The zref-clever package[*]
Code documentation

Gustavo Barros[†]

2023-02-13

EXPERIMENTAL

Contents

[*]This file describes v0.3.4, released 2023-02-13.
[†]`https://github.com/gusbrs/zref-clever`

1 Initial setup

Start the DocStrip guards.

```
⟨*package⟩
```

Identify the internal prefix (LaTeX3 DocStrip convention).

```
⟨@@=zrefclever⟩
```

Taking a stance on backward compatibility of the package. During initial development, we have used freely recent features of the kernel (albeit refraining from l3candidates, even though I'd have loved to have used `\bool_case_true:...`). We presume xparse (which made to the kernel in the 2020-10-01 release), and 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 (ltcmdhooks), with implications to the hook we add to `\appendix` (by Phelype Oleinik at https://tex.stackexchange.com/q/617905 and https://github.com/latex3/latex2e/pull/699). Second, the support for `\@currentcounter` has been improved, including `\footnote` and amsmath (by Frank Mittelbach and Ulrike Fischer at 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.

```
\providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
\IfFormatAtLeastTF{2021-11-15}
  {}
  {%
    \PackageError{zref-clever}{LaTeX kernel too old}
    {%
      'zref-clever' requires a LaTeX kernel 2021-11-15 or newer.%
      \MessageBreak Loading will abort!%
    }%
  \endinput
}%
```

Identify the package.

```
\ProvidesExplPackage {zref-clever} {2023-02-13} {0.3.4}
  {Clever LaTeX cross-references based on zref}
```

2 Dependencies

Required packages. Besides these, zref-hyperref may also be loaded depending on user options. zref-clever also requires UTF-8 input encoding (see discussion with David Carlisle at https://chat.stackexchange.com/transcript/message/62644791#62644791).

```
\RequirePackage { zref-base }
\RequirePackage { zref-user }
\RequirePackage { zref-abspage }
\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 `\l__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@ \l__zrefclever_current_counter_tl }
      { \use:c { the \l__zrefclever_current_counter_tl } }
      {
        \cs_if_exist:cT { c@ \@currentcounter }
          { \use:c { the \@currentcounter } }
      }
  }
\zref@addprop \ZREF@mainlist { thecounter }
```

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`.

```
\zref@newprop { zc@type }
  {
    \tl_if_empty:NTF \l__zrefclever_reftype_override_tl
      {
        \exp_args:NNe \prop_if_in:NnTF \l__zrefclever_counter_type_prop
          \l__zrefclever_current_counter_tl
          {
            \exp_args:NNe \prop_item:Nn \l__zrefclever_counter_type_prop
              { \l__zrefclever_current_counter_tl }
          }
          { \l__zrefclever_current_counter_tl }
      }
      { \l__zrefclever_reftype_override_tl }
  }
\zref@addprop \ZREF@mainlist { zc@type }
```

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 `\c@`⟨*counter*⟩, which contains the counter's numerical value (see 'texdoc source2e', section 'ltcounts.dtx').

```
\zref@newprop { zc@cntval } [0]
  {
    \cs_if_exist:cTF { c@ \l__zrefclever_current_counter_tl }
      { \int_use:c { c@ \l__zrefclever_current_counter_tl } }
      {
        \cs_if_exist:cT { c@ \@currentcounter }
          { \int_use:c { c@ \@currentcounter } }
      }
  }
\zref@addprop \ZREF@mainlist { zc@cntval }
\zref@newprop* { zc@pgval } [0] { \int_use:c { c@page } }
\zref@addprop \ZREF@mainlist { zc@pgval }
```

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 `begindocument` 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 `counterC` gets reset by `counterB`, and `counterB` gets reset by `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 `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 `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.

__zrefclever_get_enclosing_counters_value:n

Recursively generate a *sequence* of "enclosing counters" values, for a given ⟨*counter*⟩ and leave it in the input stream. This function must be expandable, since it gets called from `\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.

```
\__zrefclever_get_enclosing_counters_value:n {⟨counter⟩}
```

```
\cs_new:Npn \__zrefclever_get_enclosing_counters_value:n #1
  {
    \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
      {
        { \int_use:c { c@ \__zrefclever_counter_reset_by:n {#1} } }
        \__zrefclever_get_enclosing_counters_value:e
          { \__zrefclever_counter_reset_by:n {#1} }
      }
  }
```

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 (`x` 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*⟩.

```
\__zrefclever_counter_reset_by:n {⟨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@newprop { zc@enclval }
  {
    \__zrefclever_get_enclosing_counters_value:e
      \l__zrefclever_current_counter_tl
  }
\zref@addprop \ZREF@mainlist { 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 set `\c@page` to "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
\AddToHook { shipout / before }
  {
    \group_begin:
    \int_set:Nn \c@page { 1 }
    \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

__zrefclever_if_package_loaded:n
__zrefclever_if_class_loaded:n

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.

```
\prg_new_conditional:Npnn \__zrefclever_if_package_loaded:n #1 { T , F , TF }
  { \IfPackageLoadedTF {#1} { \prg_return_true: } { \prg_return_false: } }
```

```
\prg_new_conditional:Npnn \__zrefclever_if_class_loaded:n #1 { T , F , TF }
  { \IfClassLoadedTF {#1} { \prg_return_true: } { \prg_return_false: } }
```

(*End definition for* `\__zrefclever_if_package_loaded:n` *and* `\__zrefclever_if_class_loaded:n`.)

4.2 Messages

```
\msg_new:nnn { zref-clever } { option-not-type-specific }
  {
    Option~'#1'~is~not~type-specific~\msg_line_context:.~
    Set~it~in~'\iow_char: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 } { 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~'\iow_char:N\\zcDeclareLanguage'~and~
    '\iow_char: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~'\iow_char:N\\zcDeclareLanguage'~and~
    '\iow_char:N\\zcDeclareLanguageAlias'.
  }
\msg_new:nnn { zref-clever } { unknown-language-opt }
  {
    Language~'#1'~is~unknown~\msg_line_context:.~
    See~documentation~for~'\iow_char:N\\zcDeclareLanguage'~and~
    '\iow_char: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~'\iow_char:N\\zcDeclareLanguage'~and~
    '\iow_char: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 }
  {
    Language~'#1'~has~no~declared~gender~\msg_line_context:.~
    Nothing~to~do~with~option~'#2=#3'.
  }
\msg_new:nnn { zref-clever } { language-no-decl-setup }
  {
    Language~'#1'~has~no~declared~declension~cases~\msg_line_context:.~
    Nothing~to~do~with~option~'case=#2'.
  }
\msg_new:nnn { zref-clever } { unknown-decl-case }
  {
    Declension~case~'#1'~unknown~for~language~'#2'~\msg_line_context:.~
    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~\iow_char:N\\begin\{document\}. }
\msg_new:nnn { zref-clever } { langfile-loaded }
  { Loaded~'#1'~language~file. }
\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 } { endrange-property-undefined }
  {
    Option~'endrange=#1'~requested~\msg_line_context:.~
    But~the~property~'#1'~is~not~declared,~'endrange'~not~set.
  }
\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~
    '\iow_char:N\\zcref'.
  }
\msg_new:nnn { zref-clever } { missing-hyperref }
  { Missing~'hyperref'~package.~Setting~'hyperref=false'. }
\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 } { refbounds-must-be-four }
  {
    The~value~of~option~'#1'~must~be~a~comma~sepatared~list~
    of~four~items.~We~received~'#2'~items~\msg_line_context:.~
    Option~not~set.
  }
\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 } { zref-check-too-old }
  {
    Option~'check'~requested~\msg_line_context:.~
    But~'zref-check'~newer~than~'#1'~is~required,~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 } { 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:.\iow_newline:
    Use~'#2'~instead.
  }
\msg_new:nnn { zref-clever } { load-time-options }
  {
    'zref-clever'~does~not~accept~load-time~options.~
    To~configure~package~options,~use~'\iow_char:N\\zcsetup'.
  }
```

4.3 Data extraction

__zrefclever_extract_default:Nnnn

Extract property ⟨*prop*⟩ from ⟨*label*⟩ and sets variable ⟨*tl var*⟩ with extracted value. Ensure `\zref@extractdefault` is expanded exactly twice, but no further to retrieve the proper value. In case the property is not found, set ⟨*tl var*⟩ with ⟨*default*⟩.

```
\__zrefclever_extract_default:Nnnn {⟨tl var⟩}
  {⟨label⟩} {⟨prop⟩} {⟨default⟩}
```

```
\cs_new_protected:Npn \__zrefclever_extract_default:Nnnn #1#2#3#4
  {
    \exp_args:NNNo \exp_args:NNo \tl_set:Nn #1
      { \zref@extractdefault {#2} {#3} {#4} }
  }
\cs_generate_variant:Nn \__zrefclever_extract_default:Nnnn { NVnn , Nnvn }
```

(*End definition for* `\__zrefclever_extract_default:Nnnn`.)

__zrefclever_extract_unexp:nnn

Extract property ⟨*prop*⟩ from ⟨*label*⟩. Ensure that, in the context of an x expansion, `\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 ⟨*default*⟩ in the stream.

```
\__zrefclever_extract_unexp:nnn{⟨label⟩}{⟨prop⟩}{⟨default⟩}
```

```
\cs_new:Npn \__zrefclever_extract_unexp:nnn #1#2#3
  {
    \exp_args:NNo \exp_args:No
      \exp_not:n { \zref@extractdefault {#1} {#2} {#3} }
  }
\cs_generate_variant:Nn \__zrefclever_extract_unexp:nnn { Vnn , nvn , Vvn }
```

(*End definition for* `\__zrefclever_extract_unexp:nnn`.)

__zrefclever_extract:nnn

An internal version for `\zref@extractdefault`.

```
\__zrefclever_extract:nnn{⟨label⟩}{⟨prop⟩}{⟨default⟩}
```

```
\cs_new:Npn \__zrefclever_extract:nnn #1#2#3
  { \zref@extractdefault {#1} {#2} {#3} }
```

(*End definition for* `\__zrefclever_extract:nnn`.)

4.4 Option infra

This section provides the functions in which the variables naming scheme of the package options is embodied, and some basic general functions to query these option variables.

I had originally implemented the option handling of the package based on property lists, which are definitely very convenient. But as the number of options grew, I started to get concerned about the performance implications. That there was a toll was noticeable, even when we could live with it, of course. Indeed, at the time of writing, the typesetting of a reference queries about 24 different option values, most of them once per type-block, each of these queries can be potentially made in up to 5 option scope levels. Considering the size of the built-in language files is running at the hundreds, the package does have a lot of work to do in querying option values alone, and thus it is best to smooth things in this area as much as possible. This also gives me some peace of mind that the package will scale well in the long term. For some interesting discussion about alternative methods and their performance implications, see https://tex.stackexchange.com/q/147966. Phelype Oleinik also offered some insight on the matter at https://tex.stackexchange.com/questions/629946/#comment1571118_629946. The only real downside of this change is that we can no longer list the whole set of options in place at a given moment, which was useful for the purposes of regression testing, since we don't know what the whole set of active options is.

__zrefclever_opt_varname_general:nn

Defines, and leaves in the input stream, the csname of the variable used to store the general ⟨*option*⟩. The data type of the variable must be specified (`tl`, `seq`, `bool`, etc.).

```
\__zrefclever_opt_varname_general:nn {⟨option⟩} {⟨data type⟩}
```

```
\cs_new:Npn \__zrefclever_opt_varname_general:nn #1#2
  { l__zrefclever_opt_general_ #1 _ #2 }
```

(*End definition for* `\__zrefclever_opt_varname_general:nn`.)

__zrefclever_opt_varname_type:nnn

Defines, and leaves in the input stream, the csname of the variable used to store the type-specific ⟨*option*⟩ for ⟨*ref type*⟩.

```
\__zrefclever_opt_varname_type:nnn {⟨ref type⟩} {⟨option⟩} {⟨data type⟩}
```

```
\cs_new:Npn \__zrefclever_opt_varname_type:nnn #1#2#3
  { l__zrefclever_opt_type_ #1 _ #2 _ #3 }
\cs_generate_variant:Nn \__zrefclever_opt_varname_type:nnn { enn , een }
```

(*End definition for* `\__zrefclever_opt_varname_type:nnn`.)

__zrefclever_opt_varname_language:nnn

Defines, and leaves in the input stream, the csname of the variable used to store the language ⟨*option*⟩ for ⟨*lang*⟩ (for general language options, those set with `\zcDeclareLanguage`). The "`lang_unknown`" branch should be guarded against, such as we normally should not get there, but this function *must* return some valid csname. The random part is there so that, in the circumstance this could not be avoided, we (hopefully) don't retrieve the value for an "unknown language" inadvertently.

```
\__zrefclever_opt_varname_language:nnn {⟨lang⟩} {⟨option⟩} {⟨data type⟩}
```

```
\cs_new:Npn \__zrefclever_opt_varname_language:nnn #1#2#3
  {
    \__zrefclever_language_if_declared:nTF {#1}
      {
        g__zrefclever_opt_language_
        \tl_use:c { \__zrefclever_language_varname:n {#1} }
        _ #2 _ #3
      }
      { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
  }
\cs_generate_variant:Nn \__zrefclever_opt_varname_language:nnn { enn }
```

(*End definition for* `\__zrefclever_opt_varname_language:nnn`.)

__zrefclever_opt_varname_lang_default:nnn

Defines, and leaves in the input stream, the csname of the variable used to store the language-specific default reference format ⟨*option*⟩ for ⟨*lang*⟩.

```
\__zrefclever_opt_varname_lang_default:nnn {⟨lang⟩} {⟨option⟩} {⟨data type⟩}
```

```
\cs_new:Npn \__zrefclever_opt_varname_lang_default:nnn #1#2#3
  {
    \__zrefclever_language_if_declared:nTF {#1}
      {
        g__zrefclever_opt_lang_
        \tl_use:c { \__zrefclever_language_varname:n {#1} }
        _default_ #2 _ #3
      }
      { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
  }
\cs_generate_variant:Nn \__zrefclever_opt_varname_lang_default:nnn { enn }
```

(*End definition for* `\__zrefclever_opt_varname_lang_default:nnn`.)

__zrefclever_opt_varname_lang_type:nnnn

Defines, and leaves in the input stream, the csname of the variable used to store the language- and type-specific reference format ⟨*option*⟩ for ⟨*lang*⟩ and ⟨*ref type*⟩.

```
\__zrefclever_opt_varname_lang_type:nnnn {⟨lang⟩} {⟨ref type⟩}
  {⟨option⟩} {⟨data type⟩}
```

```
\cs_new:Npn \__zrefclever_opt_varname_lang_type:nnnn #1#2#3#4
  {
    \__zrefclever_language_if_declared:nTF {#1}
      {
        g__zrefclever_opt_lang_
        \tl_use:c { \__zrefclever_language_varname:n {#1} }
        _type_ #2 _ #3 _ #4
      }
      { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #4 }
  }
\cs_generate_variant:Nn
  \__zrefclever_opt_varname_lang_type:nnnn { eenn , eeen }
```

(*End definition for* `\__zrefclever_opt_varname_lang_type:nnnn`.)

__zrefclever_opt_varname_fallback:nn

Defines, and leaves in the input stream, the csname of the variable used to store the fallback ⟨*option*⟩.

