \_zrefclever_typeset.refs_last_of_type: where a number of variables are expected to be appropriately set for it to consume. Prominently among those is \l__zrefclever_type_first_label_tl, but it also expected to be called right after \_zrefclever_type_name_setup: which sets \l__zrefclever_type_name_tl and \l__zrefclever_name_in_link_bool which it uses.
\begin{verbatim}
\bool_if:nTF \l__zrefclever_use_hyperref_bool 
{ \l__zrefclever_type_first_label_tl }
\bool_if:NT \l__zrefclever_preposinlink_bool
{ \_zrefclever_extract_url_unexp:V \l__zrefclever_type_first_label_tl }
\bool_if:NF \l__zrefclever_preposinlink_bool
{ \exp_not:V \l__zrefclever_preref_tl }
\bool_if:NF \l__zrefclever_preposinlink_bool
{ \exp_not:N \group_begin:
    \exp_not:V \l__zrefclever_namefont_tl
    \exp_not:V \l__zrefclever_type_name_tl
    \exp_not:N \group_end:
    \exp_not:V \l__zrefclever_namesep_tl
}
\zref@ifrefcontainsprop
{ \l__zrefclever_type_first_label_tl }
{ \l__zrefclever_ref_property_tl }
{ 
\bool_if:nTF
{ \l__zrefclever_use_hyperref_bool && 
  ! \l__zrefclever_link_star_bool}
\bool_if:NT \l__zrefclever_preposinlink_bool
{ \exp_not:V \l__zrefclever_preref_tl }
% It's two '@s', but escaped for DocStrip.
\exp_not:N \hyper@@link
{ \__zrefclever_extract_url_unexp:V 
  \l__zrefclever_type_first_label_tl }
{ \__zrefclever_extract_unexp:Vnn 
  \l__zrefclever_type_first_label_tl { anchor } { } }
{ \bool_if:NT \l__zrefclever_preposinlink_bool
  { \exp_not:V \l__zrefclever_preref_tl }
  \exp_not:N \group_begin:
  \exp_not:V \l__zrefclever_reffont_tl
  \_zrefclever_extract_unexp:Vnn
  \l__zrefclever_type_first_label_tl 
  \exp_not:V \l__zrefclever_ref_property_tl
  { } 
  \exp_not:N \group_end:
  \bool_if:NF \l__zrefclever_preposinlink_bool
  { \exp_not:N \group_begin:
    \exp_not:V \l__zrefclever_preref_tl }
  \bool_if:NF \l__zrefclever_preposinlink_bool
  { \exp_not:N \group_begin:
    \exp_not:V \l__zrefclever_postref_tl }
\end{verbatim}
\__zrefclever_type_name_setup: Auxiliary function to \__zrefclever_typeset.refs.last_of_type:. It is responsible for setting the type name variable $\l__zrefclever_type_name_tl$ and $\l__zrefclever_name_in_link_bool$. If a type name can’t be found, $\l__zrefclever_type_name_tl$ is cleared. The function takes no arguments, but is expected to be called in \__zrefclever_typeset.refs.last_of_type:, right before \__zrefclever_get.ref.first:, which is the main consumer of the variables it sets, though not the only one (and hence this cannot be moved into \__zrefclever_get.ref.first: itself). It also expects a number of relevant variables to have been appropriately set, and which it uses, prominently $\l__zrefclever_type_first_label_type_tl$, but also the queue itself in \__zrefclever_typeset.refs.queue.curr_tl, which should be “ready except for the first label”, and the type counter $\l__zrefclever_type_count_int$.

\cs_new_protected:Npn \__zrefclever_type_name_setup:
\begin{Verbatim}
\begin{verbatim}
{ \l__zrefclever_capitalize_bool 
\end{Verbatim}
\begin{Verbatim}
\begin{verbatim}
\l__zrefclever_capitalize_first_bool \l__zrefclever_type_count_int = 0 
\l__zrefclever_type_first_label_type_tl \l__zrefclever_type_name_tl 
\l__zrefclever_type_name_tl
\l__zrefclever_type_name_format_tl
\l__zrefclever_type_name_in_link_bool
\l__zrefclever_type_name_setup
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\begin{Verbatim}
{ \l__zrefclever_capitalize_first_bool \l__zrefclever_type_count_int = 0 
\l__zrefclever_type_name_tl 
\l__zrefclever_type_name_format_tl
\l__zrefclever_type_name_setup
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}

(End definition for \__zrefclever_get.ref.first:)

---

77
\tl_put_right:Nn \l__zrefclever_name_format_tl \{-ab\}
{ \tl_clear:N \l__zrefclever_name_format_fallback_tl }

% Handle number and gender nudges.
\bool_if:NT \l__zrefclever_nudge_enabled_bool
{ \bool_if:NTF \l__zrefclever_nudge_singular_bool
{ \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
{ \msg_warning:nnx { zref-clever }
{ nudge-plural-when-sg }
{ \l__zrefclever_type_first_label_type_tl }
}
}

{ \bool_lazy_all:nT
{ \bool_if_empty:N \l__zrefclever_typeset_queue_curr_tl
{ \int_compare_p:nNn
{ \l__zrefclever_label_count_int } > { 0 }
}
}
{ \msg_warning:nnx { zref-clever }
{ nudge-comptosing }
{ \l__zrefclever_type_first_label_type_tl }
}
}
}
\bool_lazy_and:nnT
{ \l__zrefclever_nudge_gender_bool }
{ ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl
{ \__zrefclever_get_lang_opt_type:xxNFP
{ \l__zrefclever_ref_language_tl }
{ \l__zrefclever_type_first_label_type_tl }
{ gender }
{ \l__zrefclever_type_name_gender_tl }
\clist_set:NV \l_tmpa_clist
{ \l__zrefclever_type_name_gender_tl }
\clist_if_in:NVF \l_tmpa_clist
\l__zrefclever_ref_gender_tl
{ \tl_if_empty:NTF \l__zrefclever_type_name_gender_tl
{ \msg_warning:nnxxx { zref-clever }
{ nudge-gender-not-declared-for-type }
{ \l__zrefclever_ref_gender_tl }
{ \l__zrefclever_type_first_label_type_tl }
}
\l__zrefclever_type
\l__zrefclever_type_first_label_type_tl _options_prop
\l__zrefclever_name_format_fallback_tl
\l__zrefclever_type_name_tl
{\tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
{\tl_put_left:Nn\l__zrefclever_name_format_tl { - }
\tl_put_left:NV\l__zrefclever_name_format_tl\l__zrefclever_ref_decl_case_tl
\tl_put_left:Nn\l__zrefclever_name_format_fallback_tl { - }
\tl_put_left:NV\l__zrefclever_name_format_fallback_tl\l__zrefclever_ref_decl_case_tl
}\l__zrefclever_get_lang_opt_type:xxxNF
{\l__zrefclever_ref_language_tl}
{\l__zrefclever_type_first_label_type_tl}
{\l__zrefclever_name_format_tl}
\l__zrefclever_type_name_tl
{\l__zrefclever_get_lang_opt_type:xxxNF
{\l__zrefclever_ref_language_tl}
{\l__zrefclever_type_first_label_type_tl}
{\l__zrefclever_name_format_fallback_tl}
\l__zrefclever_type_name_tl
{\tl_clear:N\l__zrefclever_type_name_tl
\msg_warning:nnxx { zref-clever }
{ missing-name }
{\l__zrefclever_name_format_tl}
{\l__zrefclever_type_first_label_type_tl}
}
}
}
}
% Signal whether the type name is to be included in the hyperlink or not.
\bool_lazy_any:nTF
{ ! \l__zrefclever_use_hyperref_bool }
{ \l__zrefclever_link_star_bool }
{ \tl_if_empty:p:N \l__zrefclever_type_name_tl }
{ \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
{ \bool_set_false:N \l__zrefclever_name_in_link_bool }
{ \bool_lazy_any:nTF
\_\_zrefclever\_\_type\_name\_setup:

A convenience auxiliary function for extraction of the `url` / `urluse` property, provided by the `zref-xr` module. Ensure that, in the context of an \texttt{xexp} expansion, \texttt{\zref@extractdefault} is expanded exactly twice, but no further to retrieve the proper value. See documentation for \_\_zrefclever\_\_extract\_\_unexp:nnn.