```
\__zrefclever_opt_varname_fallback:nn {⟨option⟩} {⟨data type⟩}
```

```
\cs_new:Npn \__zrefclever_opt_varname_fallback:nn #1#2
  { c__zrefclever_opt_fallback_ #1 _ #2 }
```

(*End definition for* `\__zrefclever_opt_varname_fallback:nn`.)

__zrefclever_opt_var_set_bool:n

The LaTeX3 programming layer does not have the concept of a variable *existing* only locally, it also considers an "error" if an assignment is made to a variable which was not previously declared, but declaration is always global, which means that "setting a local variable at a local scope", given these requirements, results in it existing, and being empty, globally. Therefore, we need an independent mechanism from the mere existence of a variable to keep track of whether variables are "set" or "unset", within the logic of the precedence rules for options in different scopes. `\__zrefclever_opt_var_set_bool:n` expands to the name of the boolean variable used to track this state for ⟨*option var*⟩. See discussion with Phelype Oleinik at https://tex.stackexchange.com/questions/633341/#comment1579825_633347

```
\__zrefclever_opt_var_set_bool:n {⟨option var⟩}
```

```
\cs_new:Npn \__zrefclever_opt_var_set_bool:n #1
  { \cs_to_str:N #1 _is_set_bool }
```

(*End definition for* `\__zrefclever_opt_var_set_bool:n`*.*)

__zrefclever_opt_tl_set:Nn
__zrefclever_opt_tl_clear:N
__zrefclever_opt_tl_gset:Nn
__zrefclever_opt_tl_gclear:N

```
\__zrefclever_opt_tl_set:N {⟨option tl⟩} {⟨value⟩}
\__zrefclever_opt_tl_clear:N {⟨option tl⟩}
\__zrefclever_opt_tl_gset:N {⟨option tl⟩} {⟨value⟩}
\__zrefclever_opt_tl_gclear:N {⟨option tl⟩}
```

```
\cs_new_protected:Npn \__zrefclever_opt_tl_set:Nn #1#2
  {
    \tl_if_exist:NF #1
      { \tl_new:N #1 }
    \tl_set:Nn #1 {#2}
    \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
      { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
    \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
  }
\cs_generate_variant:Nn \__zrefclever_opt_tl_set:Nn { cn }
\cs_new_protected:Npn \__zrefclever_opt_tl_clear:N #1
  {
    \tl_if_exist:NF #1
      { \tl_new:N #1 }
    \tl_clear:N #1
    \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
      { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
    \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
  }
\cs_generate_variant:Nn \__zrefclever_opt_tl_clear:N { c }
\cs_new_protected:Npn \__zrefclever_opt_tl_gset:Nn #1#2
  {
    \tl_if_exist:NF #1
      { \tl_new:N #1 }
    \tl_gset:Nn #1 {#2}
  }
\cs_generate_variant:Nn \__zrefclever_opt_tl_gset:Nn { cn }
\cs_new_protected:Npn \__zrefclever_opt_tl_gclear:N #1
  {
    \tl_if_exist:NF #1
      { \tl_new:N #1 }
    \tl_gclear:N #1
  }
\cs_generate_variant:Nn \__zrefclever_opt_tl_gclear:N { c }
```

(*End definition for* `\__zrefclever_opt_tl_set:Nn` *and others.*)

__zrefclever_opt_tl_unset:N Unset ⟨*option tl*⟩.

```
\__zrefclever_opt_tl_unset:N {⟨option tl⟩}
```

```
\cs_new_protected:Npn \__zrefclever_opt_tl_unset:N #1
  {
    \tl_if_exist:NT #1
```

```
      {
        \tl_clear:N #1 % ?
        \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
          { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
          { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
      }
  }
\cs_generate_variant:Nn \__zrefclever_opt_tl_unset:N { c }
```

(*End definition for* `\__zrefclever_opt_tl_unset:N`.)

`\__zrefclever_opt_tl_if_set:NTF`

This conditional *defines* what means to be unset for a token list option. Note that the "set bool" not existing signals that the variable *is set*, that would be the case of all global option variables (language-specific ones). But this means care should be taken to always define and set the "set bool" for local variables.

```
\__zrefclever_opt_tl_if_set:N(TF) {⟨option tl⟩} {⟨true⟩} {⟨false⟩}
```

```
\prg_new_conditional:Npnn \__zrefclever_opt_tl_if_set:N #1 { F , TF }
  {
    \tl_if_exist:NTF #1
      {
        \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
          {
            \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
              { \prg_return_true:  }
              { \prg_return_false: }
          }
          { \prg_return_true: }
      }
      { \prg_return_false: }
  }
```

(*End definition for* `\__zrefclever_opt_tl_if_set:NTF`.)

`\__zrefclever_opt_tl_gset_if_new:Nn`
`\__zrefclever_opt_tl_gclear_if_new:N`

```
\__zrefclever_opt_tl_gset_if_new:Nn {⟨option tl⟩} {⟨value⟩}
\__zrefclever_opt_tl_gclear_if_new:N {⟨option tl⟩}
```

```
\cs_new_protected:Npn \__zrefclever_opt_tl_gset_if_new:Nn #1#2
  {
    \__zrefclever_opt_tl_if_set:NF #1
      {
        \tl_if_exist:NF #1
          { \tl_new:N #1 }
        \tl_gset:Nn #1 {#2}
      }
  }
\cs_generate_variant:Nn \__zrefclever_opt_tl_gset_if_new:Nn { cn }
\cs_new_protected:Npn \__zrefclever_opt_tl_gclear_if_new:N #1
  {
    \__zrefclever_opt_tl_if_set:NF #1
      {
        \tl_if_exist:NF #1
          { \tl_new:N #1 }
        \tl_gclear:N #1
      }
  }
\cs_generate_variant:Nn \__zrefclever_opt_tl_gclear_if_new:N { c }
```

(*End definition for* `\__zrefclever_opt_tl_gset_if_new:Nn` *and* `\__zrefclever_opt_tl_gclear_if_new:N`.)

__zrefclever_opt_tl_get:NN*TF*

```
  \__zrefclever_opt_tl_get:NN(TF) {⟨option tl to get⟩} {⟨tl var to set⟩}
    {⟨true⟩} {⟨false⟩}
```

```
\prg_new_protected_conditional:Npnn \__zrefclever_opt_tl_get:NN #1#2 { F }
  {
    \__zrefclever_opt_tl_if_set:NTF #1
      {
        \tl_set_eq:NN #2 #1
        \prg_return_true:
      }
      { \prg_return_false: }
  }
\prg_generate_conditional_variant:Nnn
  \__zrefclever_opt_tl_get:NN { cN } { F }
```

(*End definition for* `\__zrefclever_opt_tl_get:NNTF`.)

__zrefclever_opt_seq_set_clist_split:Nn
__zrefclever_opt_seq_gset_clist_split:Nn
__zrefclever_opt_seq_set_eq:NN
__zrefclever_opt_seq_gset_eq:NN

```
  \__zrefclever_opt_seq_set_clist_split:Nn {⟨option seq⟩} {⟨value⟩}
  \__zrefclever_opt_seq_gset_clist_split:Nn {⟨option seq⟩} {⟨value⟩}
  \__zrefclever_opt_seq_set_eq:NN {⟨option seq⟩} {⟨seq var⟩}
  \__zrefclever_opt_seq_gset_eq:NN {⟨option seq⟩} {⟨seq var⟩}
```

```
\cs_new_protected:Npn \__zrefclever_opt_seq_set_clist_split:Nn #1#2
  { \seq_set_split:Nnn #1 { , } {#2} }
\cs_new_protected:Npn \__zrefclever_opt_seq_gset_clist_split:Nn #1#2
  { \seq_gset_split:Nnn #1 { , } {#2} }
\cs_new_protected:Npn \__zrefclever_opt_seq_set_eq:NN #1#2
  {
    \seq_if_exist:NF #1
      { \seq_new:N #1 }
    \seq_set_eq:NN #1 #2
    \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
      { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
    \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
  }
\cs_generate_variant:Nn \__zrefclever_opt_seq_set_eq:NN { cN }
\cs_new_protected:Npn \__zrefclever_opt_seq_gset_eq:NN #1#2
  {
    \seq_if_exist:NF #1
      { \seq_new:N #1 }
    \seq_gset_eq:NN #1 #2
  }
\cs_generate_variant:Nn \__zrefclever_opt_seq_gset_eq:NN { cN }
```

(*End definition for* `\__zrefclever_opt_seq_set_clist_split:Nn` *and others.*)

__zrefclever_opt_seq_unset:N

Unset ⟨*option seq*⟩.

```
  \__zrefclever_opt_seq_unset:N {⟨option seq⟩}
```

```
\cs_new_protected:Npn \__zrefclever_opt_seq_unset:N #1
  {
    \seq_if_exist:NT #1
      {
        \seq_clear:N #1 % ?
        \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
          { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
          { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
      }
  }
\cs_generate_variant:Nn \__zrefclever_opt_seq_unset:N { c }
```

(*End definition for* `\__zrefclever_opt_seq_unset:N`.)

`\__zrefclever_opt_seq_if_set:NTF` This conditional *defines* what means to be unset for a sequence option.

```
\__zrefclever_opt_seq_if_set:N(TF) {⟨option seq⟩} {⟨true⟩} {⟨false⟩}
```

```
\prg_new_conditional:Npnn \__zrefclever_opt_seq_if_set:N #1 { F , TF }
  {
    \seq_if_exist:NTF #1
      {
        \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
          {
            \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
              { \prg_return_true:  }
              { \prg_return_false: }
          }
          { \prg_return_true: }
      }
      { \prg_return_false: }
  }
\prg_generate_conditional_variant:Nnn
  \__zrefclever_opt_seq_if_set:N { c } { F , TF }
```

(*End definition for* `\__zrefclever_opt_seq_if_set:NTF`.)

`__zrefclever_opt_seq_get:NNTF`

```
\__zrefclever_opt_seq_get:NN(TF) {⟨option seq to get⟩} {⟨seq var to set⟩}
  {⟨true⟩} {⟨false⟩}
```

```
\prg_new_protected_conditional:Npnn \__zrefclever_opt_seq_get:NN #1#2 { F }
  {
    \__zrefclever_opt_seq_if_set:NTF #1
      {
        \seq_set_eq:NN #2 #1
        \prg_return_true:
      }
      { \prg_return_false: }
  }
\prg_generate_conditional_variant:Nnn
  \__zrefclever_opt_seq_get:NN { cN } { F }
```

(*End definition for* `\__zrefclever_opt_seq_get:NNTF`.)

`\__zrefclever_opt_bool_unset:N` Unset ⟨*option bool*⟩.

```
\__zrefclever_opt_bool_unset:N {⟨option bool⟩}
```

```
\cs_new_protected:Npn \__zrefclever_opt_bool_unset:N #1
  {
    \bool_if_exist:NT #1
      {
        % \bool_set_false:N #1 % ?
        \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
          { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
          { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
      }
  }
\cs_generate_variant:Nn \__zrefclever_opt_bool_unset:N { c }
```

(*End definition for* `\__zrefclever_opt_bool_unset:N`.)

__zrefclever_opt_bool_if_set:N*TF*

This conditional *defines* what means to be unset for a boolean option.

`\__zrefclever_opt_bool_if_set:N(TF) {⟨option bool⟩} {⟨true⟩} {⟨false⟩}`

```
\prg_new_conditional:Npnn \__zrefclever_opt_bool_if_set:N #1 { F , TF }
  {
    \bool_if_exist:NTF #1
      {
        \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
          {
            \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
              { \prg_return_true:  }
              { \prg_return_false: }
          }
          { \prg_return_true: }
      }
      { \prg_return_false: }
  }
\prg_generate_conditional_variant:Nnn
  \__zrefclever_opt_bool_if_set:N { c } { F , TF }
```

(*End definition for* `\__zrefclever_opt_bool_if_set:NTF`.)

__zrefclever_opt_bool_set_true:N
__zrefclever_opt_bool_set_false:N
__zrefclever_opt_bool_gset_true:N
__zrefclever_opt_bool_gset_false:N

```
\__zrefclever_opt_bool_set_true:N {⟨option bool⟩}
\__zrefclever_opt_bool_set_false:N {⟨option bool⟩}
\__zrefclever_opt_bool_gset_true:N {⟨option bool⟩}
\__zrefclever_opt_bool_gset_false:N {⟨option bool⟩}
```

```
\cs_new_protected:Npn \__zrefclever_opt_bool_set_true:N #1
  {
    \bool_if_exist:NF #1
      { \bool_new:N #1 }
    \bool_set_true:N #1
    \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
      { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
    \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
  }
\cs_generate_variant:Nn \__zrefclever_opt_bool_set_true:N { c }
\cs_new_protected:Npn \__zrefclever_opt_bool_set_false:N #1
  {
    \bool_if_exist:NF #1
      { \bool_new:N #1 }
    \bool_set_false:N #1
    \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
      { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
    \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
  }
\cs_generate_variant:Nn \__zrefclever_opt_bool_set_false:N { c }
\cs_new_protected:Npn \__zrefclever_opt_bool_gset_true:N #1
  {
    \bool_if_exist:NF #1
      { \bool_new:N #1 }
    \bool_gset_true:N #1
  }
\cs_generate_variant:Nn \__zrefclever_opt_bool_gset_true:N { c }
\cs_new_protected:Npn \__zrefclever_opt_bool_gset_false:N #1
  {
    \bool_if_exist:NF #1
      { \bool_new:N #1 }
    \bool_gset_false:N #1
  }
\cs_generate_variant:Nn \__zrefclever_opt_bool_gset_false:N { c }
```

(*End definition for* `\__zrefclever_opt_bool_set_true:N` *and others.*)

__zrefclever_opt_bool_get:NN*TF*

`\__zrefclever_opt_bool_get:NN(TF) {⟨option bool to get⟩} {⟨bool var to set⟩} {⟨true⟩} {⟨false⟩}`

```
\prg_new_protected_conditional:Npnn \__zrefclever_opt_bool_get:NN #1#2 { F }
  {
    \__zrefclever_opt_bool_if_set:NTF #1
      {
        \bool_set_eq:NN #2 #1
        \prg_return_true:
      }
      { \prg_return_false: }
  }
\prg_generate_conditional_variant:Nnn
  \__zrefclever_opt_bool_get:NN { cN } { F }
```

(*End definition for* `\__zrefclever_opt_bool_get:NNTF`.)

__zrefclever_opt_bool_if:N*TF*

`\__zrefclever_opt_bool_if:N(TF) {⟨option bool⟩} {⟨true⟩} {⟨false⟩}`

```
\prg_new_conditional:Npnn \__zrefclever_opt_bool_if:N #1 { T , F , TF }
  {
    \__zrefclever_opt_bool_if_set:NTF #1
      { \bool_if:NTF #1 { \prg_return_true: } { \prg_return_false: } }
      { \prg_return_false: }
  }
\prg_generate_conditional_variant:Nnn
  \__zrefclever_opt_bool_if:N { c } { T , F , TF }
```

(*End definition for* `\__zrefclever_opt_bool_if:NTF`.)

4.5 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 `\__zrefclever_get_rf_opt_tl:nnnN`, `\__zrefclever_get_rf_opt_seq:nnnN`, `\__zrefclever_get_rf_opt_bool:nnnnN`, and `\__zrefclever_type_name_setup:` which are the basic functions to retrieve proper values for reference format settings.

The fact that we have multiple scopes to set reference format options has some implications for how we handle these options, and for the resulting UI. Since there is a clear precedence rule between the different levels, setting an option at a high priority level shadows everything below it. Hence, it may be relevant to be able to "unset" these options too, so as to be able go back to the lower precedence level of the language-specific options at any given point. However, since many of these options are token lists, or clists, for which "empty" is a legitimate value, we cannot rely on emptiness to distinguish that particular intention. How to deal with it, depends on the kind of option (its data type, to be precise). For token lists and clists/sequences, 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 in `\keys_define:nn` by means of the `.default:o` property of the key. 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 and https://github.com/latex3/latex3/pull/988. However, Joseph Wright seems to particularly dislike this use and the general idea of a "key with no value" being somehow meaningful for l3keys (e.g. his comments on the previous question, and https://tex.stackexchange.com/q/632157/#comment1576404_632157), which does make it somewhat risky to rely on this. For booleans, the situation is different, since they cannot meaningfully receive an empty value and the "key with no value" is a handy and expected shorthand for `key=true`. Therefore, for reference format option booleans, we use a third value "`unset`" for this purpose. And similarly for "choice" options.

However, "unsetting" options is only supported at the general and reference type levels, that is, at `\zcsetup`, at `\zcref`, and at `\zcRefTypeSetup`. For language-specific options – in the language files or at `\zcLanguageSetup` – there is no unsetting, an option which has been set can there only be changed to another value. This for two reasons. First, these are low precedence levels, so it is less meaningful to be able to unset these options. Second, these settings can only be done in the preamble (or the package itself). They are meant to be global. So, do it once, do it right, and if you need to locally change something along the document, use a higher precedence level.

`\l__zrefclever_setup_type_tl`
`\l__zrefclever_setup_language_tl`
`\l__zrefclever_lang_decl_case_tl`
`\l__zrefclever_lang_declension_seq`
`\l__zrefclever_lang_gender_seq`

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 specific options retrieval.

```
\tl_new:N \l__zrefclever_setup_type_tl
\tl_new:N \l__zrefclever_setup_language_tl
\tl_new:N \l__zrefclever_lang_decl_case_tl
\seq_new:N \l__zrefclever_lang_declension_seq
\seq_new:N \l__zrefclever_lang_gender_seq
```

(*End definition for* `\l__zrefclever_setup_type_tl` *and others.*)

zrefclever_rf_opts_tl_not_type_specific_seq
efclever_rf_opts_tl_maybe_type_specific_seq
\g__zrefclever_rf_opts_seq_refbounds_seq
clever_rf_opts_bool_maybe_type_specific_seq
\g__zrefclever_rf_opts_tl_type_names_seq
\g__zrefclever_rf_opts_tl_typesetup_seq
\g__zrefclever_rf_opts_tl_reference_seq

Lists of reference format 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. These variables are *constants*, but I don't seem to be able to find a way to concatenate two constants into a third one without triggering LaTeX3 debug error “Inconsistent local/global assignment”. And repeating things in a new `\seq_const_from_clist:Nn` defeats the purpose of these variables.

```
\seq_new:N \g__zrefclever_rf_opts_tl_not_type_specific_seq
\seq_gset_from_clist:Nn
  \g__zrefclever_rf_opts_tl_not_type_specific_seq
  {
    tpairsep ,
    tlistsep ,
    tlastsep ,
    notesep ,
  }
\seq_new:N \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
\seq_gset_from_clist:Nn
  \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
  {
    namesep ,
    pairsep ,
    listsep ,
    lastsep ,
    rangesep ,
    namefont ,
    reffont ,
  }
\seq_new:N \g__zrefclever_rf_opts_seq_refbounds_seq
\seq_gset_from_clist:Nn
  \g__zrefclever_rf_opts_seq_refbounds_seq
  {
    refbounds-first ,
    refbounds-first-sg ,
    refbounds-first-pb ,
    refbounds-first-rb ,
    refbounds-mid ,
    refbounds-mid-rb ,
    refbounds-mid-re ,
    refbounds-last ,
    refbounds-last-pe ,
    refbounds-last-re ,
  }
\seq_new:N \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
\seq_gset_from_clist:Nn
  \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
  {
    cap ,
    abbrev ,
    rangetopair ,
  }
```

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_rf_opt_tl:nnnN, but by __zrefclever_type_name_setup:.

```
\seq_new:N \g__zrefclever_rf_opts_tl_type_names_seq
\seq_gset_from_clist:Nn
  \g__zrefclever_rf_opts_tl_type_names_seq
  {
    Name-sg ,
    name-sg ,
    Name-pl ,
    name-pl ,
    Name-sg-ab ,
    name-sg-ab ,
    Name-pl-ab ,
    name-pl-ab ,
  }
```

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

```
\seq_new:N \g__zrefclever_rf_opts_tl_typesetup_seq
\seq_gconcat:NNN \g__zrefclever_rf_opts_tl_typesetup_seq
  \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
  \g__zrefclever_rf_opts_tl_type_names_seq
\seq_new:N \g__zrefclever_rf_opts_tl_reference_seq
\seq_gconcat:NNN \g__zrefclever_rf_opts_tl_reference_seq
  \g__zrefclever_rf_opts_tl_not_type_specific_seq
  \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
```

(*End definition for* \g__zrefclever_rf_opts_tl_not_type_specific_seq *and others.*)

We set here also the "derived" refbounds options, which are (almost) the same for every option scope.

```
\clist_map_inline:nn
  {
    reference ,
    typesetup ,
    langsetup ,
    langfile ,
  }
  {
    \keys_define:nn { zref-clever/ #1 }
      {
        +refbounds-first .meta:n =
          {
            refbounds-first = {##1} ,
            refbounds-first-sg = {##1} ,
            refbounds-first-pb = {##1} ,
            refbounds-first-rb = {##1} ,
          } ,
        +refbounds-mid .meta:n =
          {
            refbounds-mid = {##1} ,
            refbounds-mid-rb = {##1} ,
            refbounds-mid-re = {##1} ,
          } ,
        +refbounds-last .meta:n =
          {
            refbounds-last = {##1} ,
            refbounds-last-pe = {##1} ,
            refbounds-last-re = {##1} ,
          } ,
        +refbounds-rb .meta:n =
          {
            refbounds-first-rb = {##1} ,
            refbounds-mid-rb = {##1} ,
          } ,
        +refbounds-re .meta:n =
          {
            refbounds-mid-re = {##1} ,
            refbounds-last-re = {##1} ,
          } ,
        +refbounds .meta:n =
          {
            +refbounds-first = {##1} ,
            +refbounds-mid = {##1} ,
            +refbounds-last = {##1} ,
          } ,
        refbounds .meta:n = { +refbounds = {##1} } ,
      }
  }
\clist_map_inline:nn
  {
    reference ,
    typesetup ,
  }
  {
    \keys_define:nn { zref-clever/ #1 }
      {
        +refbounds-first .default:o = \c_novalue_tl ,
        +refbounds-mid .default:o = \c_novalue_tl ,
        +refbounds-last .default:o = \c_novalue_tl ,
        +refbounds-rb .default:o = \c_novalue_tl ,
        +refbounds-re .default:o = \c_novalue_tl ,
        +refbounds .default:o = \c_novalue_tl ,
        refbounds .default:o = \c_novalue_tl ,
      }
  }
\clist_map_inline:nn
  {
    langsetup ,
    langfile ,
  }
  {
    \keys_define:nn { zref-clever/ #1 }
      {
        +refbounds-first .value_required:n = true ,
        +refbounds-mid .value_required:n = true ,
        +refbounds-last .value_required:n = true ,
        +refbounds-rb .value_required:n = true ,
        +refbounds-re .value_required:n = true ,
        +refbounds .value_required:n = true ,
        refbounds .value_required:n = true ,
    }
  }
```

4.6 Languages

`\l__zrefclever_current_language_tl` is an internal alias for babel's `\languagename` or polyglossia's `\mainbabelname` 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@main@language` or for polyglossia's `\mainbabelname`, 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.

```
\tl_new:N \l__zrefclever_ref_language_tl
\tl_new:N \l__zrefclever_current_language_tl
\tl_new:N \l__zrefclever_main_language_tl
```

`\l_zrefclever_ref_language_tl` A public version of `\l__zrefclever_ref_language_tl` for use in zref-vario.

```
\tl_new:N \l_zrefclever_ref_language_tl
\tl_set:Nn \l_zrefclever_ref_language_tl { \l__zrefclever_ref_language_tl }
```

(*End definition for* `\l_zrefclever_ref_language_tl`*. This function is documented on page* **??**.)

`\__zrefclever_language_varname:n` Defines, and leaves in the input stream, the csname of the variable used to store the ⟨*base language*⟩ (as the value of this variable) for a ⟨*language*⟩ declared for zref-clever.

```
    \__zrefclever_language_varname:n {⟨language⟩}
```

```
\cs_new:Npn \__zrefclever_language_varname:n #1
  { g__zrefclever_declared_language_ #1 _tl }
```

(*End definition for* `\__zrefclever_language_varname:n`*.*)

`\zrefclever_language_varname:n` A public version of `\__zrefclever_language_varname:n` for use in zref-vario.

```
\cs_set_eq:NN \zrefclever_language_varname:n
  \__zrefclever_language_varname:n
```

(*End definition for* `\zrefclever_language_varname:n`*. This function is documented on page* **??**.)

`\__zrefclever_language_if_declared:nTF` A language is considered to be declared for zref-clever if it passes this conditional, which requires that a variable with `\__zrefclever_language_varname:n{⟨language⟩}` exists.

```
    \__zrefclever_language_if_declared:n(TF) {⟨language⟩}
```

```
\prg_new_conditional:Npnn \__zrefclever_language_if_declared:n #1 { T , F , TF }
  {
    \tl_if_exist:cTF { \__zrefclever_language_varname:n {#1} }
     { \prg_return_true:  }
     { \prg_return_false: }
  }
\prg_generate_conditional_variant:Nnn
  \__zrefclever_language_if_declared:n { x } { T , F , TF }
```

(*End definition for* `\__zrefclever_language_if_declared:nTF`*.*)

\zrefclever_language_if_declared:nTF A public version of `\__zrefclever_language_if_declared:n` for use in zref-vario.

```
\prg_set_eq_conditional:NNn \zrefclever_language_if_declared:n
  \__zrefclever_language_if_declared:n { TF }
```

(*End definition for* `\zrefclever_language_if_declared:nTF`. *This function is documented on page* ??.)

\zcDeclareLanguage Declare a new language for use with zref-clever. ⟨*language*⟩ is taken to be both the "language name" and the "base language name". 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". [⟨*options*⟩] receive a `k=v` set of options, with three valid options. The first, `declension`, takes the noun declension cases prefixes for ⟨*language*⟩ as a comma separated list, whose first element is taken to be the default case. The second, `gender`, receives the genders for ⟨*language*⟩ as comma separated list. The third, `allcaps`, is a boolean, and indicates that for ⟨*language*⟩ all nouns must be capitalized for grammatical reasons, in which case, the `cap` option is disregarded for ⟨*language*⟩. If ⟨*language*⟩ is already known, just warn. This implies a particular restriction regarding [⟨*options*⟩], 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 [⟨options⟩] {⟨language⟩}
```

```
\NewDocumentCommand \zcDeclareLanguage { O { } m }
  {
    \group_begin:
    \tl_if_empty:nF {#2}
      {
        \__zrefclever_language_if_declared:nTF {#2}
          { \msg_warning:nnn { zref-clever } { language-declared } {#2} }
          {
            \tl_new:c { \__zrefclever_language_varname:n {#2} }
            \tl_gset:cn { \__zrefclever_language_varname:n {#2} } {#2}
            \tl_set:Nn \l__zrefclever_setup_language_tl {#2}
            \keys_set:nn { zref-clever/declarelang } {#1}
          }
      }
    \group_end:
  }
\@onlypreamble \zcDeclareLanguage
```

(*End definition for* `\zcDeclareLanguage`.)

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

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

```
\NewDocumentCommand \zcDeclareLanguageAlias { m m }
  {
    \tl_if_empty:nF {#1}
      {
        \__zrefclever_language_if_declared:nTF {#2}
          {
            \tl_new:c { \__zrefclever_language_varname:n {#1} }
            \tl_gset:cx { \__zrefclever_language_varname:n {#1} }
              { \tl_use:c { \__zrefclever_language_varname:n {#2} } }
          }
          { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
      }
  }
\@onlypreamble \zcDeclareLanguageAlias
```

(*End definition for* `\zcDeclareLanguageAlias`*.*)

```
\keys_define:nn { zref-clever/declarelang }
  {
    declension .code:n =
      {
        \seq_new:c
          {
            \__zrefclever_opt_varname_language:enn
              { \l__zrefclever_setup_language_tl } { declension } { seq }
          }
        \seq_gset_from_clist:cn
          {
            \__zrefclever_opt_varname_language:enn
              { \l__zrefclever_setup_language_tl } { declension } { seq }
          }
          {#1}
      } ,
    declension .value_required:n = true ,
    gender .code:n =
      {
        \seq_new:c
          {
            \__zrefclever_opt_varname_language:enn
              { \l__zrefclever_setup_language_tl } { gender } { seq }
          }
        \seq_gset_from_clist:cn
          {
            \__zrefclever_opt_varname_language:enn
              { \l__zrefclever_setup_language_tl } { gender } { seq }
          }
          {#1}
      } ,
    gender .value_required:n = true ,
    allcaps .choices:nn =
      { true , false }
      {
        \bool_new:c
          {
            \__zrefclever_opt_varname_language:enn
              { \l__zrefclever_setup_language_tl } { allcaps } { bool }
          }
        \use:c { bool_gset_ \l_keys_choice_tl :c }
          {
            \__zrefclever_opt_varname_language:enn
              { \l__zrefclever_setup_language_tl } { allcaps } { bool }
          }
      } ,
    allcaps .default:n = true ,
  }
```

\__zrefclever_process_language_settings: Auxiliary function for `\__zrefclever_zcref:nnn`, responsible for processing language related settings. It is necessary to separate them from the reference options machinery for two reasons. First, because their behavior is language dependent, but the language itself can also be set as an option (`lang`, value stored in `\l__zrefclever_ref_language_tl`). Second, some of its tasks must be done regardless of any option being given (e.g. the default declension case, the `allcaps` option). Hence, we must validate the language settings after the reference options have been set. It is expected to be called right (or soon) after `\keys_set:nn` in `\__zrefclever_zcref:nnn`, where current values for `\l__zrefclever_ref_language_tl` and `\l__zrefclever_ref_decl_case_tl` are in place.

```
\cs_new_protected:Npn \__zrefclever_process_language_settings:
  {
    \__zrefclever_language_if_declared:xTF
      { \l__zrefclever_ref_language_tl }
      {
```

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 `\l__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.

```
        \__zrefclever_opt_seq_get:cNF
          {
            \__zrefclever_opt_varname_language:enn
              { \l__zrefclever_ref_language_tl } { declension } { seq }
          }
          \l__zrefclever_lang_declension_seq
          { \seq_clear:N \l__zrefclever_lang_declension_seq }
        \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
          {
            \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
              {
                \msg_warning:nnxx { zref-clever }
                  { language-no-decl-ref }
                  { \l__zrefclever_ref_language_tl }
                  { \l__zrefclever_ref_decl_case_tl }
                \tl_clear:N \l__zrefclever_ref_decl_case_tl
              }
          }
          {
            \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
              {
                \seq_get_left:NN \l__zrefclever_lang_declension_seq
                  \l__zrefclever_ref_decl_case_tl
              }
              {
                \seq_if_in:NVF \l__zrefclever_lang_declension_seq
```

```
                    \l__zrefclever_ref_decl_case_tl
                    {
                      \msg_warning:nnxx { zref-clever }
                        { unknown-decl-case }
                        { \l__zrefclever_ref_decl_case_tl }
                        { \l__zrefclever_ref_language_tl }
                      \seq_get_left:NN \l__zrefclever_lang_declension_seq
                        \l__zrefclever_ref_decl_case_tl
                    }
                }
            }
```

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.

```
        \__zrefclever_opt_seq_get:cNF
          {
            \__zrefclever_opt_varname_language:enn
              { \l__zrefclever_ref_language_tl } { gender } { seq }
          }
          \l__zrefclever_lang_gender_seq
          { \seq_clear:N \l__zrefclever_lang_gender_seq }
        \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
          {
            \tl_if_empty:NF \l__zrefclever_ref_gender_tl
              {
                \msg_warning:nnxxx { zref-clever }
                  { language-no-gender }
                  { \l__zrefclever_ref_language_tl }
                  { g }
                  { \l__zrefclever_ref_gender_tl }
                \tl_clear:N \l__zrefclever_ref_gender_tl
              }
          }
          {
            \tl_if_empty:NF \l__zrefclever_ref_gender_tl
              {
                \seq_if_in:NVF \l__zrefclever_lang_gender_seq
                  \l__zrefclever_ref_gender_tl
                  {
                    \msg_warning:nnxx { zref-clever }
                      { gender-not-declared }
                      { \l__zrefclever_ref_language_tl }
                      { \l__zrefclever_ref_gender_tl }
                    \tl_clear:N \l__zrefclever_ref_gender_tl
                  }
              }
          }
```

Ensure the general `cap` is set to `true` when the language was declared with `allcaps` option.

```
        \__zrefclever_opt_bool_if:cT
          {
            \__zrefclever_opt_varname_language:enn
              { \l__zrefclever_ref_language_tl } { allcaps } { bool }
          }
          { \keys_set:nn { zref-clever/reference } { 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 }
              { \l__zrefclever_ref_language_tl }
              { g }
              { \l__zrefclever_ref_gender_tl }
            \tl_clear:N \l__zrefclever_ref_gender_tl
          }
      }
  }
```

(*End definition for* `\__zrefclever_process_language_settings:`.)