\texttt{\textbackslash cs\_new:Npn \_\_zrefclever\_\_extract\_\_url\_\_unexp:nn \#1 \{ \zref@ifpropundefined { urluse } { \_\_zrefclever\_\_extract\_\_unexp:nnn \#1 \{ url \} \{ \cempty \} } \zref@ifrefcontainsprop \#1 \{ urluse \} { \_\_zrefclever\_\_extract\_\_unexp:nnn \#1 \{ urluse \} \{ \cempty \} } \zref@extractdefault:NNnn \_\_zrefclever\_\_label\_extdoc\_a_tl \#1 \{ externaldocument \} \{ \cempty \} \zref@extractdefault:NNnn \_\_zrefclever\_\_label\_extdoc\_b_tl \#2 \{ externaldocument \} \{ \cempty \} }

(End definition for \_\_zrefclever\_\_type\_name\_setup:.)

\_\_zrefclever\_\_extract\_\_url\_\_unexp:n

Auxiliary function to \_\_zrefclever\_\_typeset\_\_refs\_not\_\_last\_\_of\_\_type:. Sets \_\_zrefclever\_\_next\_\_maybe\_\_range\_\_bool to true if \texttt{\langle label b \rangle} comes in immediate sequence from \texttt{\langle label a \rangle}. And sets both \_\_zrefclever\_\_next\_\_maybe\_\_range\_\_bool and \_\_zrefclever\_\_next\_\_is\_\_same\_\_bool to true if the two labels are the “same” (that is, have the same counter value). These two boolean variables are the basis for all range and compression handling inside \_\_zrefclever\_\_typeset\_\_refs\_not\_\_last\_\_of\_\_type:, so this function is expected to be called at its beginning, if compression is enabled.

\_\_zrefclever\_\_labels\_in\_\_sequence:nn

\texttt{\textbackslash cs\_new:NNpn \_\_zrefclever\_\_labels\_in\_\_sequence:nn \{\langle label a\rangle\} \{\langle label b\rangle\} }

(End definition for \_\_zrefclever\_\_extract\_\_url\_\_unexp:n.)
Finally, some functions for retrieving reference options values, according to the relevant precedence rules. They both receive an \(\langle\text{option}\rangle\) as argument, and store the retrieved value in an appropriate. Though these are mostly general functions (for a change...), they are not completely so, they rely on the current state of \(\l__zrefclever_label_type_a_tl\), as set during the processing of the label stack. This could be easily generalized, of course, but I don’t think it is worth it, \(\l__zrefclever_label_type_a_tl\) is indeed what we want in all practical cases. The difference between each of these functions is the kind of option each should be used for. \(\_\_zrefclever_get_ref_opt_typeset:nN\) is meant for the general options, and attempts to find values for them in all precedence levels (four plus “fallback”). \(\_\_zrefclever_get_ref_opt_font:nN\) is intended for “font” options, which cannot be “language-specific”, thus for these we just search general options and type options. \(\_\_zrefclever_get_ref_opt_bool:nN\) is intended for boolean options.

\(\_\_zrefclever_get_ref_opt_typeset:nN\)

\(\_\_zrefclever_get_ref_opt_typeset:nN \langle\text{option}\rangle\) \{(\text{tl variable})\}

\cs_new_protected:Npn \_\_zrefclever_get_ref_opt_typeset:nN #1#2

{ % First attempt: general options.
\prop_get:NnNF \l__zrefclever_ref_options_prop \#1 \#2
{ % If not found, try type specific options.
\bool_lazy_and:nnTF
\prop_if_exist_p:c
{ \l__zrefclever_type_
\l__zrefclever_label_type_a_tl _options_prop
}
\prop_if_in_p:cn
{ \l__zrefclever_type_
\l__zrefclever_label_type_a_tl _options_prop
}
{%startverbatim%
{#1}
}
{\prop_get:cnN
 { l__zrefclever_type_
 \l__zrefclever_label_type_a_tl _options_prop
 } #2
}
%
% If not found, try type- and language-specific.
\__zrefclever_get_lang_opt_type:xxnNF
 { \l__zrefclever_ref_language_tl }
 { \l__zrefclever_label_type_a_tl }
 {#1} #2
%
%
% If not found, try language-specific default.
\__zrefclever_get_lang_opt_default:xnNF
 { \l__zrefclever_ref_language_tl }
 {#1} #2
%
%
% If not found, try fallback.
\__zrefclever_get_fallback_unknown_lang_opt:nNF {#1} #2
{ \tl_clear:N #2
 \msg_warning:nnn \{ zref-clever \}
 { missing-string } {#1}
 }
%
(End definition for \__zrefclever_get_ref_opt_typeset:nN.)
\__zrefclever_get_ref_opt_font:nN
\__zrefclever_get_ref_opt_font:nN {⟨ option ⟩} {⟨ tl variable ⟩}
\cs_new_protected:Npn \__zrefclever_get_ref_opt_font:nN \__zrefclever_get_ref_opt_font:nN #1#2
{%
% First attempt: general options.
\prop_get:NnNF \__zrefclever_get_options_prop {#1} #2
%
% If not found, try type specific options.
\bool_if:nTF
{ \prop_if_exist_p:c
 { l__zrefclever_type_
 \l__zrefclever_label_type_a_tl _options_prop
 }
} #2
%
\prop_get:cnNF
{%endverbatim%}
\__zrefclever_get_ref_opt_bool:nN \__zrefclever_get_ref_opt_font:nN
\__zrefclever_get_ref_opt_bool:nnN \__zrefclever_get_ref_opt_bool:nN {
  ⟨option⟩
  ⟨default⟩
  ⟨bool variable⟩
}\cs_new_protected:Npn \__zrefclever_get_ref_opt_bool:nnN #1#2#3
\__zrefclever_get_ref_opt_font:nN (=End definition for \__zrefclever_get_ref_opt_font:nN.)
% If not found, use default argument.
\bool_lazy_or:nnTF
{ \str_if_eq_p:nn {#2} { true } }
{ \str_if_eq_p:nn {#2} { false } }
{ \tl_set:Nn \l_tmpa_tl {#2} }
{
    % And, if even that fails, presume false.
    \tl_set:Nn \l_tmpa_tl { false }
}
%

Having retrieved the option value, set the boolean. At this point, we
% *know* \l_tmpa_tl is either 'true' or 'false'.
\use:c { bool_set_\l_tmpa_tl:N } #3
%
(End definition for \_zrefclever_get_ref_opt_bool:nnN.)

9 Compatibility

This section is meant to aggregate any “special handling” needed for \TeX kernel features, document classes, and packages, needed for zref-clever to work properly with them.

9.1 appendix

One relevant case of different reference types sharing 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.

The standard \appendix command is a one way switch, in other words, it cannot be reverted (see https://tex.stackexchange.com/a/444057). So, even if the fact that it is a “switch” rather than an environment complicates things, because we have to make ungrouped settings to correspond to its effects, in practice this is not a big deal, since these settings are never really reverted (by default, at least). Hence, hooking into \appendix is a viable and natural alternative. The memoir class and the appendix package define the appendices and subappendices environments, which provide for a way for the appendix to “end”, but in this case, of course, we can hook into the environment instead.

\_zrefclever_compat_module:nn { appendix }
{
\AddToHook { cmd / appendix / before }
{
  \_zrefclever_zcsetup:n

86
Depending on the definition of \appendix, using the hook may lead to trouble with the first released version of \ltcmdhooks (the one released with the 2021-06-01 kernel). Particularly, if the definition of the command being hooked at contains a double hash mark (##) the patch to add the hook, if it needs to be done with the \santokens method, may fail noisily (see https://tex.stackexchange.com/q/617905, with a detailed explanation and possible workaround by Phelype Oleinik). The 2021-11-15 kernel release already handles this gracefully, thanks to fix by Phelype Oleinik at https://github.com/latex3/latex2e/pull/699.

9.2 appendices

This module applies both to the appendix package, and to the memoir class, since it "emulates" the package.

\__zrefclever_compat_module:nn { appendices }
\__zrefclever_if_package_loaded:nT { appendix }

\__zrefclever_zcsetup:n
\cs_if_exist:cTF { chapter }

\__zrefclever_zcsetup:n
\cs_if_exist:cTF { section }

\__AddToHook { env / appendices / begin }
\stepcounter { zc@save@appendix }
\setcounter { zc@appendix } { \value { zc@save@appendix } }
\__zrefclever_zcsetup:n
memoir

The memoir document class has quite a number of cross-referencing related features, mostly dealing with captions, subfloats, and notes. Some of them are implemented in ways which make difficult the use of zref, particularly \zlabel, short of redefining the whole stuff ourselves. Hopefully, these features are specialized enough to make zref-clever useful enough with memoir without much friction, but unless some support is added upstream, it is difficult not to be a little intrusive here.

1. Caption functionality which receives ⟨label⟩ as optional argument, namely:

(a) The sidecaption and sidecontcaption environments. These environments store the label in an internal macro, \m@mscaplabel, at the begin environment code (more precisely in \@@sidecaption), but both the call to \refstepcounter and the expansion of \m@mscaplabel take place at \endsidecaption. For this reason, hooks are not particularly helpful, and there is not any easy way to grab the ⟨label⟩ argument to start with. I can see
two ways to deal with these environments, none of which I really like. First, map through \m@mscaplabel until \label is found, then grab the next token which is the \langle label \rangle. This can be used to set a \zlabel either with a kernel environment hook, or with \@mem@scap@afterhook (the former requires running \refstepcounter on our own, since the env/.../end hook comes before this is done by \endsidecaption). Second, locally redefine \label to set both labels inside the environments.

(b) The bilingual caption commands: \bitwonumcaption, \bionenumcaption, and \bicaption. These commands do not support setting the label in their arguments (the labels do get set, but they end up included in the title property of the label too). So we do the same for them as for \sidecaption, just taking care of grouping, since we can’t count on the convenience of the environment hook (luckily for us, they are scoped themselves, so we can add an extra group there).

2. The \subcaptionref command, which makes a reference to the subcaption without the number of the main caption (e.g. “(b)”, instead of “2.3(b)”), for labels set inside the \langle subtitle \rangle argument of the subcaptioning commands, namely: \subcaption, \contsubcaption, \subbottom, \contsubbottom, \subtop. This functionality is implemented by \memor by setting a second label with prefix sub\langle label \rangle, and storing there just that part of interest. With zref this part is easier, since we can just add an extra property and retrieve it later on. The thing is that it is hard to find a place to hook into to add the property to the main list, since \memor does not really consider the possibility of some other command setting labels. \@mmsubcaption is the best place to hook I could find. It is used by subcaptioning commands, and only those. And there is no hope for an environment hook in this case anyway.

3. \memor’s \footnote, \verbfootnote, \sidefootnote and \pagenote, just as the regular \footnote until recently in the kernel, do not set \@currentcounter alongside \@currentlabel, proper referencing to them requires setting the type for it.

4. Note that \memor’s appendix features “emulates” the appendix package, hence the corresponding compatibility module is loaded for \memor even if that package is not itself loaded. The same is true for the \appendix command module, since it is also defined.

\__zrefclever_compat_module:nn { \memor }
\{ 
\__zrefclever_if_class_loaded:nT { \memor } 
\} 
Add subfigure and subtable support out of the box. Technically, this is not “default” behavior for \memor, users have to enable it with \newsubfloat, but let this be smooth. Still, this does not cover any other floats created with \newfloat. Also include setup for \verse.

\__zrefclever_zcsetup:n 
\{ 
countertype = 
\} 
\subfigure = figure 
\subtable = table 
\poemline = line ,

89
Support for caption memoir features that require that \langle label \rangle be supplied as an optional argument.
\\cs_new_protected:Npn \__zrefclever_memoir_both_labels: 
\{ 
\cs_set_eq:NN \__zrefclever_memoir_orig_label:n \label
\cs_set:Npn \__zrefclever_memoir_label_and_zlabel:n \{##1
\__zrefclever_memoir_orig_label:n {##1}
\zlabel{##1}
\}
\cs_set_eq:NN \label \__zrefclever_memoir_label_and_zlabel:n
\}\AddToHook \{ env / sidecaption / begin \}
\AddToHook \{ env / sidecontcaption / begin \}
\AddToHook \{ cmd / bitwonumcaption / before \}
\AddToHook \{ cmd / bitwonumcaption / after \}
\AddToHook \{ cmd / bionenumcaption / before \}
\AddToHook \{ cmd / bionenumcaption / after \}
\AddToHook \{ cmd / bicaption / before \}
\AddToHook \{ cmd / bicaption / after \}
\Support for subcaption reference.
\zref@newprop \{ subcaption \}
\AddToHook \{ cmd / @memsubcaption / before \}
\AddToHook \{ cmd / @makefntext / before \}
\AddToHook \{ cmd / @makesidefntext / before \}
\Support for \footnote, \verbfootnote, \sidefootnote, and \pagenote.
\tl_new:N \l__zrefclever_memoir_footnote_type_tl
\tl_set:Nn \l__zrefclever_memoir_footnote_type_tl \{ footnote \}
\AddToHook \{ env / minipage / begin \}
\AddToHook \{ cmd / @makefntext / before \}
\AddToHook \{ cmd / @makesidefntext / before \}
\Support for \footnote, \verbfootnote, \sidefootnote, and \pagenote.
9.4 KOMA

Support for KOMA-Script document classes.

Add support for \captionbeside and \captionofbeside environments. These environments do run some variation of \caption and hence \refstepcounter. However, this happens inside a parbox inside the environment, thus grouped, such that we cannot see the variables set by \refstepcounter when we are setting the label. \@currentlabel is smuggled out of the group by KOMA, but the same care is not granted for \@currentcounter. So we have to rely on \@captype, which the underlying caption infrastructure feeds to \refstepcounter. Since we must use env/.../after hooks, care should be taken not to set the \currentcounter option unscoped, which would be quite disastrous. For this reason, though more “invasive”, we set \@currentcounter instead, which at least will be set straight the next time \refstepcounter runs. It sounds reasonable, it is the same treatment \@currentlabel is receiving in this case.
First, we define a function for label setting inside \texttt{amsmath} math environments, we want it to set both \zlabel and \label. We may “get a ride”, but not steal the place altogether. This makes for potentially redundant labels, but seems a good compromise. We \textit{must} use the lower level \texttt{zref@label} in this context, and hence also handle protection with \texttt{zref@wrapper@babel}, because \texttt{\zlabel} makes itself no-op when \texttt{\label} is equal to \texttt{\ltx@gobble}, and that’s precisely the case inside the \texttt{multline} environment (and, damn!, I took a beating of this detail...).

\begin{verbatim}
\cs_set_nopar:Npn \__zrefclever_ltxlabel:n #1
{\__zrefclever_orig_ltxlabel:n {#1}
\zref@wrapper@babel \zref@label {#1}}
\end{verbatim}

Then we must store the original value of \texttt{\ltx@label}, which is the macro actually responsible for setting the labels inside \texttt{amsmath}’s math environments. And, after that, redefine it to be \texttt{\__zrefclever_ltxlabel:n} instead. We must handle \texttt{hyperref} here, which comes very late in the preamble, and which loads \texttt{nameref} at \texttt{begindocument}, which in turn, lets \texttt{\ltx@label} be \texttt{\label}. This has to come after \texttt{nameref}. Other classes packages also redefine \texttt{\ltx@label}, which may cause some trouble. A \texttt{grep} on \texttt{texmf-dist} returns hits for: \texttt{thm-restate.sty}, \texttt{smartref.sty}, \texttt{jmlrbook.cls}, \texttt{cleveref.sty}, \texttt{cryptocode.sty}, \texttt{nameref.sty}, \texttt{easyeqn.sty}, \texttt{empheq.sty}, \texttt{ntheorem.sty}, \texttt{nccmath.sty}, \texttt{nwejm.cls}, \texttt{nwejmart.cls}, \texttt{aguplus.sty}, \texttt{aguplus.cls}, \texttt{agupp.sty}, \texttt{amsmath.hyp}, \texttt{amsmath.sty} (surprise!), \texttt{amsmath.4ht}, \texttt{nameref.4ht}, \texttt{frenchle.sty}, \texttt{french.sty}, plus corresponding documentations and different versions of the same packages. That’s not too many, but not “just a few” either. The critical ones are explicitly handled here (\texttt{amsmath} itself, and \texttt{nameref}). A number of those I’m really not acquainted with. For \texttt{cleveref}, in particular, this procedure is not compatible with it. If we happen to come later than it and override its definition, this may be a substantial problem for \texttt{cleveref}, since it will find the label, but it won’t contain the data it is expecting. However, this should normally not occur, if the user has followed the documented recommendation for \texttt{cleveref} to load it last, or at least very late, and besides I don’t see much of an use case for using both \texttt{cleveref} and \texttt{zref-clever} together. I have documented in the user manual that this module may cause potential issues, and how to work around them. And I have made an upstream feature request for a hook, so that this could be made more cleanly at \url{https://github.com/latex3/hyperref/issues/212}.