4.7 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 corresponding variables. Hence, there is no need to "load" anything in this case: definitions and assignments made by the user are performed immediately.

\g__zrefclever_loaded_langfiles_seq

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

```
\seq_new:N \g__zrefclever_loaded_langfiles_seq
```

(*End definition for* `\g__zrefclever_loaded_langfiles_seq`.)

__zrefclever_provide_langfile:n

Load language file for known ⟨*language*⟩ if it is available and if it has not already been loaded.

```
\__zrefclever_provide_langfile:n {⟨language⟩}
```

```
\cs_new_protected:Npn \__zrefclever_provide_langfile:n #1
  {
    \group_begin:
    \@bsphack
    \__zrefclever_language_if_declared:nT {#1}
      {
        \seq_if_in:NxF
          \g__zrefclever_loaded_langfiles_seq
          { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
          {
            \exp_args:Nx \file_get:nnNTF
              {
                zref-clever-
                \tl_use:c { \__zrefclever_language_varname:n {#1} }
                .lang
              }
              { \ExplSyntaxOn }
              \l_tmpa_tl
              {
                \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
                \tl_clear:N \l__zrefclever_setup_type_tl
                \__zrefclever_opt_seq_get:cNF
                  {
                    \__zrefclever_opt_varname_language:nnn
                      {#1} { declension } { seq }
                  }
                  \l__zrefclever_lang_declension_seq
                  { \seq_clear:N \l__zrefclever_lang_declension_seq }
                \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
```

```
                  { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
                  {
                    \seq_get_left:NN \l__zrefclever_lang_declension_seq
                      \l__zrefclever_lang_decl_case_tl
                  }
                \__zrefclever_opt_seq_get:cNF
                  {
                    \__zrefclever_opt_varname_language:nnn
                      {#1} { gender } { seq }
                  }
                  \l__zrefclever_lang_gender_seq
                  { \seq_clear:N \l__zrefclever_lang_gender_seq }
                \keys_set:nV { zref-clever/langfile } \l_tmpa_tl
                \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
                  { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
                \msg_info:nnx { zref-clever } { langfile-loaded }
                  { \tl_use:c { \__zrefclever_language_varname: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, if it was not found the first time, it won't be the next.

```
                \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
                  { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
              }
          }
      }
    \@esphack
    \group_end:
  }
\cs_generate_variant:Nn \__zrefclever_provide_langfile:n { x }
```

(*End definition for* `\__zrefclever_provide_langfile:n`.)

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:nnxx { zref-clever } { language-no-decl-setup }
              { \l__zrefclever_setup_language_tl } {#1}
          }
          {
            \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
              { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
              {
                \msg_info:nnxx { zref-clever } { unknown-decl-case }
                  {#1} { \l__zrefclever_setup_language_tl }
                \seq_get_left:NN \l__zrefclever_lang_declension_seq
                  \l__zrefclever_lang_decl_case_tl
              }
          }
      } ,
    case .value_required:n = true ,

    gender .value_required:n = true ,
    gender .code:n =
      {
        \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
          {
            \msg_info:nnxxx { zref-clever } { language-no-gender }
              { \l__zrefclever_setup_language_tl } { gender } {#1}
          }
          {
            \tl_if_empty:NTF \l__zrefclever_setup_type_tl
              {
                \msg_info:nnn { zref-clever }
                  { option-only-type-specific } { gender }
              }
              {
                \seq_clear:N \l_tmpa_seq
                \clist_map_inline:nn {#1}
                  {
                    \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
                      { \seq_put_right:Nn \l_tmpa_seq {##1} }
                      {
                        \msg_info:nnxx { zref-clever }
                          { gender-not-declared }
                          { \l__zrefclever_setup_language_tl } {##1}
                      }
                  }
                \__zrefclever_opt_seq_if_set:cF
                  {
                    \__zrefclever_opt_varname_lang_type:eenn
                      { \l__zrefclever_setup_language_tl }
                      { \l__zrefclever_setup_type_tl }
                      { gender }
                      { seq }
                  }
                  {
                    \seq_new:c
                      {
                        \__zrefclever_opt_varname_lang_type:eenn
                          { \l__zrefclever_setup_language_tl }
                          { \l__zrefclever_setup_type_tl }
                          { gender }
                          { seq }
                      }
                    \seq_gset_eq:cN
                      {
                        \__zrefclever_opt_varname_lang_type:eenn
                          { \l__zrefclever_setup_language_tl }
                          { \l__zrefclever_setup_type_tl }
                          { gender }
                          { seq }
                      }
                      \l_tmpa_seq
                  }
              }
          }
      } ,
  }
\seq_map_inline:Nn
  \g__zrefclever_rf_opts_tl_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_opt_tl_gset_if_new:cn
                  {
                    \__zrefclever_opt_varname_lang_default:enn
                      { \l__zrefclever_setup_language_tl }
                      {#1} { tl }
                  }
                  {##1}
              }
              {
                \msg_info:nnn { zref-clever }
                  { option-not-type-specific } {#1}
              }
          } ,
      }
  }
\seq_map_inline:Nn
  \g__zrefclever_rf_opts_tl_maybe_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_opt_tl_gset_if_new:cn
                  {
                    \__zrefclever_opt_varname_lang_default:enn
                      { \l__zrefclever_setup_language_tl }
                      {#1} { tl }
                  }
                  {##1}
              }
              {
                \__zrefclever_opt_tl_gset_if_new:cn
                  {
                    \__zrefclever_opt_varname_lang_type:eenn
                      { \l__zrefclever_setup_language_tl }
                      { \l__zrefclever_setup_type_tl }
                      {#1} { tl }
                  }
                  {##1}
              }
          } ,
      }
  }
\keys_define:nn { zref-clever/langfile }
  {
    endrange .value_required:n = true ,
    endrange .code:n =
      {
        \str_case:nnF {#1}
          {
            { ref }
            {
              \tl_if_empty:NTF \l__zrefclever_setup_type_tl
                {
                  \__zrefclever_opt_tl_gclear_if_new:c
                    {
                      \__zrefclever_opt_varname_lang_default:enn
                        { \l__zrefclever_setup_language_tl }
                        { endrangefunc } { tl }
                    }
                  \__zrefclever_opt_tl_gclear_if_new:c
                    {
                      \__zrefclever_opt_varname_lang_default:enn
                        { \l__zrefclever_setup_language_tl }
                        { endrangeprop } { tl }
                    }
                }
                {
                  \__zrefclever_opt_tl_gclear_if_new:c
                    {
                      \__zrefclever_opt_varname_lang_type:eenn
                        { \l__zrefclever_setup_language_tl }
                        { \l__zrefclever_setup_type_tl }
                        { endrangefunc } { tl }
                    }
                  \__zrefclever_opt_tl_gclear_if_new:c
                    {
                      \__zrefclever_opt_varname_lang_type:eenn
                        { \l__zrefclever_setup_language_tl }
                        { \l__zrefclever_setup_type_tl }
                        { endrangeprop } { tl }
                    }
                }
            }

            { stripprefix }
            {
              \tl_if_empty:NTF \l__zrefclever_setup_type_tl
                {
                  \__zrefclever_opt_tl_gset_if_new:cn
                    {
                      \__zrefclever_opt_varname_lang_default:enn
                        { \l__zrefclever_setup_language_tl }
                        { endrangefunc } { tl }
                    }
                    { __zrefclever_get_endrange_stripprefix }
                  \__zrefclever_opt_tl_gclear_if_new:c
                    {
                      \__zrefclever_opt_varname_lang_default:enn
                        { \l__zrefclever_setup_language_tl }
                        { endrangeprop } { tl }
                    }
                }
                {
                  \__zrefclever_opt_tl_gset_if_new:cn
                    {
                      \__zrefclever_opt_varname_lang_type:eenn
                        { \l__zrefclever_setup_language_tl }
                        { \l__zrefclever_setup_type_tl }
                        { endrangefunc } { tl }
                    }
                    { __zrefclever_get_endrange_stripprefix }
                  \__zrefclever_opt_tl_gclear_if_new:c
                    {
                      \__zrefclever_opt_varname_lang_type:eenn
                        { \l__zrefclever_setup_language_tl }
                        { \l__zrefclever_setup_type_tl }
                        { endrangeprop } { tl }
                    }
                }
            }

            { pagecomp }
            {
              \tl_if_empty:NTF \l__zrefclever_setup_type_tl
                {
                  \__zrefclever_opt_tl_gset_if_new:cn
                    {
                      \__zrefclever_opt_varname_lang_default:enn
                        { \l__zrefclever_setup_language_tl }
                        { endrangefunc } { tl }
                    }
                    { __zrefclever_get_endrange_pagecomp }
                  \__zrefclever_opt_tl_gclear_if_new:c
                    {
                      \__zrefclever_opt_varname_lang_default:enn
                        { \l__zrefclever_setup_language_tl }
                        { endrangeprop } { tl }
                    }
                }
                {
                  \__zrefclever_opt_tl_gset_if_new:cn
                    {
                      \__zrefclever_opt_varname_lang_type:eenn
                        { \l__zrefclever_setup_language_tl }
                        { \l__zrefclever_setup_type_tl }
                        { endrangefunc } { tl }
                    }
                    { __zrefclever_get_endrange_pagecomp }
                  \__zrefclever_opt_tl_gclear_if_new:c
                    {
                      \__zrefclever_opt_varname_lang_type:eenn
                        { \l__zrefclever_setup_language_tl }
                        { \l__zrefclever_setup_type_tl }
                        { endrangeprop } { tl }
                    }
                }
            }

            { pagecomp2 }
            {
              \tl_if_empty:NTF \l__zrefclever_setup_type_tl
                {
                  \__zrefclever_opt_tl_gset_if_new:cn
                    {
                      \__zrefclever_opt_varname_lang_default:enn
                        { \l__zrefclever_setup_language_tl }
                        { endrangefunc } { tl }
                    }
                    { __zrefclever_get_endrange_pagecomptwo }
                  \__zrefclever_opt_tl_gclear_if_new:c
                    {
                      \__zrefclever_opt_varname_lang_default:enn
                        { \l__zrefclever_setup_language_tl }
                        { endrangeprop } { tl }
                    }
                }
                {
                  \__zrefclever_opt_tl_gset_if_new:cn
                    {
                      \__zrefclever_opt_varname_lang_type:eenn
                        { \l__zrefclever_setup_language_tl }
                        { \l__zrefclever_setup_type_tl }
                        { endrangefunc } { tl }
                    }
                    { __zrefclever_get_endrange_pagecomptwo }
                \__zrefclever_opt_tl_gclear_if_new:c
                 {
                  \__zrefclever_opt_varname_lang_type:eenn
                   { \l__zrefclever_setup_language_tl }
                   { \l__zrefclever_setup_type_tl }
                   { endrangeprop } { tl }
                 }
               }
             }
            }
            {
             \tl_if_empty:nTF {#1}
              {
               \msg_info:nnn { zref-clever }
                { endrange-property-undefined } {#1}
              }
              {
               \zref@ifpropundefined {#1}
                {
                 \msg_info:nnn { zref-clever }
                  { endrange-property-undefined } {#1}
                }
                {
                 \tl_if_empty:NTF \l__zrefclever_setup_type_tl
                  {
                   \__zrefclever_opt_tl_gset_if_new:cn
                    {
                     \__zrefclever_opt_varname_lang_default:enn
                      { \l__zrefclever_setup_language_tl }
                      { endrangefunc } { tl }
                    }
                    { __zrefclever_get_endrange_property }
                   \__zrefclever_opt_tl_gset_if_new:cn
                    {
                     \__zrefclever_opt_varname_lang_default:enn
                      { \l__zrefclever_setup_language_tl }
                      { endrangeprop } { tl }
                    }
                    {#1}
                  }
                  {
                   \__zrefclever_opt_tl_gset_if_new:cn
                    {
                     \__zrefclever_opt_varname_lang_type:eenn
                      { \l__zrefclever_setup_language_tl }
                      { \l__zrefclever_setup_type_tl }
                      { endrangefunc } { tl }
                    }
                    { __zrefclever_get_endrange_property }
                   \__zrefclever_opt_tl_gset_if_new:cn
                    {
                     \__zrefclever_opt_varname_lang_type:eenn
                      { \l__zrefclever_setup_language_tl }
                      { \l__zrefclever_setup_type_tl }
```

```
                              { endrangeprop } { tl }
                          }
                          {#1}
                      }
                  }
              }
          }
      } ,
  }
\seq_map_inline:Nn
  \g__zrefclever_rf_opts_tl_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:nnn { zref-clever }
                  { option-only-type-specific } {#1}
              }
              {
                \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
                  {
                    \__zrefclever_opt_tl_gset_if_new:cn
                      {
                        \__zrefclever_opt_varname_lang_type:eenn
                          { \l__zrefclever_setup_language_tl }
                          { \l__zrefclever_setup_type_tl }
                          {#1} { tl }
                      }
                      {##1}
                  }
                  {
                    \__zrefclever_opt_tl_gset_if_new:cn
                      {
                        \__zrefclever_opt_varname_lang_type:eeen
                          { \l__zrefclever_setup_language_tl }
                          { \l__zrefclever_setup_type_tl }
                          { \l__zrefclever_lang_decl_case_tl - #1 } { tl }
                      }
                      {##1}
                  }
              }
          } ,
      }
  }
\seq_map_inline:Nn
  \g__zrefclever_rf_opts_seq_refbounds_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_opt_seq_if_set:cF
                  {
                    \__zrefclever_opt_varname_lang_default:enn
                      { \l__zrefclever_setup_language_tl } {#1} { seq }
                  }
                  {
                    \seq_gclear:N \g_tmpa_seq
                    \__zrefclever_opt_seq_gset_clist_split:Nn
                      \g_tmpa_seq {##1}
                    \bool_lazy_or:nnTF
                      { \tl_if_empty_p:n {##1} }
                      {
                        \int_compare_p:nNn
                          { \seq_count:N \g_tmpa_seq } = { 4 }
                      }
                      {
                        \__zrefclever_opt_seq_gset_eq:cN
                          {
                            \__zrefclever_opt_varname_lang_default:enn
                              { \l__zrefclever_setup_language_tl }
                              {#1} { seq }
                          }
                          \g_tmpa_seq
                      }
                      {
                        \msg_info:nnxx { zref-clever }
                          { refbounds-must-be-four }
                          {#1} { \seq_count:N \g_tmpa_seq }
                      }
                  }
              }
              {
                \__zrefclever_opt_seq_if_set:cF
                  {
                    \__zrefclever_opt_varname_lang_type:eenn
                      { \l__zrefclever_setup_language_tl }
                      { \l__zrefclever_setup_type_tl } {#1} { seq }
                  }
                  {
                    \seq_gclear:N \g_tmpa_seq
                    \__zrefclever_opt_seq_gset_clist_split:Nn
                      \g_tmpa_seq {##1}
                    \bool_lazy_or:nnTF
                      { \tl_if_empty_p:n {##1} }
                      {
                        \int_compare_p:nNn
                          { \seq_count:N \g_tmpa_seq } = { 4 }
                      }
                      {
                        \__zrefclever_opt_seq_gset_eq:cN
```

```
                        {
                          \__zrefclever_opt_varname_lang_type:eenn
                            { \l__zrefclever_setup_language_tl }
                            { \l__zrefclever_setup_type_tl }
                            {#1} { seq }
                        }
                        \g_tmpa_seq
                    }
                    {
                      \msg_info:nnxx { zref-clever }
                        { refbounds-must-be-four }
                        {#1} { \seq_count:N \g_tmpa_seq }
                    }
                }
            }
        } ,
    }
  }
\seq_map_inline:Nn
  \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
  {
    \keys_define:nn { zref-clever/langfile }
      {
        #1 .choice: ,
        #1 / true .code:n =
          {
            \tl_if_empty:NTF \l__zrefclever_setup_type_tl
              {
                \__zrefclever_opt_bool_if_set:cF
                  {
                    \__zrefclever_opt_varname_lang_default:enn
                      { \l__zrefclever_setup_language_tl }
                      {#1} { bool }
                  }
                  {
                    \__zrefclever_opt_bool_gset_true:c
                      {
                        \__zrefclever_opt_varname_lang_default:enn
                          { \l__zrefclever_setup_language_tl }
                          {#1} { bool }
                      }
                  }
              }
              {
                \__zrefclever_opt_bool_if_set:cF
                  {
                    \__zrefclever_opt_varname_lang_type:eenn
                      { \l__zrefclever_setup_language_tl }
                      { \l__zrefclever_setup_type_tl }
                      {#1} { bool }
                  }
                  {
                    \__zrefclever_opt_bool_gset_true:c
                      {
```

```
           \__zrefclever_opt_varname_lang_type:eenn
            { \l__zrefclever_setup_language_tl }
            { \l__zrefclever_setup_type_tl }
            {#1} { bool }
          }
        }
      }
    } ,
   #1 / false .code:n =
    {
     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
      {
       \__zrefclever_opt_bool_if_set:cF
        {
         \__zrefclever_opt_varname_lang_default:enn
          { \l__zrefclever_setup_language_tl }
          {#1} { bool }
        }
        {
         \__zrefclever_opt_bool_gset_false:c
          {
           \__zrefclever_opt_varname_lang_default:enn
            { \l__zrefclever_setup_language_tl }
            {#1} { bool }
          }
        }
      }
      {
       \__zrefclever_opt_bool_if_set:cF
        {
         \__zrefclever_opt_varname_lang_type:eenn
          { \l__zrefclever_setup_language_tl }
          { \l__zrefclever_setup_type_tl }
          {#1} { bool }
        }
        {
         \__zrefclever_opt_bool_gset_false:c
          {
           \__zrefclever_opt_varname_lang_type:eenn
            { \l__zrefclever_setup_language_tl }
            { \l__zrefclever_setup_type_tl }
            {#1} { bool }
          }
        }
      }
    } ,
   #1 .default:n = true ,
   no #1 .meta:n = { #1 = false } ,
   no #1 .value_forbidden:n = true ,
  }
}
```