\begin{verbatim}
\__zrefclever_if_package_loaded:nTF {hyperref }
{\AddToHook { package / nameref / after }
{ \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label}
\end{verbatim}
The `subequations` environment uses `parentequation` and `equation` as counters, but only the later is subject to `refstepcounter`. What happens is: at the start, `equation` is re-stepped, it is then stored in `parentequation` and set to '0' and, at the end of the environment it is restored to the value of `parentequation`. We cannot even set `\@currentcounter` at `env/.../begin`, since the call to `\refstepcounter{equation}` done by `subequations` will override that in sequence. Unfortunately, the suggestion to set `\@currentcounter` to `parentequation` here was not accepted, see https://github.com/latex3/latex2e/issues/687#issuecomment-951451024 and subsequent discussion. So, for `subequations`, we really must specify manually `currentcounter` and the resetting. Note that, for `subequations`, `\zlabel` works just fine (that is, if given immediately after `begin(subequations)`, to refer to the parent equation).

\AddToHook { env / subequations / begin }
{ \__zrefclever_zcsetup:x
  \__zrefclever_counter_reset_by:n { equation },
  \__zrefclever_counter_reset_by:n { parentequation },
  \__zrefclever_counter_settype : { parentequation = equation },
}

\amsmath does use `refstepcounter` for the `equation` counter throughout and does set `\@currentcounter` for tags. But we still have to manually reset `currentcounter` to default because, since we had to manually set `currentcounter` to `parentequation` in `subequations`, we also have to manually set it to `equation` in environments which may be used within it. The `xxalignat` environment is not included, because it is “starred” by default (i.e. unnumbered), and does not display or accepts labels or tags anyway. The -ed (gathered, aligned, and alignedat) and cases environments “must appear within an enclosing math environment”. Same logic applies to other environments defined or redefined by the package, like array, matrix and variations. Finally, split too can only be used as part of another environment.
And a last touch of care for amsmath’s refinements: make the equation references \textup.

\zcRefTypeSetup { equation }
{
  reffont = \upshape ,
  preref = \{\textup{()} ,
  postref = \{\textup{)} ,
}
\msg_info:nnn { zref-clever } { compat-package } { amsmath }

\bool_new:N \l__zrefclever_mathtools_showonlyrefs_bool
\__zrefclever_compat_module:nn { mathtools }
{
  \__zrefclever_if_package_loaded:nT { mathtools }
  {
    \MH_if_boolean:nT { show_only_refs }
    {
      \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
      \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
      {
        \@bsphack
        \seq_map_inline:Nn #1
        { \exp_args:Nx \tl_if_eq:nnTF
          { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
          { equation }
        }
      }
    }
    \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
    \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
    {
      \@bsphack
      \seq_map_inline:Nn #1
      { \exp_args:Nx \tl_if_eq:nnTF
        { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
        { equation }
      }
    }
  }
}

9.6 mathtools

All math environments defined by mathtools, extending the amsmath set, are meant to be used within enclosing math environments, hence we don’t need to handle them specially, since the numbering and the counting is being done on the side of amsmath. This includes the new cases and matrix variants, and also multlined.

Hence, as far as I can tell, the only cross-reference related feature to deal with is the showonlyrefs option, whose machinery involves writing an extra internal label to the .aux file to track for labels which get actually referred to. This is a little more involved, and implies in doing special handling inside \zcref, but the feature is very cool, so it’s worth it.
From the \texttt{breqn} documentation: “Use of the normal \texttt{\label} command instead of the \texttt{label} option works, I think, most of the time (untested)”. Indeed, light testing suggests it does work for \texttt{\zlabel} just as well. However, if it happens not to work, there was no easy alternative handle I could find. In particular, it does not seem viable to leverage the \texttt{label=} option without hacking the package internals, even if the case of doing so would not be specially tricky, just “not very civil”.

Contrary to the practice in \texttt{amsmath}, which prints \texttt{\tag} even in unnumbered environments, the starred environments from \texttt{breqn} don’t typeset any tag/number at all, even for a manually given \texttt{number=} as an option. So, even if one can actually set a label in them, it is not really meaningful to make a reference to them. Also contrary to \texttt{amsmath}’s practice, \texttt{breqn} uses \texttt{\stepcounter} instead of \texttt{\refstepcounter} for incrementing the equation counters (see \url{https://tex.stackexchange.com/a/241150}).
\clist_map_inline:nn
  \dmath ,
  \dseries ,
  darray ,
\}
\AddToHook { env / #1 / begin }
  { \__zrefclever_zcsetup:n { currentcounter = equation } }
\msg_info:nnn { zref-clever } { compat-package } { breqn }
\}
\}

9.8 listings
\__zrefclever_compat_module:nn { listings }
\{ \__zrefclever_if_package_loaded:nT { listings }
\{ \__zrefclever_zcsetup:n
  \countertype =
  \{ lstlisting = listing ,
    lstnumber = line ,
  \},
  \counterresetby = { lstnumber = lstlisting } ,
\}
Set \texttt{currentcounter} with the label received in the \texttt{label=} option from the \texttt{lstlisting} environment. The \textit{only} place to set this label is the \texttt{PreInit} hook. This hook, comes right after \texttt{\lst@MakeCaption} in \texttt{\lst@Init}, which runs \texttt{\refstepcounter} on \texttt{lstlisting}, so we must come after it. Also \texttt{listings} itself sets \texttt{\@currentlabel} to \texttt{\the\lstnumber} in the \texttt{Init} hook, which comes right after the \texttt{PreInit} one in \texttt{\lst@Init}. Since, if we add to \texttt{Init} here, we go to the end of it, we’d be seeing the wrong \texttt{\@currentlabel} at that point.
\lst@AddToHook { PreInit }
  { \tl_if_empty:NF \lst@label { \zlabel { \lst@label } } }
Set \texttt{currentcounter} to \texttt{lstnumber} in the \texttt{Init} hook, since \texttt{listings} itself sets \texttt{\@currentlabel} to \texttt{\the\lstnumber} here. Note that \texttt{listings} does \textit{not} use \texttt{\refstepcounter} on \texttt{lstnumber}, but does so in the \texttt{EveryPar} hook, and there must be some grouping involved such that \texttt{\@currentcounter} ends up not being visible to the label. See section “Line numbers” of ‘\texttt{texdoc listings-devel}’ (the \texttt{.dtx}), and search for the definition of macro \texttt{\c@lstnumber}. Indeed, the fact that \texttt{listings} manually sets \texttt{\@currentlabel} to \texttt{\the\lstnumber} is a signal that the work of \texttt{\refstepcounter} is being restrained somehow.
\lst@AddToHook { Init }
  { \__zrefclever_zcsetup:n { currentcounter = lstnumber } }
\msg_info:nnn { zref-clever } { compat-package } { listings }
\}
\}
9.9  enumitem

The procedure below will “see” any changes made to the \texttt{enumerate} environment (made with \texttt{enumitem}'s \texttt{\renewlist}) as long as it is done in the preamble. Though, technically, \texttt{\renewlist} can be issued anywhere in the document, this should be more than enough for the purpose at hand. Besides, trying to retrieve this information “on the fly” would be much overkill.

The only real reason to “renew” \texttt{enumerate} itself is to change \texttt{\{max-depth\}}. \texttt{\renewlist} \texttt{hard-codes} \texttt{max-depth} in the environment’s definition (well, just as the kernel does), so we cannot retrieve this information from any sort of variable. But \texttt{\renewlist} also creates any needed missing counters, so we can use their existence to make the appropriate settings. In the end, the existence of the counters is indeed what matters from \texttt{zref-clever}'s perspective. Since the first four are defined by the kernel and already setup for \texttt{zref-clever} by default, we start from 5, and stop at the first non-existent \texttt{\c@enumN} counter.

9.10  subcaption

\__zrefclever_compat_module:nn \{ subcaption \}
\__zrefclever_if_package_loaded:nT \{ subcaption \}
\__zrefclever_zcsetup:n
{ countertype =
}
Support for `subref` reference.

\zref@newprop { subref }
\cs_if_exist_use:c { thesub \&captype } }
\tl_put_right:Nn \caption@subtypehook 
\zref@localaddprop \ZREF@mainlist { subref }
}
}
}\__zrefclever_compat_module:nn { subfig }
\__zrefclever_if_package_loaded:nT { subfig }
\__zrefclever_zcsetup:n
\zref@newprop { subref }
\cs_if_exist_use:c { thesub \&captype } }
\tl_put_right:Nn \caption@subtypehook 
\zref@localaddprop \ZREF@mainlist { subref }
}
}
\__zrefclever_if_package_loaded:nT { subfig }
\__zrefclever_zcsetup:n
\zref@newprop { subref }
\cs_if_exist_use:c { thesub \&captype } }
\tl_put_right:Nn \caption@subtypehook 
\zref@localaddprop \ZREF@mainlist { subref }
}
}
\__zrefclever_if_package_loaded:nT { subfig }
\__zrefclever_zcsetup:n
\zref@newprop { subref }
\cs_if_exist_use:c { thesub \&captype } }
\tl_put_right:Nn \caption@subtypehook 
\zref@localaddprop \ZREF@mainlist { subref }
}
}
\__zrefclever_if_package_loaded:nT { subfig }
\__zrefclever_zcsetup:n
\zref@newprop { subref }
\cs_if_exist_use:c { thesub \&captype } }
\tl_put_right:Nn \caption@subtypehook 
\zref@localaddprop \ZREF@mainlist { subref }
}
}
\__zrefclever_if_package_loaded:nT { subfig }
\__zrefclever_zcsetup:n
\zref@newprop { subref }
\cs_if_exist_use:c { thesub \&captype } }
\tl_put_right:Nn \caption@subtypehook 
\zref@localaddprop \ZREF@mainlist { subref }
}
}
\__zrefclever_if_package_loaded:nT { subfig }
\__zrefclever_zcsetup:n
\zref@newprop { subref }
\cs_if_exist_use:c { thesub \&captype } }
\tl_put_right:Nn \caption@subtypehook 
\zref@localaddprop \ZREF@mainlist { subref }
}
}
10.1 English

English language file has been initially provided by the author.

(*package)
\zcDeclareLanguage { english }
\zcDeclareLanguageAlias { american } { english }
\zcDeclareLanguageAlias { australian } { english }
\zcDeclareLanguageAlias { british } { english }
\zcDeclareLanguageAlias { canadian } { english }
\zcDeclareLanguageAlias { newzealand } { english }
\zcDeclareLanguageAlias { UKenglish } { english }
\zcDeclareLanguageAlias { USenglish } { english }
(*package)

(*lang-english)
namesep = {\nobreakspace},
pairsep = {\-
\nobreakspace},
listsep = {\-},
lastsep = {\-\nobreakspace},
tpairsep = {\-\nobreakspace},
tlistsep = {\-},
tlastsep = {\-\nobreakspace},
notesep = {\-},
rangesep = {\-to\nobreakspace},

type = book,
\quad Name-sg = Book,
\quad name-sg = book,
\quad Name-pl = Books,
\quad name-pl = books,

type = part,
\quad Name-sg = Part,
\quad name-sg = part,
\quad Name-pl = Parts,
\quad name-pl = parts,

type = chapter,
\quad Name-sg = Chapter,
\quad name-sg = chapter,
\quad Name-pl = Chapters,
\quad name-pl = chapters,

type = section,
\quad Name-sg = Section,
\quad name-sg = section,
\quad Name-pl = Sections,
\quad name-pl = sections,

type = paragraph,
\quad Name-sg = Paragraph,
\quad name-sg = paragraph,
\quad Name-pl = Paragraphs,
\quad name-pl = paragraphs,
Name-sg-ab = Par. ,
name-sg-ab = par. ,
Name-pl-ab = Par. ,
name-pl-ab = par. ,

type = appendix ,
Name-sg = Appendix ,
name-sg = appendix ,
Name-pl = Appendices ,
name-pl = appendices ,

type = page ,
Name-sg = Page ,
name-sg = page ,
Name-pl = Pages ,
name-pl = pages ,
rangesep = {\textendash} ,

type = line ,
Name-sg = Line ,
name-sg = line ,
Name-pl = Lines ,
name-pl = lines ,

type = figure ,
Name-sg = Figure ,
name-sg = figure ,
Name-pl = Figures ,
name-pl = figures ,
Name-sg-ab = Fig. ,
name-sg-ab = fig. ,
Name-pl-ab = Figs. ,
name-pl-ab = figs. ,

type = table ,
Name-sg = Table ,
name-sg = table ,
Name-pl = Tables ,
name-pl = tables ,

type = item ,
Name-sg = Item ,
name-sg = item ,
Name-pl = Items ,
name-pl = items ,

type = footnote ,
Name-sg = Footnote ,
name-sg = footnote ,
Name-pl = Footnotes ,
name-pl = footnotes ,

type = endnote ,
Name-sg = Note ,
name-sg = note,
Name-pl = Notes,
name-pl = notes,
type = note,
Name-sg = Note,
name-sg = note,
Name-pl = Notes,
name-pl = notes,
type = equation,
Name-sg = Equation,
name-sg = equation,
Name-pl = Equations,
name-pl = equations,
Name-sg-ab = Eq.,
name-sg-ab = eq.,
Name-pl-ab = Eqs.,
name-pl-ab = eqs.,
preref = {()},
postref = {}}

type = theorem,
Name-sg = Theorem,
name-sg = theorem,
Name-pl = Theorems,
name-pl = theorems,
type = lemma,
Name-sg = Lemma,
name-sg = lemma,
Name-pl = Lemmas,
name-pl = lemmas,
type = corollary,
Name-sg = Corollary,
name-sg = corollary,
Name-pl = Corollaries,
name-pl = corollaries,
type = proposition,
Name-sg = Proposition,
name-sg = proposition,
Name-pl = Propositions,
name-pl = propositions,
type = definition,
Name-sg = Definition,
name-sg = definition,
Name-pl = Definitions,
name-pl = definitions,
type = proof,
Name-sg = Proof,
German language file has been initially provided by the author.

```
\zcDeclareLanguage
[ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
```