It is convenient for a number of language typesetting options (some basic separators) to have some "fallback" value available in case 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". Other typesetting options, for which it is not a problem being empty, need not be catered for with a fallback value.

```
\cs_new_protected:Npn \__zrefclever_opt_tl_cset_fallback:nn #1#2
  {
    \tl_const:cn
      { \__zrefclever_opt_varname_fallback:nn {#1} { tl } } {#2}
  }
\keyval_parse:nnn
  { }
  { \__zrefclever_opt_tl_cset_fallback:nn }
  {
    tpairsep  = {,~} ,
    tlistsep  = {,~} ,
    tlastsep  = {,~} ,
    notesep   = {~} ,
    namesep   = {\nobreakspace} ,
    pairsep   = {,~} ,
    listsep   = {,~} ,
    lastsep   = {,~} ,
    rangesep  = {\textendash} ,
  }
```

4.8 Options

Auxiliary

__zrefclever_prop_put_non_empty:Nnn If ⟨*value*⟩ is empty, remove ⟨*key*⟩ from ⟨*property list*⟩. Otherwise, add ⟨*key*⟩ = ⟨*value*⟩ to ⟨*property list*⟩.

```
\__zrefclever_prop_put_non_empty:Nnn ⟨property list⟩ {⟨key⟩} {⟨value⟩}
```

```
\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@ifrefcontainsprop`, 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@ifpropundefined`: close it at the door. We must also control for an empty value, since "empty" passes both `\zref@ifpropundefined` and `\zref@ifrefcontainsprop`.

```
\tl_new:N \l__zrefclever_ref_property_tl
\keys_define:nn { zref-clever/reference }
  {
    ref .code:n =
      {
        \tl_if_empty:nTF {#1}
          {
            \msg_warning:nnn { zref-clever }
              { zref-property-undefined } {#1}
            \tl_set:Nn \l__zrefclever_ref_property_tl { default }
          }
          {
            \zref@ifpropundefined {#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 =
      {
        \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 ,
  }
```

sort option

```
\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 ,
  }
```

typesort option

`\l__zrefclever_typesort_seq` is stored reversed, since the sort priorities are computed in the negative range in `\__zrefclever_sort_default_different_types:nn`, so that we can implicitly rely on '0' being the "last value", and spare creating an integer variable using `\seq_map_indexed_inline:Nn`.

```
\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
      } ,
    typesort .initial:n =
      { part , chapter , section , paragraph },
    typesort .value_required:n = true ,
    notypesort .code:n =
      { \seq_clear:N \l__zrefclever_typesort_seq } ,
    notypesort .value_forbidden:n = true ,
  }
```

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 ,
  }
```

endrange option

The working of `endrange` option depends on two underlying option values / variables: `endrangefunc` and `endrangeprop`. `endrangefunc` is the more general one, and `endrangeprop` is used when the first is set to `\__zrefclever_get_endrange_-property:VVN`, which is the case when the user is setting `endrange` to an arbitrary zref property, instead of one of the `\str_case:nn` matches.

`endrangefunc` *must* receive three arguments and, more specifically, its signature *must* be `VVN`. For this reason, `endrangefunc` should be stored without the signature, which is added, and hard-coded, at the calling place. The first argument is ⟨*beg range label*⟩, the second ⟨*end range label*⟩, and the last ⟨*tl var to set*⟩. Of course, ⟨*tl var to set*⟩ must be set to a proper value, and that's the main task of the function. `endrangefunc` must also handle the case where `\zref@ifrefcontainsprop` is false, since `\__zrefclever_get_ref_endrange:nnN` cannot take care of that. For this purpose, it may set ⟨*tl var to set*⟩ to the special value `zc@missingproperty`, to signal a missing property for `\__zrefclever_get_ref_endrange:nnN`.

An empty `endrangefunc` signals that no processing is to be made to the end range reference, that is, that it should be treated like any other one, as defined by the `ref` option. This may happen either because `endrange` was never set for the reference type, and empty is the value "returned" by `\__zrefclever_get_rf_opt_tl:nnnN` for options not set, or because `endrange` was set to `ref` at some scope which happens to get precedence.

One thing I was divided about in this functionality was whether to (x-)expand the references before processing them, when such processing is required. At first sight, it makes sense to do so, since we are aiming at "removing common parts" as close as possible to the printed representation of the references (cleveref does expand them in `\crefstripprefix`). On the other hand, this brings some new challenges: if a fragile command gets there, we are in trouble; also, if a protected one gets there, though things won't break as badly, we may "strip" the macro and stay with different arguments, which will then end up in the input stream. I think biblatex is a good reference here, and it offers `\NumCheckSetup`, `\NumsCheckSetup`, and `\PagesCheckSetup` aimed at locally redefining some commands which may interfere with the processing. This is a good idea, thus we offer a similar hook for the same purpose: `endrange-setup`.

```
\NewHook { zref-clever/endrange-setup }
```

```
\keys_define:nn { zref-clever/reference }
  {
    endrange .code:n =
      {
        \str_case:nnF {#1}
          {
            { ref }
            {
              \__zrefclever_opt_tl_clear:c
                {
                  \__zrefclever_opt_varname_general:nn
                    { endrangefunc } { tl }
                }
              \__zrefclever_opt_tl_clear:c
                {
                  \__zrefclever_opt_varname_general:nn
                    { endrangeprop } { tl }
                }
            }

            { stripprefix }
            {
```

```
          \__zrefclever_opt_tl_set:cn
            {
              \__zrefclever_opt_varname_general:nn
                { endrangefunc } { tl }
            }
            { __zrefclever_get_endrange_stripprefix }
          \__zrefclever_opt_tl_clear:c
            {
              \__zrefclever_opt_varname_general:nn
                { endrangeprop } { tl }
            }
        }

        { pagecomp }
        {
          \__zrefclever_opt_tl_set:cn
            {
              \__zrefclever_opt_varname_general:nn
                { endrangefunc } { tl }
            }
            { __zrefclever_get_endrange_pagecomp }
          \__zrefclever_opt_tl_clear:c
            {
              \__zrefclever_opt_varname_general:nn
                { endrangeprop } { tl }
            }
        }

        { pagecomp2 }
        {
          \__zrefclever_opt_tl_set:cn
            {
              \__zrefclever_opt_varname_general:nn
                { endrangefunc } { tl }
            }
            { __zrefclever_get_endrange_pagecomptwo }
          \__zrefclever_opt_tl_clear:c
            {
              \__zrefclever_opt_varname_general:nn
                { endrangeprop } { tl }
            }
        }

        { unset }
        {
          \__zrefclever_opt_tl_unset:c
            {
              \__zrefclever_opt_varname_general:nn
                { endrangefunc } { tl }
            }
          \__zrefclever_opt_tl_unset:c
            {
              \__zrefclever_opt_varname_general:nn
                { endrangeprop } { tl }
                }
            }
          }
          {
            \tl_if_empty:nTF {#1}
              {
                \msg_warning:nnn { zref-clever }
                  { endrange-property-undefined } {#1}
              }
              {
                \zref@ifpropundefined {#1}
                  {
                    \msg_warning:nnn { zref-clever }
                      { endrange-property-undefined } {#1}
                  }
                  {
                    \__zrefclever_opt_tl_set:cn
                      {
                        \__zrefclever_opt_varname_general:nn
                          { endrangefunc } { tl }
                      }
                      { __zrefclever_get_endrange_property }
                    \__zrefclever_opt_tl_set:cn
                      {
                        \__zrefclever_opt_varname_general:nn
                          { endrangeprop } { tl }
                      }
                      {#1}
                  }
              }
          }
      } ,
    endrange .value_required:n = true ,
  }
\cs_new_protected:Npn \__zrefclever_get_endrange_property:nnN #1#2#3
  {
    \tl_if_empty:NTF \l__zrefclever_endrangeprop_tl
      {
        \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
          {
            \__zrefclever_extract_default:Nnvn #3
              {#2} { l__zrefclever_ref_property_tl } { }
          }
          { \tl_set:Nn #3 { zc@missingproperty } }
      }
      {
        \zref@ifrefcontainsprop {#2} { \l__zrefclever_endrangeprop_tl }
          {
```

If the range came about by normal compression, we already know the beginning and the end references share the same "form" and "prefix" (this is ensured at `\__zrefclever_-labels_in_sequence:nn`), but the same is not true if the `range` option is being used, in which case, we have to check the replacement `\l__zrefclever_ref_property_tl` by `\l__zrefclever_endrangeprop_tl` is really granted.

```
              \bool_if:NTF \l__zrefclever_typeset_range_bool
                {
                  \group_begin:
                  \bool_set_false:N \l_tmpa_bool
                  \exp_args:Nxx \tl_if_eq:nnT
                    {
                      \__zrefclever_extract_unexp:nnn
                        {#1} { externaldocument } { }
                    }
                    {
                      \__zrefclever_extract_unexp:nnn
                        {#2} { externaldocument } { }
                    }
                    {
                      \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
                        {
                          \exp_args:Nxx \tl_if_eq:nnT
                            {
                              \__zrefclever_extract_unexp:nnn
                                {#1} { zc@pgfmt } { }
                            }
                            {
                              \__zrefclever_extract_unexp:nnn
                                {#2} { zc@pgfmt } { }
                            }
                            { \bool_set_true:N \l_tmpa_bool }
                        }
                        {
                          \exp_args:Nxx \tl_if_eq:nnT
                            {
                              \__zrefclever_extract_unexp:nnn
                                {#1} { zc@counter } { }
                            }
                            {
                              \__zrefclever_extract_unexp:nnn
                                {#2} { zc@counter } { }
                            }
                            {
                              \exp_args:Nxx \tl_if_eq:nnT
                                {
                                  \__zrefclever_extract_unexp:nnn
                                    {#1} { zc@enclval } { }
                                }
                                {
                                  \__zrefclever_extract_unexp:nnn
                                    {#2} { zc@enclval } { }
                                }
                                { \bool_set_true:N \l_tmpa_bool }
                            }
                        }
                    }
                  \bool_if:NTF \l_tmpa_bool
                    {
                      \__zrefclever_extract_default:Nnvn \l_tmpb_tl
                        {#2} { l__zrefclever_endrangeprop_tl } { }
                    }
                    {
                      \zref@ifrefcontainsprop
                        {#2} { \l__zrefclever_ref_property_tl }
                        {
                          \__zrefclever_extract_default:Nnvn \l_tmpb_tl
                            {#2} { l__zrefclever_ref_property_tl } { }
                        }
                        { \tl_set:Nn \l_tmpb_tl { zc@missingproperty } }
                    }
                  \exp_args:NNNV
                    \group_end:
                    \tl_set:Nn #3 \l_tmpb_tl
                }
                {
                  \__zrefclever_extract_default:Nnvn #3
                    {#2} { l__zrefclever_endrangeprop_tl } { }
                }
            }
            {
              \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
                {
                  \__zrefclever_extract_default:Nnvn #3
                    {#2} { l__zrefclever_ref_property_tl } { }
                }
                { \tl_set:Nn #3 { zc@missingproperty } }
            }
        }
    }
  \cs_generate_variant:Nn \__zrefclever_get_endrange_property:nnN { VVN }
```

For the technique for smuggling the assignment out of the group, see Enrico Gregorio's answer at https://tex.stackexchange.com/a/56314.

```
\cs_new_protected:Npn \__zrefclever_get_endrange_stripprefix:nnN #1#2#3
  {
    \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
      {
        \group_begin:
        \UseHook { zref-clever/endrange-setup }
        \tl_set:Nx \l_tmpa_tl
          {
            \__zrefclever_extract:nnn
              {#1} { \l__zrefclever_ref_property_tl } { }
          }
        \tl_set:Nx \l_tmpb_tl
          {
            \__zrefclever_extract:nnn
              {#2} { \l__zrefclever_ref_property_tl } { }
          }
        \bool_set_false:N \l_tmpa_bool
        \bool_until_do:Nn \l_tmpa_bool
          {
            \exp_args:Nxx \tl_if_eq:nnTF
              { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
              {
                \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
                \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
                \tl_if_empty:NT \l_tmpb_tl
                  { \bool_set_true:N \l_tmpa_bool }
              }
              { \bool_set_true:N \l_tmpa_bool }
          }
        \exp_args:NNNV
          \group_end:
          \tl_set:Nn #3 \l_tmpb_tl
      }
      { \tl_set:Nn #3 { zc@missingproperty } }
    }
\cs_generate_variant:Nn \__zrefclever_get_endrange_stripprefix:nnN { VVN }
```