10.2 German
{ german }
\zcDeclareLanguageAlias { austrian } { german }
\zcDeclareLanguageAlias { germanb } { german }
\zcDeclareLanguageAlias { ngerman } { german }
\zcDeclareLanguageAlias { naustrian } { german }
\zcDeclareLanguageAlias { nswissgerman } { german }
\zcDeclareLanguageAlias { swissgerman } { german }
⟨/package⟩
⟨∗lang-german⟩

namesep = {\nobreakspace} ,
pairssep = {\-\nobreakspace} ,
listsep = {\-\nobreakspace} ,
lastsep = {\-\nobreakspace} ,
tpairssep = {\-\nobreakspace} ,
tlistsep = {\-\nobreakspace} ,
tlastsep = {\-\nobreakspace} ,
noteseq = {\-} ,
rangesep = {\-\nobreakspace} ,
type = book ,
gender = n ,
case = N ,
Name-sg = Buch ,
Name-pl = Bücher ,
case = A ,
Name-sg = Buch ,
Name-pl = Bücher ,
case = D ,
Name-sg = Buch ,
Name-pl = Bücher ,
case = G ,
Name-sg = Buch ,
Name-pl = Bücher ,
type = part ,
gender = m ,
case = N ,
Name-sg = Teil ,
Name-pl = Teile ,
case = A ,
Name-sg = Teil ,
Name-pl = Teile ,
case = D ,
Name-sg = Teil ,
Name-pl = Teilen ,
case = G ,
Name-sg = Teile ,
Name-pl = Teile ,
type = chapter ,
gender = n ,
case = N ,
Name-sg = Kapitel ,
Name-pl = Anhänge,

type = page,
gender = f,
case = N,
Name-sg = Seite,
Name-pl = Seiten,
case = A,
Name-sg = Seite,
Name-pl = Seiten,
case = D,
Name-sg = Seite,
Name-pl = Seiten,
case = G,
Name-sg = Seite,
Name-pl = Seiten,
rangesep = {	extendash}

type = line,
gender = f,
case = N,
Name-sg = Zeile,
Name-pl = Zeilen,
case = A,
Name-sg = Zeile,
Name-pl = Zeilen,
case = D,
Name-sg = Zeile,
Name-pl = Zeilen,
case = G,
Name-sg = Zeile,
Name-pl = Zeilen,

type = figure,
gender = f,
case = N,
Name-sg = Abbildung,
Name-pl = Abbildungen,
Name-sg-ab = Abb.,
Name-pl-ab = Abb.,
case = A,
Name-sg = Abbildung,
Name-pl = Abbildungen,
Name-sg-ab = Abb.,
Name-pl-ab = Abb.,
case = D,
Name-sg = Abbildung,
Name-pl = Abbildungen,
Name-sg-ab = Abb.,
Name-pl-ab = Abb.,
case = G,
Name-sg = Abbildung,
Name-pl = Abbildungen,
Name-sg-ab = Abb.,
type = lemma,
gender = n,
case = N,
Name-sg = Lemma,
Name-pl = Lemmata,
case = A,
Name-sg = Lemma,
Name-pl = Lemmata,
case = D,
Name-sg = Lemma,
Name-pl = Lemmata,
case = G,
Name-sg = Lemmas,
Name-pl = Lemmata,

type = corollary,
gender = n,
case = N,
Name-sg = Korollar,
Name-pl = Korollare,
case = A,
Name-sg = Korollar,
Name-pl = Korollare,
case = D,
Name-sg = Korollar,
Name-pl = Korollaren,
case = G,
Name-sg = Korollars,
Name-pl = Korollare,

type = proposition,
gender = m,
case = N,
Name-sg = Satz,
Name-pl = Sätze,
case = A,
Name-sg = Satz,
Name-pl = Sätze,
case = D,
Name-sg = Satz,
Name-pl = Sätzen,
case = G,
Name-sg = Satzes,
Name-pl = Sätze,

type = definition,
gender = f,
case = N,
Name-sg = Definition,
Name-pl = Definitionen,
case = A,
Name-sg = Definition,
Name-pl = Definitionen,
case = D ,
Name-sg = Definition ,
Name-pl = Definitionen ,
case = G ,
Name-sg = Definition ,
Name-pl = Definitionen ,
type = proof ,
gender = m ,
case = N ,
Name-sg = Beweis ,
Name-pl = Beweise ,
case = A ,
Name-sg = Beweis ,
Name-pl = Beweise ,
case = D ,
Name-sg = Beweis ,
Name-pl = Beweisen ,
case = G ,
Name-sg = Beweises ,
Name-pl = Beweise ,

type = result ,
gender = n ,
case = N ,
Name-sg = Ergebnis ,
Name-pl = Ergebnisse ,
case = A ,
Name-sg = Ergebnis ,
Name-pl = Ergebnisse ,
case = D ,
Name-sg = Ergebnis ,
Name-pl = Ergebnissen ,
case = G ,
Name-sg = Ergebnisses ,
Name-pl = Ergebnisse ,

type = remark ,
gender = f ,
case = N ,
Name-sg = Bemerkung ,
Name-pl = Bemerkungen ,
case = A ,
Name-sg = Bemerkung ,
Name-pl = Bemerkungen ,
case = D ,
Name-sg = Bemerkung ,
Name-pl = Bemerkungen ,
case = G ,
Name-sg = Bemerkung ,
Name-pl = Bemerkungen ,

type = example ,
gender = n ,
case = N,
Name-sg = Beispiel,
Name-pl = Beispiele,
case = A,
Name-sg = Beispiel,
Name-pl = Beispiele,
case = D,
Name-sg = Beispiel,
Name-pl = Beispielen,
case = G,
Name-sg = Beispiels,
Name-pl = Beispiele,
type = algorithm,
gender = m,
case = N,
Name-sg = Algorithmus,
Name-pl = Algorithmen,
case = A,
Name-sg = Algorithmus,
Name-pl = Algorithmen,
case = D,
Name-sg = Algorithmus,
Name-pl = Algorithmen,
case = G,
Name-sg = Algorithmus,
Name-pl = Algorithmen,
type = listing,
gender = n,
case = N,
Name-sg = Listing,
Name-pl = Listings,
case = A,
Name-sg = Listing,
Name-pl = Listings,
case = D,
Name-sg = Listing,
Name-pl = Listings,
case = G,
Name-sg = Listings,
Name-pl = Listings,
type = exercise,
gender = f,
case = N,
Name-sg = Übungsaufgabe,
Name-pl = Übungsaufgaben,
case = A,
Name-sg = Übungsaufgabe,
Name-pl = Übungsaufgaben,
case = D,
Name-sg = Übungsaufgabe,
Name-pl = Übungsaufgaben,
### 10.3 French

French language file has been initially provided by the author, and has been improved thanks to Denis Bitouzé and François Lagarde (at issue #1) and participants of the Groupe francophone des Utilisateurs de TeX (GUTenberg) (at [https://groups.google.com/g/gut_fr/c/rNLM6weGcyg](https://groups.google.com/g/gut_fr/c/rNLM6weGcyg)) and the fr.comp.text.tex (at [https://groups.google.com/g/fr.comp.text.tex/c/Fa11T6FMFFs](https://groups.google.com/g/fr.comp.text.tex/c/Fa11T6FMFFs)) mailing lists.

```latex
(*package*)
\zcDeclareLanguage [ gender = { f , m } ] { french }
\zcDeclareLanguageAlias { acadian } { french }
\zcDeclareLanguageAlias { canadien } { french }
\zcDeclareLanguageAlias { francais } { french }
\zcDeclareLanguageAlias { frenchb } { french }
(*package*)

(*lang-french*)
namesep = {\nobreakspace} ,
pairsep = {-et\nobreakspace} ,
listsep = {,\nobreakspace} ,
lastsep = {et\nobreakspace} ,
tpairsep = {-et\nobreakspace} ,
tlistsep = {,\nobreakspace} ,
tlastsep = {et\nobreakspace} ,
notesep = {\-} ,
rangesep = {\-\-\nobreakspace} ,

(type = book ,
gender = m ,
Name-sg = Livre ,
name-sg = livre ,
Name-pl = Livres ,
name-pl = livres ,
```
type = part,
gender = f,
Name-sg = Partie,
name-sg = partie,
Name-pl = Parties,
name-pl = parties,
type = chapter,
gender = m,
Name-sg = Chapitre,
name-sg = chapitre,
Name-pl = Chapitres,
name-pl = chapitres,
type = section,
gender = f,
Name-sg = Section,
name-sg = section,
Name-pl = Sections,
name-pl = sections,
type = paragraph,
gender = m,
Name-sg = Paragraphe,
name-sg = paragraphe,
Name-pl = Paragraphes,
name-pl = paragraphes,
type = appendix,
gender = f,
Name-sg = Annexe,
name-sg = annexe,
Name-pl = Annexes,
name-pl = annexes,
type = page,
gender = f,
Name-sg = Page,
name-sg = page,
Name-pl = Pages,
name-pl = pages,
rangesep = {–},
type = line,
gender = f,
Name-sg = Ligne,
name-sg = ligne,
Name-pl = Lignes,
name-pl = lignes,
type = figure,
gender = f,
Name-sg = Figure,
name-sg = figure,
type = lemma ,
    gender = m ,
    Name-sg = Lemme ,
    name-sg = lemme ,
    Name-pl = Lemmes ,
    name-pl = lemmes ,

type = corollary ,
    gender = m ,
    Name-sg = Corollaire ,
    name-sg = corollaire ,
    Name-pl = Corollaires ,
    name-pl = corollaires ,

type = proposition ,
    gender = f ,
    Name-sg = Proposition ,
    name-sg = proposition ,
    Name-pl = Propositions ,
    name-pl = propositions ,

type = definition ,
    gender = f ,
    Name-sg = Définition ,
    name-sg = définition ,
    Name-pl = Définitions ,
    name-pl = définitions ,

type = proof ,
    gender = f ,
    Name-sg = Démonstration ,
    name-sg = démonstration ,
    Name-pl = Démonstrations ,
    name-pl = démonstrations ,

type = result ,
    gender = m ,
    Name-sg = Résultat ,
    name-sg = résultat ,
    Name-pl = Résultats ,
    name-pl = résultats ,

type = remark ,
    gender = f ,
    Name-sg = Remarque ,
    name-sg = remarque ,
    Name-pl = Remarques ,
    name-pl = remarques ,

type = example ,
    gender = m ,
    Name-sg = Exemple ,
    name-sg = exemple ,
    Name-pl = Exemples ,
10.4 Portuguese

Portuguese language file provided by the author, who’s a native speaker of (Brazilian) Portuguese. I do expect this to be sufficiently general, but if Portuguese speakers from other places feel the need for a Portuguese variant, please let me know.
type = book,  
gender = m,  
Name-sg = Livro,  
name-sg = livro,  
Name-pl = Livros,  
name-pl = livros,  

type = part,  
gender = f,  
Name-sg = Parte,  
name-sg = parte,  
Name-pl = Partes,  
name-pl = partes,  

type = chapter,  
gender = m,  
Name-sg = Capítulo,  
name-sg = capítulo,  
Name-pl = Capítulos,  
name-pl = capítulos,  

type = section,  
gender = f,  
Name-sg = Seção,  
name-sg = seção,  
Name-pl = Seções,  
name-pl = seções,  

type = paragraph,  
gender = m,  
Name-sg = Parágrafo,  
name-sg = parágrafo,  
Name-pl = Parágrafos,  
name-pl = parágrafos,  
Name-sg-ab = Par.,  
name-sg-ab = par.,  
Name-pl-ab = Par.,  
name-pl-ab = par.,  

type = appendix,  
gender = m,  
Name-sg = Apêndice,  
name-sg = apêndice,  
Name-pl = Apêndices,  
name-pl = apêndices,  

type = page,  
gender = f,  
Name-sg = Página,  
name-sg = página,  
Name-pl = Páginas,  
name-pl = páginas,  
rangesep = {\textendash},  

116
type = result,
  gender = m,
  Name-sg = Resultado,
  name-sg = resultado,
  Name-pl = Resultados,
  name-pl = resultados,

type = remark,
  gender = f,
  Name-sg = Observação,
  name-sg = observação,
  Name-pl = Observações,
  name-pl = observações,

type = example,
  gender = m,
  Name-sg = Exemplo,
  name-sg = exemplo,
  Name-pl = Exemplos,
  name-pl = exemplos,

type = algorithm,
  gender = m,
  Name-sg = Algoritmo,
  name-sg = algoritmo,
  Name-pl = Algoritmos,
  name-pl = algoritmos,

type = listing,
  gender = f,
  Name-sg = Listagem,
  name-sg = listagem,
  Name-pl = Listagens,
  name-pl = listagens,

type = exercise,
  gender = m,
  Name-sg = Exercício,
  name-sg = exercício,
  Name-pl = Exercícios,
  name-pl = exercícios,

type = solution,
  gender = f,
  Name-sg = Solução,
  name-sg = solução,
  Name-pl = Soluções,
  name-pl = soluções,

(/lang-portuguese)
10.5 Spanish

Spanish language file has been initially provided by the author.

(*package)
\zcDeclareLanguage [ gender = \{ f , m \} ] { spanish }
(*package)

(*lang-spanish)

namesep = {\nobreakspace} ,
pairsep = {-y\nobreakspace} ,
listsep = {,-} ,
lastsep = {-y\nobreakspace} ,
tpairsep = {-y\nobreakspace} ,
tlistsep = {,-} ,
tlastsep = {-y\nobreakspace} ,
notesep = { } ,
rangesep = {-a\nobreakspace} ,

type = book ,
gender = m ,
Name-sg = Libro ,
name-sg = libro ,
Name-pl = Libros ,
name-pl = libros ,

type = part ,
gender = f ,
Name-sg = Parte ,
name-sg = parte ,
Name-pl = Partes ,
name-pl = partes ,

type = chapter ,
gender = m ,
Name-sg = Capítulo ,
name-sg = capítulo ,
Name-pl = Capítulos ,
name-pl = capítulos ,

type = section ,
gender = f ,
Name-sg = Sección ,
name-sg = sección ,
Name-pl = Secciones ,
name-pl = secciones ,

type = paragraph ,
gender = m ,
Name-sg = Párrafo ,
name-sg = párrafo ,
Name-pl = Párrafos ,
name-pl = párrafos ,

type = appendix ,
gender = f,
Name-sg = Demostración,
name-sg = demostración,
Name-pl = Demostraciones,
name-pl = demostraciones,

type = result,
gender = m,
Name-sg = Resultado,
name-sg = resultado,
Name-pl = Resultados,
name-pl = resultados,

type = remark,
gender = f,
Name-sg = Observación,
name-sg = observación,
Name-pl = Observaciones,
name-pl = observaciones,

type = example,
gender = m,
Name-sg = Ejemplo,
name-sg = ejemplo,
Name-pl = Ejemplos,
name-pl = ejemplos,

type = algorithm,
gender = m,
Name-sg = Algoritmo,
name-sg = algoritmo,
Name-pl = Algoritmos,
name-pl = algoritmos,

type = listing,
gender = m,
Name-sg = Listado,
name-sg = listado,
Name-pl = Listados,
name-pl = listados,

type = exercise,
gender = m,
Name-sg = Ejercicio,
name-sg = ejercicio,
Name-pl = Ejercicios,
name-pl = ejercicios,

type = solution,
gender = f,
Name-sg = Solución,
name-sg = solución,
Name-pl = Soluciones,
name-pl = soluciones,
10.6 Dutch

Dutch language file initially contributed by niluxv (PR #5). All genders were checked against the “Dikke Van Dale”. Many words have multiple genders.

```latex
\zcDeclareLanguage [ gender = { f , m , n } ] { dutch }
```

```latex
\zcDeclareLanguage [ name-sg = Boek , name-pl = Boeken ,
 gender = n ,
 type = book ,
] { dutch }
```

```latex
\zcDeclareLanguage [ name-sg = Deel , name-pl = Delen ,
 gender = n ,
 type = part ,
] { dutch }
```

```latex
\zcDeclareLanguage [ name-sg = Hoofdstuk , name-pl = Hoofdstukken ,
 gender = n ,
 type = chapter ,
] { dutch }
```

```latex
\zcDeclareLanguage [ name-sg = Paragraaf , name-pl = Paragrafen ,
 gender = m ,
 type = section ,
] { dutch }
```

```latex
\zcDeclareLanguage [ name-sg = Alinea , name-pl = Alinea’s ,
 gender = f ,
 type = paragraph ,
] { dutch }
```

An alternative plural is “lemmata”. That is also a correct English plural for lemma, but the English language file chooses “lemmas”. For consistency we therefore choose “lemma’s”. 
name-pl = proposities,
type = definition,
gender = f,
Name-sg = Definitie,
name-sg = definitie,
Name-pl = Definities,
name-pl = definities,
type = proof,
gender = n,
Name-sg = Bewijs,
name-sg = bewijs,
Name-pl = Bewijzen,
name-pl = bewijzen,
type = result,
gender = n,
Name-sg = Resultaat,
name-sg = resultaat,
Name-pl = Resultaten,
name-pl = resultaten,
type = remark,
gender = f,
Name-sg = Opmerking,
name-sg = opmerking,
Name-pl = Opmerkingen,
name-pl = opmerkingen,
type = example,
gender = n,
Name-sg = Voorbeeld,
name-sg = voorbeeld,
Name-pl = Voorbeelden,
name-pl = voorbeelden,
type = algorithm,
gender = {n, f, m},
Name-sg = Algoritme,
name-sg = algoritme,
Name-pl = Algoritmes,
name-pl = algoritmes,