__zrefclever_is_integer_rgx:n Test if argument is composed only of digits (adapted from https://tex.stackexchange.com/a/427559).

```
\prg_new_protected_conditional:Npnn
  \__zrefclever_is_integer_rgx:n #1 { F , TF }
  {
    \regex_match:nnTF { \A\d+\Z } {#1}
      { \prg_return_true:  }
      { \prg_return_false: }
  }
\prg_generate_conditional_variant:Nnn
  \__zrefclever_is_integer_rgx:n { V } { F , TF }
```

(*End definition for __zrefclever_is_integer_rgx:n.*)

```
\cs_new_protected:Npn \__zrefclever_get_endrange_pagecomp:nnN #1#2#3
  {
    \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
      {
        \group_begin:
        \UseHook { zref-clever/endrange-setup }
        \tl_set:Nx \l_tmpa_tl
          {
            \__zrefclever_extract:nnn
              {#1} { \l__zrefclever_ref_property_tl } { }
          }
        \tl_set:Nx \l_tmpb_tl
          {
            \__zrefclever_extract:nnn
              {#2} { \l__zrefclever_ref_property_tl } { }
          }
        \bool_set_false:N \l_tmpa_bool
        \__zrefclever_is_integer_rgx:VTF \l_tmpa_tl
          {
            \__zrefclever_is_integer_rgx:VF \l_tmpb_tl
              { \bool_set_true:N \l_tmpa_bool }
          }
          { \bool_set_true:N \l_tmpa_bool }
        \bool_until_do:Nn \l_tmpa_bool
```

```
          {
            \exp_args:Nxx \tl_if_eq:nnTF
              { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
              {
                \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
                \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
                \tl_if_empty:NT \l_tmpb_tl
                  { \bool_set_true:N \l_tmpa_bool }
              }
              { \bool_set_true:N \l_tmpa_bool }
          }
        \exp_args:NNNV
          \group_end:
          \tl_set:Nn #3 \l_tmpb_tl
      }
      { \tl_set:Nn #3 { zc@missingproperty } }
  }
\cs_generate_variant:Nn \__zrefclever_get_endrange_pagecomp:nnN { VVN }
\cs_new_protected:Npn \__zrefclever_get_endrange_pagecomptwo:nnN #1#2#3
  {
    \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
      {
        \group_begin:
        \UseHook { zref-clever/endrange-setup }
        \tl_set:Nx \l_tmpa_tl
          {
            \__zrefclever_extract:nnn
              {#1} { \l__zrefclever_ref_property_tl } { }
          }
        \tl_set:Nx \l_tmpb_tl
          {
            \__zrefclever_extract:nnn
              {#2} { \l__zrefclever_ref_property_tl } { }
          }
        \bool_set_false:N \l_tmpa_bool
        \__zrefclever_is_integer_rgx:VTF \l_tmpa_tl
          {
            \__zrefclever_is_integer_rgx:VF \l_tmpb_tl
              { \bool_set_true:N \l_tmpa_bool }
          }
          { \bool_set_true:N \l_tmpa_bool }
        \bool_until_do:Nn \l_tmpa_bool
          {
            \exp_args:Nxx \tl_if_eq:nnTF
              { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
              {
                \bool_lazy_or:nnTF
                  { \int_compare_p:nNn { \l_tmpb_tl } > { 99 } }
                  { \int_compare_p:nNn { \tl_head:V \l_tmpb_tl } = { 0 } }
                  {
                    \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
                    \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
                  }
                  { \bool_set_true:N \l_tmpa_bool }
              }
              { \bool_set_true:N \l_tmpa_bool }
          }
        \exp_args:NNNV
          \group_end:
          \tl_set:Nn #3 \l_tmpb_tl
      }
      { \tl_set:Nn #3 { zc@missingproperty } }
  }
\cs_generate_variant:Nn \__zrefclever_get_endrange_pagecomptwo:nnN { VVN }
```

`range` and `rangetopair` options

The `rangetopair` option is being handled with other reference format option booleans at `\g__zrefclever_rf_opts_bool_maybe_type_specific_seq`.

```
\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

The `cap` option is currently being handled with other reference format option booleans at `\g__zrefclever_rf_opts_bool_maybe_type_specific_seq`.

```
\bool_new:N \l__zrefclever_capfirst_bool
\keys_define:nn { zref-clever/reference }
  {
    capfirst .bool_set:N = \l__zrefclever_capfirst_bool ,
    capfirst .initial:n = false ,
    capfirst .default:n = true ,
  }
```

`abbrev` and `noabbrevfirst` options

The `abbrev` option is currently being handled with other reference format option booleans at `\g__zrefclever_rf_opts_bool_maybe_type_specific_seq`.

```
\bool_new:N \l__zrefclever_noabbrev_first_bool
\keys_define:nn { zref-clever/reference }
  {
    noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
    noabbrevfirst .initial:n = false ,
    noabbrevfirst .default:n = true ,
  }
```

`S` option

```
\keys_define:nn { zref-clever/reference }
  {
    S .meta:n =
      { capfirst = {#1} , noabbrevfirst = {#1} },
    S .default:n = true ,
  }
```

hyperref option

```
\bool_new:N \l__zrefclever_hyperlink_bool
\bool_new:N \l__zrefclever_hyperref_warn_bool
\keys_define:nn { zref-clever/reference }
  {
    hyperref .choice: ,
    hyperref / auto .code:n =
      {
        \bool_set_true:N \l__zrefclever_hyperlink_bool
        \bool_set_false:N \l__zrefclever_hyperref_warn_bool
      } ,
    hyperref / true .code:n =
      {
        \bool_set_true:N \l__zrefclever_hyperlink_bool
        \bool_set_true:N \l__zrefclever_hyperref_warn_bool
      } ,
    hyperref / false .code:n =
      {
        \bool_set_false:N \l__zrefclever_hyperlink_bool
        \bool_set_false:N \l__zrefclever_hyperref_warn_bool
      } ,
    hyperref .initial:n = auto ,
    hyperref .default:n = true ,
```

nohyperref is provided mainly as a means to inhibit hyperlinking locally in zref-vario's commands without the need to be setting zref-clever's internal variables directly. What limits setting hyperref out of the preamble is that enabling hyperlinks requires loading packages. But nohyperref can only disable them, so we can use it in the document body too.

```
    nohyperref .meta:n = { hyperref = false } ,
    nohyperref .value_forbidden:n = true ,
  }
\AddToHook { begindocument }
  {
    \__zrefclever_if_package_loaded:nTF { hyperref }
      {
        \bool_if:NT \l__zrefclever_hyperlink_bool
          { \RequirePackage { zref-hyperref } }
      }
      {
        \bool_if:NT \l__zrefclever_hyperref_warn_bool
          { \msg_warning:nn { zref-clever } { missing-hyperref } }
        \bool_set_false:N \l__zrefclever_hyperlink_bool
      }
    \keys_define:nn { zref-clever/reference }
      {
        hyperref .code:n =
          { \msg_warning:nn { zref-clever } { hyperref-preamble-only } } ,
        nohyperref .code:n =
          { \bool_set_false:N \l__zrefclever_hyperlink_bool } ,
    }
  }
```

nameinlink option

```
\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 ,
  }
```

preposinlink option (deprecated)

```
\keys_define:nn { zref-clever/reference }
  {
    preposinlink .code:n =
      {
        % NOTE Option deprecated in 2022-01-12 for v0.2.0-alpha.
        \msg_warning:nnnn { zref-clever }{ option-deprecated }
          { preposinlink } { refbounds }
      } ,
  }
```

lang option

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`.

```
\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_current_language_tl { english }
            \tl_set:Nn \l__zrefclever_main_language_tl { english }
          }
      }
  }
\keys_define:nn { zref-clever/reference }
  {
    lang .code:n =
      {
        \AddToHook { begindocument }
          {
            \str_case:nnF {#1}
              {
                { current }
                {
                  \tl_set:Nn \l__zrefclever_ref_language_tl
                    { \l__zrefclever_current_language_tl }
                }

                { main }
                {
                  \tl_set:Nn \l__zrefclever_ref_language_tl
                    { \l__zrefclever_main_language_tl }
                }
              }
              {
                \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
                \__zrefclever_language_if_declared:nF {#1}
                  {
                    \msg_warning:nnn { zref-clever }
                      { unknown-language-opt } {#1}
                  }
              }
            \__zrefclever_provide_langfile:x
              { \l__zrefclever_ref_language_tl }
          }
      } ,
    lang .initial:n = current ,
    lang .value_required:n = true ,
  }
\AddToHook { begindocument / before }
```

```
  {
    \AddToHook { begindocument }
      {
```

Redefinition of the `lang` key option for the document body. Also, drop the language file loading in the document body, it is somewhat redundant, since `\__zrefclever_zcref:nnn` already ensures it.

```
        \keys_define:nn { zref-clever/reference }
          {
            lang .code:n =
              {
                \str_case:nnF {#1}
                  {
                    { current }
                    {
                      \tl_set:Nn \l__zrefclever_ref_language_tl
                        { \l__zrefclever_current_language_tl }
                    }

                    { main }
                    {
                      \tl_set:Nn \l__zrefclever_ref_language_tl
                        { \l__zrefclever_main_language_tl }
                    }
                  }
                  {
                    \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
                    \__zrefclever_language_if_declared:nF {#1}
                      {
                        \msg_warning:nnn { zref-clever }
                          { unknown-language-opt } {#1}
                      }
                  }
              } ,
          }
      }
  }
```

`d` option

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.

```
\tl_new:N \l__zrefclever_ref_decl_case_tl
\keys_define:nn { zref-clever/reference }
  {
    d .code:n =
      { \msg_warning:nnn { zref-clever } { option-document-only } { d } } ,
  }
\AddToHook { begindocument }
```

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

We just store the value at this point, which is validated by `\__zrefclever_process_-language_settings:` after `\keys_set:nn`.

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

nudge & co. options

```
\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 }
  {
    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 }
                { 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}
               }
             }
           } ,
    nudgeif .value_required:n = true ,
    nudgeif .initial:n = all ,
    sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
    sg .initial:n = false ,
    sg .default:n = true ,
    g .code:n =
      { \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_settings:` after `\keys_set:nn`.

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

font option

```
\tl_new:N \l__zrefclever_ref_typeset_font_tl
\keys_define:nn { zref-clever/reference }
 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }
```

titleref option

```
\keys_define:nn { zref-clever/reference }
 {
  titleref .code:n =
   {
    % NOTE Option deprecated in 2022-04-22 for 0.3.0.
    \msg_warning:nnxx { zref-clever }{ option-deprecated } { titleref }
     { \iow_char:N\\usepackage\iow_char:N\{zref-titleref\iow_char:N\} }
   } ,
 }
```

vario option

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

```
  {
    vario .code:n =
      {
        % NOTE Option deprecated in 2022-04-22 for 0.3.0.
        \msg_warning:nnxx { zref-clever }{ option-deprecated } { vario }
          { \iow_char:N\\usepackage\iow_char:N\{zref-vario\iow_char:N\} }
      } ,
  }
```

note option

```
\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 =
      { \msg_warning:nnn { zref-clever } { option-document-only } { check } } ,
  }
\AddToHook { begindocument }
  {
    \__zrefclever_if_package_loaded:nTF { zref-check }
      {
        \IfPackageAtLeastTF { zref-check } { 2021-09-16 }
          {
            \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 .code:n =
                  {
                    \msg_warning:nnn { zref-clever }
                      { zref-check-too-old } { 2021-09-16~v0.2.1 }
                  } ,
              }
          }
      }
      {
        \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
        \keys_define:nn { zref-clever/reference }
          {
            check .code:n =
              { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
          }
      }
  }
```

reftype option

This allows one to manually specify the reference type. It is the equivalent of cleveref's optional argument to `\label`.

```
\tl_new:N \l__zrefclever_reftype_override_tl
\keys_define:nn { zref-clever/label }
  {
    reftype .tl_set:N = \l__zrefclever_reftype_override_tl ,
    reftype .default:n = {} ,
    reftype .initial:n = {} ,
  }
```

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 }
          }
          {
            \__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 `\subsubsubsection`".

`counterresetters` option

`\l__zrefclever_counter_resetters_seq` is used by `\__zrefclever_counter_reset_by: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}
              }
          }
      } ,
    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 `\__zrefclever_counter_reset_by: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 `\__zrefclever_counter_reset_by: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 }
          }
          {
            \__zrefclever_prop_put_non_empty:Nnn
              \l__zrefclever_counter_resetby_prop
          }
          {#1}
      } ,
    counterresetby .value_required:n = true ,
    counterresetby .initial:n =
      {
```

The counters for the `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.

```
        enumii  = enumi   ,
        enumiii = enumii  ,
        enumiv  = enumiii ,
      } ,
  }
```

currentcounter option

`\l__zrefclever_current_counter_tl` is pretty much the starting point of all of the data specification for label setting done by `zref` with our setup for it. It exists because we must provide some "handle" to specify the current counter for packages/features that do not set `\@currentcounter` appropriately.

```
\tl_new:N \l__zrefclever_current_counter_tl
\keys_define:nn { zref-clever/label }
  {
    currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
    currentcounter .default:n = \@currentcounter ,
    currentcounter .initial:n = \@currentcounter ,
  }
```

nocompat option

```
\bool_new:N \g__zrefclever_nocompat_bool
\seq_new:N \g__zrefclever_nocompat_modules_seq
\keys_define:nn { zref-clever/reference }
  {
    nocompat .code:n =
      {
        \tl_if_empty:nTF {#1}
          { \bool_gset_true:N \g__zrefclever_nocompat_bool }
          {
            \clist_map_inline:nn {#1}
              {
                \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {##1}
                  {
                    \seq_gput_right:Nn
                      \g__zrefclever_nocompat_modules_seq {##1}
                  }
              }
          }
      } ,
  }
\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
          { \msg_warning:nnn { zref-clever } { unknown-compat-module } {#1} }
      }
  }
```

`\__zrefclever_compat_module:nn` Function to be used for compatibility modules loading. It should load the module as long as `\l__zrefclever_nocompat_bool` is false and ⟨*module*⟩ is not in `\l__zrefclever_-nocompat_modules_seq`. The `begindocument` hook is needed so that we can have the option functional along the whole preamble, not just at package load time. This requirement might be relaxed if we made the option only available at load time, but this would not buy us much leeway anyway, since for most compatibility modules, we must test for the presence of packages at `begindocument`, only kernel features and document classes could be checked reliably before that. Besides, since we are using the new hook management system, there is always its functionality to deal with potential loading order issues.

`\__zrefclever_compat_module:nn {⟨module⟩} {⟨code⟩}`

```
\cs_new_protected:Npn \__zrefclever_compat_module:nn #1#2
  {
    \AddToHook { begindocument }
      {
        \bool_if:NF \g__zrefclever_nocompat_bool
          { \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {#1} {#2} }
        \seq_gremove_all:Nn \g__zrefclever_nocompat_modules_seq {#1}
      }
  }
```

(*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.

```
\seq_map_inline:Nn
  \g__zrefclever_rf_opts_tl_reference_seq
  {
    \keys_define:nn { zref-clever/reference }
      {
        #1 .default:o = \c_novalue_tl ,
        #1 .code:n =
          {
            \tl_if_novalue:nTF {##1}
              {
                \__zrefclever_opt_tl_unset:c
                  { \__zrefclever_opt_varname_general:nn {#1} { tl } }
              }
              {
                \__zrefclever_opt_tl_set:cn
                  { \__zrefclever_opt_varname_general:nn {#1} { tl } }
                  {##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 } { refbounds }
      } ,
    refpos .code:n =
      {
        % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
        \msg_warning:nnnn { zref-clever }{ option-deprecated }
          { refpos } { refbounds }
      } ,
    preref .code:n =
      {
        % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
        \msg_warning:nnnn { zref-clever }{ option-deprecated }
          { preref } { refbounds }
      } ,
    postref .code:n =
      {
        % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
        \msg_warning:nnnn { zref-clever }{ option-deprecated }
          { postref } { refbounds }
      } ,
  }
\seq_map_inline:Nn
  \g__zrefclever_rf_opts_seq_refbounds_seq
  {
    \keys_define:nn { zref-clever/reference }
      {
        #1 .default:o = \c_novalue_tl ,
        #1 .code:n =
          {
            \tl_if_novalue:nTF {##1}
              {
                \__zrefclever_opt_seq_unset:c
                  { \__zrefclever_opt_varname_general:nn {#1} { seq } }
              }
              {
                \seq_clear:N \l_tmpa_seq
                \__zrefclever_opt_seq_set_clist_split:Nn
                  \l_tmpa_seq {##1}
                \bool_lazy_or:nnTF
                  { \tl_if_empty_p:n {##1} }
                  { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
                  {
                    \__zrefclever_opt_seq_set_eq:cN
                      { \__zrefclever_opt_varname_general:nn {#1} { seq } }
                      \l_tmpa_seq
                  }
                  {
                    \msg_warning:nnxx { zref-clever }
                      { refbounds-must-be-four }
                      {#1} { \seq_count:N \l_tmpa_seq }
                  }
              }
          } ,
      }
  }
\seq_map_inline:Nn
  \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
  {
    \keys_define:nn { zref-clever/reference }
      {
        #1 .choice: ,
        #1 / true .code:n =
          {
            \__zrefclever_opt_bool_set_true:c
              { \__zrefclever_opt_varname_general:nn {#1} { bool } }
          } ,
        #1 / false .code:n =
          {
            \__zrefclever_opt_bool_set_false:c
              { \__zrefclever_opt_varname_general:nn {#1} { bool } }
          } ,
        #1 / unset .code:n =
          {
            \__zrefclever_opt_bool_unset:c
              { \__zrefclever_opt_varname_general:nn {#1} { bool } }
          } ,
        #1 .default:n = true ,
        no #1 .meta:n = { #1 = false } ,
        no #1 .value_forbidden:n = true ,
      }
  }
```

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 `\zcref`'s options. Anyway, for package options (`\zcsetup`) we want the whole set, so we aggregate the two into `zref-clever/zcsetup`, and use that here.

```
\keys_define:nn { }
  {
    zref-clever/zcsetup .inherit:n =
      {
        zref-clever/label ,
        zref-clever/reference ,
      }
  }
```

zref-clever does not accept load-time options. Despite the tradition of so doing, Joseph Wright has a point in recommending otherwise at https://chat.stackexchange.com/transcript/message/60360822#60360822: separating "loading the package" from "configuring the package" grants less trouble with "option clashes" and with expansion of options at load-time.

```
\bool_lazy_and:nnT
  { \tl_if_exist_p:c { opt@ zref-clever.sty } }
  { ! \tl_if_empty_p:c { opt@ zref-clever.sty } }
  { \msg_warning:nn { zref-clever } { load-time-options } }
```

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

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

```
\NewDocumentCommand \zcRefTypeSetup { m m }
  {
    \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
    \keys_set:nn { zref-clever/typesetup } {#2}
    \tl_clear:N \l__zrefclever_setup_type_tl
  }
```

(*End definition for* `\zcRefTypeSetup`.)

```
\seq_map_inline:Nn
  \g__zrefclever_rf_opts_tl_not_type_specific_seq
  {
    \keys_define:nn { zref-clever/typesetup }
      {
        #1 .code:n =
          {
            \msg_warning:nnn { zref-clever }
              { option-not-type-specific } {#1}
          } ,
      }
  }
\seq_map_inline:Nn
```

```
  \g__zrefclever_rf_opts_tl_typesetup_seq
  {
    \keys_define:nn { zref-clever/typesetup }
      {
        #1 .default:o = \c_novalue_tl ,
        #1 .code:n =
          {
            \tl_if_novalue:nTF {##1}
              {
                \__zrefclever_opt_tl_unset:c
                  {
                    \__zrefclever_opt_varname_type:enn
                      { \l__zrefclever_setup_type_tl } {#1} { tl }
                  }
              }
              {
                \__zrefclever_opt_tl_set:cn
                  {
                    \__zrefclever_opt_varname_type:enn
                      { \l__zrefclever_setup_type_tl } {#1} { tl }
                  }
                  {##1}
              }
          } ,
      }
  }
\keys_define:nn { zref-clever/typesetup }
  {
    endrange .code:n =
      {
        \str_case:nnF {#1}
          {
            { ref }
            {
              \__zrefclever_opt_tl_clear:c
                {
                  \__zrefclever_opt_varname_type:enn
                    { \l__zrefclever_setup_type_tl } { endrangefunc } { tl }
                }
              \__zrefclever_opt_tl_clear:c
                {
                  \__zrefclever_opt_varname_type:enn
                    { \l__zrefclever_setup_type_tl } { endrangeprop } { tl }
                }
            }

            { stripprefix }
            {
              \__zrefclever_opt_tl_set:cn
                {
                  \__zrefclever_opt_varname_type:enn
                    { \l__zrefclever_setup_type_tl } { endrangefunc } { tl }
                }
                { __zrefclever_get_endrange_stripprefix }
  \__zrefclever_opt_tl_clear:c
   {
    \__zrefclever_opt_varname_type:enn
     { \l__zrefclever_setup_type_tl } { endrangeprop } { tl }
   }
 }

 { pagecomp }
 {
  \__zrefclever_opt_tl_set:cn
   {
    \__zrefclever_opt_varname_type:enn
     { \l__zrefclever_setup_type_tl } { endrangefunc } { tl }
   }
   { __zrefclever_get_endrange_pagecomp }
  \__zrefclever_opt_tl_clear:c
   {
    \__zrefclever_opt_varname_type:enn
     { \l__zrefclever_setup_type_tl } { endrangeprop } { tl }
   }
 }

 { pagecomp2 }
 {
  \__zrefclever_opt_tl_set:cn
   {
    \__zrefclever_opt_varname_type:enn
     { \l__zrefclever_setup_type_tl } { endrangefunc } { tl }
   }
   { __zrefclever_get_endrange_pagecomptwo }
  \__zrefclever_opt_tl_clear:c
   {
    \__zrefclever_opt_varname_type:enn
     { \l__zrefclever_setup_type_tl } { endrangeprop } { tl }
   }
 }

 { unset }
 {
  \__zrefclever_opt_tl_unset:c
   {
    \__zrefclever_opt_varname_type:enn
     { \l__zrefclever_setup_type_tl } { endrangefunc } { tl }
   }
  \__zrefclever_opt_tl_unset:c
   {
    \__zrefclever_opt_varname_type:enn
     { \l__zrefclever_setup_type_tl } { endrangeprop } { tl }
   }
 }
}
{
 \tl_if_empty:nTF {#1}
  {
```

```
              \msg_warning:nnn { zref-clever }
                { endrange-property-undefined } {#1}
            }
            {
              \zref@ifpropundefined {#1}
                {
                  \msg_warning:nnn { zref-clever }
                    { endrange-property-undefined } {#1}
                }
                {
                  \__zrefclever_opt_tl_set:cn
                    {
                      \__zrefclever_opt_varname_type:enn
                        { \l__zrefclever_setup_type_tl }
                        { endrangefunc } { tl }
                    }
                    { __zrefclever_get_endrange_property }
                  \__zrefclever_opt_tl_set:cn
                    {
                      \__zrefclever_opt_varname_type:enn
                        { \l__zrefclever_setup_type_tl }
                        { endrangeprop } { tl }
                    }
                    {#1}
                }
            }
        }
    } ,
  endrange .value_required:n = true ,
}
\keys_define:nn { zref-clever/typesetup }
{
  refpre .code:n =
    {
      % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
      \msg_warning:nnnn { zref-clever }{ option-deprecated }
        { refpre } { refbounds }
    } ,
  refpos .code:n =
    {
      % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
      \msg_warning:nnnn { zref-clever }{ option-deprecated }
        { refpos } { refbounds }
    } ,
  preref .code:n =
    {
      % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
      \msg_warning:nnnn { zref-clever }{ option-deprecated }
        { preref } { refbounds }
    } ,
  postref .code:n =
    {
      % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
      \msg_warning:nnnn { zref-clever }{ option-deprecated }
```

```
        { postref } { refbounds }
    } ,
  }
\seq_map_inline:Nn
  \g__zrefclever_rf_opts_seq_refbounds_seq
  {
    \keys_define:nn { zref-clever/typesetup }
      {
        #1 .default:o = \c_novalue_tl ,
        #1 .code:n =
          {
            \tl_if_novalue:nTF {##1}
              {
                \__zrefclever_opt_seq_unset:c
                  {
                    \__zrefclever_opt_varname_type:enn
                      { \l__zrefclever_setup_type_tl } {#1} { seq }
                  }
              }
              {
                \seq_clear:N \l_tmpa_seq
                \__zrefclever_opt_seq_set_clist_split:Nn
                  \l_tmpa_seq {##1}
                \bool_lazy_or:nnTF
                  { \tl_if_empty_p:n {##1} }
                  { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
                  {
                    \__zrefclever_opt_seq_set_eq:cN
                      {
                        \__zrefclever_opt_varname_type:enn
                          { \l__zrefclever_setup_type_tl } {#1} { seq }
                      }
                      \l_tmpa_seq
                  }
                  {
                    \msg_warning:nnxx { zref-clever }
                      { refbounds-must-be-four }
                      {#1} { \seq_count:N \l_tmpa_seq }
                  }
              }
          } ,
      }
  }
\seq_map_inline:Nn
  \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
  {
    \keys_define:nn { zref-clever/typesetup }
      {
        #1 .choice: ,
        #1 / true .code:n =
          {
            \__zrefclever_opt_bool_set_true:c
              {
                \__zrefclever_opt_varname_type:enn
```

```
                  { \l__zrefclever_setup_type_tl }
                  {#1} { bool }
              }
          } ,
        #1 / false .code:n =
          {
            \__zrefclever_opt_bool_set_false:c
              {
                \__zrefclever_opt_varname_type:enn
                  { \l__zrefclever_setup_type_tl }
                  {#1} { bool }
              }
          } ,
        #1 / unset .code:n =
          {
            \__zrefclever_opt_bool_unset:c
              {
                \__zrefclever_opt_varname_type:enn
                  { \l__zrefclever_setup_type_tl }
                  {#1} { bool }
              }
          } ,
        #1 .default:n = true ,
        no #1 .meta:n = { #1 = false } ,
        no #1 .value_forbidden:n = true ,
      }
  }
```

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.

\zcLanguageSetup

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