2022-01-09, niluxv: EN-NL Van Dale translates listing as (3) “uitdraai van computerprogramma”, “listing”.
type = listing,
gender = m,
Name-sg = Listing,
name-sg = listing,
Name-pl = Listings,
name-pl = listings,
type = exercise,
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
\AddToHook ............... 100, 983, 1026, 1051, 1087, 1089, 1137, 1210, 1226, 1247, 1378, 1391, 1399, 3366, 3402, 3419, 3421, 3426, 3471, 3473, 3475, 3477, 3479, 3481, 3483, 3485, 3489, 3493, 3495, 3500, 3510, 3519, 3519, 3525, 3527, 3553, 3563, 3595, 3647, 3668
\appendix ........................... 2, 86, 87, 89
\appendixname ........................ 86
\AtEndOfPackage ..................... 1389

B
\babelname ...................... 1036
\babelprovide .................... 18, 31
\bicaption ........................ 89
\bionumcaption ................... 89
\bitwonumcaption .................. 89
bool commands:
\ bool_case_true: .................... 2
\ bool_get_true:N ................. 1365
\ bool_if:NTF ..................... 562, 573, 987, 991, 1401, 1864, 2238, 2313, 2450, 2474, 2493, 2500, 2505, 2552, 2556, 2608, 2632, 2636, 2642, 2652, 2658, 2779, 2786, 2793, 2796, 2817, 2846, 2849, 2884, 2897, 2905, 2908, 2964, 2966
\ bool_if:nTF .................... 68, 1976, 1986, 2010, 2027, 2042, 2107, 2115, 2486, 2773, 2878, 3285
\ bool_lazy_all:nTF ............. 2580, 2976
\ bool_lazy_and:nTF ............. 1838, 1856, 2949, 2991, 3231, 3311
\ bool_lazy_any:nTF ............ 3111, 3120
\ bool_lazy_or:nTF .............. 910, 936, 1471, 1504, 1842, 2937, 3348
\ bool_new:N .................... 511, 834, 835, 862, 886, 895, 902, 928, 960, 961, 1016, 1145, 1146, 1147, 1148, 1149, 1240, 1241, 1358, 1872, 1885, 2147, 2148, 2158, 2165, 2166, 2179, 2180, 2181, 3607
\ bool_set:N ...................... 1835
\ bool_set_false:N ................ 847, 851, 968, 977, 978, 993, 1157, 1161, 1168, 1176, 1177, 1178, 1263, 1968, 2205, 2244, 2255, 2462, 2606, 2607, 3118, 3135
\ bool_set_true:N .............. 583, 841, 842, 846, 852, 967, 972, 973, 1155, 1162, 1167, 1184, 1186, 1188, 1191, 1192, 1193, 1251, 1256, 1982, 1992, 1996, 2018, 2033, 2048, 2072, 2213, 2239, 2245, 2249, 2256, 3134, 3137, 3170, 3177, 3178, 3196, 3203, 3207, 3213, 3215, 3614
\ bool_until_do:NN .............. 2008, 2206
\ bool_while_do:nn ............. 3699

C
\caption .......................... 91
clist commands:
\clist_clear:N ................. 645, 1675
\clist_if_empty:NTF .......... 656, 1686
\clist_if_in:NnTF ............ 3003
\clist_map_inline:nn ........ 646, 1179, 1306, 1367, 1676, 3577, 3661
\clist_put_right:Nn .......... 649, 1679
\clist_set:Nn .................. 3001

Index
V
\value .......................... 3405, 3424
\verb footnote  .................... 89, 90

Z
\zcDeclareLanguage .......... 13, 15, 51, 342, 3765, 3969, 4422, 4639, 4867, 5080
\zcDeclareLanguageAlias ......... 14, 359, 3766, 3767, 3768, 3769, 3770, 3771, 3772, 3972, 3973, 3974, 3975, 3976, 3977, 4423, 4424, 4425, 4426, 4640, 4641, 4642
\zclanguageSetup ................. 12, 17, 18, 20, 42, 45, 46, 1580
\zcpageref ..................... 52, 1873
\zcref ............................ 32, 35, 40, 41, 50, 52–54, 60, 62, 94, 1828, 1876, 1877
\zcrefTypeSetup ............... 12, 42, 1450, 3598
\zcssetup ........................ 31, 35, 40–42, 1445
\zslabel . 88, 89, 92, 93, 95, 96, 3467, 3688
zrefcheck commands:
\zrefcheck_zcref_beg_label: . 1841
\zrefcheck_zcref_end_label:–
maybe: .......................... 1860
\zrefcheck_zcref_run_checks_on_–
labels:n .......................... 1861
zrefclever internal commands:
\_zrefclever_abbrev_bool ...... 2167, 2309, 2950
\_zrefclever_base_language_tl .
\_zrefclever_capitalize_bool ..
.......................... 2167, 2307, 2938
\_zrefclever_capitalize_first_=–
bool .......................... 902, 924, 2940
\_zrefclever_compat_module:nn ..
.......................... 40, 1397, 3364, 3382, 3443, 3515, 3542, 3608, 3643, 3674, 3694, 3721, 3744
\_zrefclever_counter_reset_by:=n
.......................... 6, 38, 58, 60, 62, 96, 3570, 3654
\_zrefclever_counter_reset_by_–
aux:nn .......................... 73, 76
\_zrefclever_counter_reset_by_–
auxi:nn .......................... 83, 87
\_zrefclever_counter_resetby_–
prop .......................... 5, 38, 69, 70, 1327, 1339
\_zrefclever_counter_reseters_–
_seq .......................... 5, 38, 72, 1301, 1308, 1311
\_zrefclever_counter_type_prop
.......................... 4, 37, 35, 38, 1272, 1284
\_zrefclever_current_counter_–
tl .......................... 3, 5, 39, 21, 25, 26, 36, 39, 41, 46, 47, 95, 1351, 1354
\_zrefclever_current_language_–
tl .......................... 30, 31, 1024, 1030, 1036, 1040, 1045, 1058, 1078, 1101, 1121
\_zrefclever_declare_lang_opt_–
default:nn ................. 46, 1619, 1703, 1723, 1748, 1769
\_zrefclever_declare_lang_opt_–
type:nnnn ............. 46, 1619, 1688, 1708, 1728, 1774, 1813, 1819
\_zrefclever_extract:nnn ......... 11, 283, 2020, 2022, 2035, 2052, 2140, 2142, 3167, 3169, 3173, 3175, 3193, 3195, 3199, 3201
\_zrefclever_extract_default:NNn
\_zrefclever_extract_unexp:nnn
.......................... 11, 81, 277, 2516, 2784, 2790, 2803, 2830, 2842, 2893, 2901, 2915, 3141, 3144, 3145, 3163, 3164, 3185, 3186, 3189, 3190, 3207, 3211, 3621, 3629
\_zrefclever_get_encoding_–
counters_value:nn . 5, 6, 56, 61, 94
\_zrefclever_get_fallback_–
unknown_lang_opt:n ......... 804
\_zrefclever_get_fallback_–
unknown_lang_opt:NTF 25, 802, 3268
\_zrefclever_get_lang_opt_–
default:nn ............. 12, 787, 801
\_zrefclever_get_lang_opt_–
default:nnNTF 25, 786, 3263, 3343
\_zrefclever_get_lang_opt_–
type:nnN ............ 12, 771, 785
\_zrefclever_get_lang_opt_–
type:nnNTF ............ 24, 770, 2905, 3043, 3085, 3091, 3257, 3337
\_zrefclever_get_ref:nn ........ 73, 2345, 2362, 2375, 2380, 2404, 2419, 2423, 2436, 2440, 2477, 2497, 2674, 2689, 2696, 2721, 2734, 2738, 2749, 2753, 2769