```
\NewDocumentCommand \zcLanguageSetup { m m }
  {
    \group_begin:
    \__zrefclever_language_if_declared:nTF {#1}
      {
        \tl_clear:N \l__zrefclever_setup_type_tl
        \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
        \__zrefclever_opt_seq_get:cNF
          {
            \__zrefclever_opt_varname_language:nnn
              {#1} { declension } { seq }
          }
          \l__zrefclever_lang_declension_seq
          { \seq_clear:N \l__zrefclever_lang_declension_seq }
        \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
          { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
          {
            \seq_get_left:NN \l__zrefclever_lang_declension_seq
              \l__zrefclever_lang_decl_case_tl
          }
        \__zrefclever_opt_seq_get:cNF
          {
            \__zrefclever_opt_varname_language:nnn
              {#1} { gender } { seq }
          }
          \l__zrefclever_lang_gender_seq
          { \seq_clear:N \l__zrefclever_lang_gender_seq }
        \keys_set:nn { zref-clever/langsetup } {#2}
      }
      { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
    \group_end:
  }
\@onlypreamble \zcLanguageSetup
```

(*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 .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_warning:nnxx { zref-clever } { language-no-decl-setup }
              { \l__zrefclever_setup_language_tl } {#1}
          }
          {
            \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
              { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
              {
                \msg_warning:nnxx { zref-clever } { unknown-decl-case }
                  {#1} { \l__zrefclever_setup_language_tl }
                \seq_get_left:NN \l__zrefclever_lang_declension_seq
                  \l__zrefclever_lang_decl_case_tl
              }
          }
      } ,
    case .value_required:n = true ,

    gender .value_required:n = true ,
    gender .code:n =
      {
        \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
          {
            \msg_warning:nnxxx { zref-clever } { language-no-gender }
              { \l__zrefclever_setup_language_tl } { gender } {#1}
          }
          {
            \tl_if_empty:NTF \l__zrefclever_setup_type_tl
              {
                \msg_warning:nnn { zref-clever }
                  { option-only-type-specific } { gender }
              }
              {
                \seq_clear:N \l_tmpa_seq
                \clist_map_inline:nn {#1}
                  {
                    \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
                      { \seq_put_right:Nn \l_tmpa_seq {##1} }
                      {
                        \msg_warning:nnxx { zref-clever }
                          { gender-not-declared }
                          { \l__zrefclever_setup_language_tl } {##1}
                      }
                  }
                \__zrefclever_opt_seq_gset_eq:cN
                  {
                    \__zrefclever_opt_varname_lang_type:eenn
                      { \l__zrefclever_setup_language_tl }
                      { \l__zrefclever_setup_type_tl }
                      { gender }
                      { seq }
                  }
                  \l_tmpa_seq
              }
          }
      } ,
  }
\seq_map_inline:Nn
  \g__zrefclever_rf_opts_tl_not_type_specific_seq
  {
    \keys_define:nn { zref-clever/langsetup }
      {
        #1 .value_required:n = true ,
        #1 .code:n =
          {
            \tl_if_empty:NTF \l__zrefclever_setup_type_tl
              {
                \__zrefclever_opt_tl_gset:cn
                  {
                    \__zrefclever_opt_varname_lang_default:enn
                      { \l__zrefclever_setup_language_tl } {#1} { tl }
                  }
                  {##1}
              }
            {
              \msg_warning:nnn { zref-clever }
                { option-not-type-specific } {#1}
            }
        } ,
    }
  }
\seq_map_inline:Nn
  \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
  {
    \keys_define:nn { zref-clever/langsetup }
      {
        #1 .value_required:n = true ,
        #1 .code:n =
          {
            \tl_if_empty:NTF \l__zrefclever_setup_type_tl
              {
                \__zrefclever_opt_tl_gset:cn
                  {
                    \__zrefclever_opt_varname_lang_default:enn
                      { \l__zrefclever_setup_language_tl } {#1} { tl }
                  }
                  {##1}
              }
              {
                \__zrefclever_opt_tl_gset:cn
                  {
                    \__zrefclever_opt_varname_lang_type:eenn
                      { \l__zrefclever_setup_language_tl }
                      { \l__zrefclever_setup_type_tl }
                      {#1} { tl }
                  }
                  {##1}
              }
          } ,
      }
  }
\keys_define:nn { zref-clever/langsetup }
  {
    endrange .value_required:n = true ,
    endrange .code:n =
      {
        \str_case:nnF {#1}
          {
            { ref }
            {
              \tl_if_empty:NTF \l__zrefclever_setup_type_tl
                {
                  \__zrefclever_opt_tl_gclear:c
                    {
                      \__zrefclever_opt_varname_lang_default:enn
                        { \l__zrefclever_setup_language_tl }
                        { endrangefunc } { tl }
                    }
                  \__zrefclever_opt_tl_gclear:c
                    {
                      \__zrefclever_opt_varname_lang_default:enn
                        { \l__zrefclever_setup_language_tl }
                        { endrangeprop } { tl }
                    }
                }
                {
                  \__zrefclever_opt_tl_gclear:c
                    {
                      \__zrefclever_opt_varname_lang_type:eenn
                        { \l__zrefclever_setup_language_tl }
                        { \l__zrefclever_setup_type_tl }
                        { endrangefunc } { tl }
                    }
                  \__zrefclever_opt_tl_gclear:c
                    {
                      \__zrefclever_opt_varname_lang_type:eenn
                        { \l__zrefclever_setup_language_tl }
                        { \l__zrefclever_setup_type_tl }
                        { endrangeprop } { tl }
                    }
                }
            }

            { stripprefix }
            {
              \tl_if_empty:NTF \l__zrefclever_setup_type_tl
                {
                  \__zrefclever_opt_tl_gset:cn
                    {
                      \__zrefclever_opt_varname_lang_default:enn
                        { \l__zrefclever_setup_language_tl }
                        { endrangefunc } { tl }
                    }
                    { __zrefclever_get_endrange_stripprefix }
                  \__zrefclever_opt_tl_gclear:c
                    {
                      \__zrefclever_opt_varname_lang_default:enn
                        { \l__zrefclever_setup_language_tl }
                        { endrangeprop } { tl }
                    }
                }
                {
                  \__zrefclever_opt_tl_gset:cn
                    {
                      \__zrefclever_opt_varname_lang_type:eenn
                        { \l__zrefclever_setup_language_tl }
                        { \l__zrefclever_setup_type_tl }
                        { endrangefunc } { tl }
                    }
                    { __zrefclever_get_endrange_stripprefix }
                  \__zrefclever_opt_tl_gclear:c
                    {
```

```
              \__zrefclever_opt_varname_lang_type:eenn
                { \l__zrefclever_setup_language_tl }
                { \l__zrefclever_setup_type_tl }
                { endrangeprop } { tl }
            }
        }
    }

    { pagecomp }
    {
      \tl_if_empty:NTF \l__zrefclever_setup_type_tl
        {
          \__zrefclever_opt_tl_gset:cn
            {
              \__zrefclever_opt_varname_lang_default:enn
                { \l__zrefclever_setup_language_tl }
                { endrangefunc } { tl }
            }
            { __zrefclever_get_endrange_pagecomp }
          \__zrefclever_opt_tl_gclear:c
            {
              \__zrefclever_opt_varname_lang_default:enn
                { \l__zrefclever_setup_language_tl }
                { endrangeprop } { tl }
            }
        }
        {
          \__zrefclever_opt_tl_gset:cn
            {
              \__zrefclever_opt_varname_lang_type:eenn
                { \l__zrefclever_setup_language_tl }
                { \l__zrefclever_setup_type_tl }
                { endrangefunc } { tl }
            }
            { __zrefclever_get_endrange_pagecomp }
          \__zrefclever_opt_tl_gclear:c
            {
              \__zrefclever_opt_varname_lang_type:eenn
                { \l__zrefclever_setup_language_tl }
                { \l__zrefclever_setup_type_tl }
                { endrangeprop } { tl }
            }
        }
    }

    { pagecomp2 }
    {
      \tl_if_empty:NTF \l__zrefclever_setup_type_tl
        {
          \__zrefclever_opt_tl_gset:cn
            {
              \__zrefclever_opt_varname_lang_default:enn
                { \l__zrefclever_setup_language_tl }
                { endrangefunc } { tl }
```

```
     }
     { __zrefclever_get_endrange_pagecomptwo }
    \__zrefclever_opt_tl_gclear:c
     {
      \__zrefclever_opt_varname_lang_default:enn
       { \l__zrefclever_setup_language_tl }
       { endrangeprop } { tl }
     }
   }
   {
    \__zrefclever_opt_tl_gset:cn
     {
      \__zrefclever_opt_varname_lang_type:eenn
       { \l__zrefclever_setup_language_tl }
       { \l__zrefclever_setup_type_tl }
       { endrangefunc } { tl }
     }
     { __zrefclever_get_endrange_pagecomptwo }
    \__zrefclever_opt_tl_gclear:c
     {
      \__zrefclever_opt_varname_lang_type:eenn
       { \l__zrefclever_setup_language_tl }
       { \l__zrefclever_setup_type_tl }
       { endrangeprop } { tl }
     }
   }
 }
}
{
 \tl_if_empty:nTF {#1}
  {
   \msg_warning:nnn { zref-clever }
    { endrange-property-undefined } {#1}
  }
  {
   \zref@ifpropundefined {#1}
    {
     \msg_warning:nnn { zref-clever }
      { endrange-property-undefined } {#1}
    }
    {
     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
      {
       \__zrefclever_opt_tl_gset:cn
        {
         \__zrefclever_opt_varname_lang_default:enn
          { \l__zrefclever_setup_language_tl }
          { endrangefunc } { tl }
        }
        { __zrefclever_get_endrange_property }
       \__zrefclever_opt_tl_gset:cn
        {
         \__zrefclever_opt_varname_lang_default:enn
          { \l__zrefclever_setup_language_tl }
```

```
               { endrangeprop } { tl }
             }
             {#1}
           }
           {
             \__zrefclever_opt_tl_gset:cn
               {
                 \__zrefclever_opt_varname_lang_type:eenn
                   { \l__zrefclever_setup_language_tl }
                   { \l__zrefclever_setup_type_tl }
                   { endrangefunc } { tl }
               }
               { __zrefclever_get_endrange_property }
             \__zrefclever_opt_tl_gset:cn
               {
                 \__zrefclever_opt_varname_lang_type:eenn
                   { \l__zrefclever_setup_language_tl }
                   { \l__zrefclever_setup_type_tl }
                   { endrangeprop } { tl }
               }
               {#1}
           }
         }
       }
     }
   } ,
 }
\keys_define:nn { zref-clever/langsetup }
 {
  refpre .code:n =
   {
    % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
    \msg_warning:nnnn { zref-clever }{ option-deprecated }
     { refpre } { refbounds }
   } ,
  refpos .code:n =
   {
    % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
    \msg_warning:nnnn { zref-clever }{ option-deprecated }
     { refpos } { refbounds }
   } ,
  preref .code:n =
   {
    % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
    \msg_warning:nnnn { zref-clever }{ option-deprecated }
     { preref } { refbounds }
   } ,
  postref .code:n =
   {
    % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
    \msg_warning:nnnn { zref-clever }{ option-deprecated }
     { postref } { refbounds }
   } ,
 }
```

```
\seq_map_inline:Nn
  \g__zrefclever_rf_opts_tl_type_names_seq
  {
    \keys_define:nn { zref-clever/langsetup }
      {
        #1 .value_required:n = true ,
        #1 .code:n =
          {
            \tl_if_empty:NTF \l__zrefclever_setup_type_tl
              {
                \msg_warning:nnn { zref-clever }
                  { option-only-type-specific } {#1}
              }
              {
                \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
                  {
                    \__zrefclever_opt_tl_gset:cn
                      {
                        \__zrefclever_opt_varname_lang_type:eenn
                          { \l__zrefclever_setup_language_tl }
                          { \l__zrefclever_setup_type_tl }
                          {#1} { tl }
                      }
                      {##1}
                  }
                  {
                    \__zrefclever_opt_tl_gset:cn
                      {
                        \__zrefclever_opt_varname_lang_type:eeen
                          { \l__zrefclever_setup_language_tl }
                          { \l__zrefclever_setup_type_tl }
                          { \l__zrefclever_lang_decl_case_tl - #1 }
                          { tl }
                      }
                      {##1}
                  }
              }
          } ,
      }
  }
\seq_map_inline:Nn
  \g__zrefclever_rf_opts_seq_refbounds_seq
  {
    \keys_define:nn { zref-clever/langsetup }
      {
        #1 .value_required:n = true ,
        #1 .code:n =
          {
            \tl_if_empty:NTF \l__zrefclever_setup_type_tl
              {
                \seq_gclear:N \g_tmpa_seq
                \__zrefclever_opt_seq_gset_clist_split:Nn
                  \g_tmpa_seq {##1}
                \bool_lazy_or:nnTF
```

```
                  { \tl_if_empty_p:n {##1} }
                  {
                    \int_compare_p:nNn
                      { \seq_count:N \g_tmpa_seq } = { 4 }
                  }
                  {
                    \__zrefclever_opt_seq_gset_eq:cN
                      {
                        \__zrefclever_opt_varname_lang_default:enn
                          { \l__zrefclever_setup_language_tl }
                          {#1} { seq }
                      }
                      \g_tmpa_seq
                  }
                  {
                    \msg_warning:nnxx { zref-clever }
                      { refbounds-must-be-four }
                      {#1} { \seq_count:N \g_tmpa_seq }
                  }
              }
              {
                \seq_gclear:N \g_tmpa_seq
                \__zrefclever_opt_seq_gset_clist_split:Nn
                  \g_tmpa_seq {##1}
                \bool_lazy_or:nnTF
                  { \tl_if_empty_p:n {##1} }
                  {
                    \int_compare_p:nNn
                      { \seq_count:N \g_tmpa_seq } = { 4 }
                  }
                  {
                    \__zrefclever_opt_seq_gset_eq:cN
                      {
                        \__zrefclever_opt_varname_lang_type:eenn
                          { \l__zrefclever_setup_language_tl }
                          { \l__zrefclever_setup_type_tl } {#1} { seq }
                      }
                      \g_tmpa_seq
                  }
                  {
                    \msg_warning:nnxx { zref-clever }
                      { refbounds-must-be-four }
                      {#1} { \seq_count:N \g_tmpa_seq }
                  }
              }
          } ,
      }
  }
\seq_map_inline:Nn
  \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
  {
    \keys_define:nn { zref-clever/langsetup }
      {
        #1 .choice: ,
        #1 / true .code:n =
          {
            \tl_if_empty:NTF \l__zrefclever_setup_type_tl
              {
                \__zrefclever_opt_bool_gset_true:c
                  {
                    \__zrefclever_opt_varname_lang_default:enn
                      { \l__zrefclever_setup_language_tl }
                      {#1} { bool }
                  }
              }
              {
                \__zrefclever_opt_bool_gset_true:c
                  {
                    \__zrefclever_opt_varname_lang_type:eenn
                      { \l__zrefclever_setup_language_tl }
                      { \l__zrefclever_setup_type_tl }
                      {#1} { bool }
                  }
              }
          } ,
        #1 / false .code:n =
          {
            \tl_if_empty:NTF \l__zrefclever_setup_type_tl
              {
                \__zrefclever_opt_bool_gset_false:c
                  {
                    \__zrefclever_opt_varname_lang_default:enn
                      { \l__zrefclever_setup_language_tl }
                      {#1} { bool }
                  }
              }
              {
                \__zrefclever_opt_bool_gset_false:c
                  {
                    \__zrefclever_opt_varname_lang_type:eenn
                      { \l__zrefclever_setup_language_tl }
                      { \l__zrefclever_setup_type_tl }
                      {#1} { bool }
                  }
              }
          } ,
        #1 .default:n = true ,
        no #1 .meta:n = { #1 = false } ,
        no #1 .value_forbidden:n = true ,
      }
  }
```

6 User interface

6.1 \zcref

\zcref The main user command of the package.

\zcref⟨*⟩[⟨*options*⟩]{⟨*labels*⟩}

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

(*End definition for* `\zcref`*.*)

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

__zrefclever_zcref:nnnn {⟨*labels*⟩} {⟨*⟩} {⟨*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 \l__zrefclever_zcref_labels_seq {#1}
      \bool_set:Nn \l__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 language settings.

```
      \__zrefclever_process_language_settings:
```

Integration with zref-check.

```
      \bool_lazy_and:nnT
        { \l__zrefclever_zrefcheck_available_bool }
        { \l__zrefclever_zcref_with_check_bool }
        { \zrefcheck_zcref_beg_label: }
```

Sort the labels.

```
      \bool_lazy_or:nnT
        { \l__zrefclever_typeset_sort_bool }
        { \l__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:
      \l__zrefclever_ref_typeset_font_tl
      \__zrefclever_typeset_refs:
      \group_end:
```

Typeset `note`.

```
      \tl_if_empty:NF \l__zrefclever_zcref_note_tl
        {
          \__zrefclever_get_rf_opt_tl:nxxN { notesep }
            { \l__zrefclever_label_type_a_tl }
            { \l__zrefclever_ref_language_tl }
            \l_tmpa_tl
          \l_tmpa_tl
          \l__zrefclever_zcref_note_tl
        }
```

Integration with zref-check.

```
    \bool_lazy_and:nnT
      { \l__zrefclever_zrefcheck_available_bool }
      { \l__zrefclever_zcref_with_check_bool }
      {
        \zrefcheck_zcref_end_label_maybe:
        \zrefcheck_zcref_run_checks_on_labels:n
          { \l__zrefclever_zcref_labels_seq }
      }
```

Integration with mathtools.

```
  \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
    {
      \__zrefclever_mathtools_showonlyrefs:n
        { \l__zrefclever_zcref_labels_seq }
    }
  \group_end:
}
```

(*End definition for* `\__zrefclever_zcref:nnnn`.)

\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 definition for* `\l__zrefclever_zcref_labels_seq` *and* `\l__zrefclever_link_star_bool`.)

6.2 \zcpageref

\zcpageref

A `\pageref` equivalent of `\zcref`.

`\zcpageref⟨*⟩[⟨options⟩]{⟨labels⟩}`

```
\NewDocumentCommand \zcpageref { s O { } m }
  {
    \group_begin:
    \IfBooleanT {#1}
      { \bool_set_false:N \l__zrefclever_hyperlink_bool }
    \zcref [#2, ref = page] {#3}
    \group_end:
  }
```

(*End definition for* `\zcpageref`.)

7 Sorting

Sorting is certainly a "big task" for 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 abspage, from the 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 `typesort` option or, if that is silent for the case, by the order in which labels were given by the user in `\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.

\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

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.

```
\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 definition for* `\l__zrefclever_label_type_a_tl` *and others.*)

\l__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 \l__zrefclever_sort_decided_bool
```

(*End definition for* `\l__zrefclever_sort_decided_bool`*.*)

\l__zrefclever_sort_prior_a_int
\l__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 \l__zrefclever_sort_prior_a_int
\int_new:N \l__zrefclever_sort_prior_b_int
```

(*End definition for* `\l__zrefclever_sort_prior_a_int` *and* `\l__zrefclever_sort_prior_b_int`*.*)

\l__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 \l__zrefclever_label_types_seq
```

(*End definition for* `\l__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:nnnn` where a number of environment variables are to be set appropriately. In particular, `\l__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:
  {
```

Store label types sequence.

```
    \seq_clear:N \l__zrefclever_label_types_seq
    \tl_if_eq:NnF \l__zrefclever_ref_propserty_tl { page }
      {
        \seq_map_function:NN \l__zrefclever_zcref_labels_seq
          \__zrefclever_label_type_put_new_right:n
      }
```

Sort.

```
    \seq_sort:Nn \l__zrefclever_zcref_labels_seq
      {
        \zref@ifrefundefined {##1}
          {
            \zref@ifrefundefined {##2}
              {
                % 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:
              }
          }
          {
            \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} }
              }
          }
      }
  }
```

(*End definition for* `\__zrefclever_sort_labels:`.)

`\__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 } { }
    \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
      }
  }
```

(*End definition for* `\__zrefclever_label_type_put_new_right:n`.)

`\__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`

```
\__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 } { }
    \tl_reverse:N \l__zrefclever_label_enclval_a_tl
    \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_b_tl
      {#2} { zc@enclval } { }
    \tl_reverse:N \l__zrefclever_label_enclval_b_tl
    \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
      {#1} { externaldocument } { }
    \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
      {#2} { externaldocument } { }

    \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
      {
        \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
              }
              {
                \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 }
                        <
                      { \__zrefclever_extract:nnn {#2} { zc@cntval } { } }
                      { \sort_return_same:    }
                      { \sort_return_swapped: }
                  }
                  {
                    % Neither is empty: we can compare the values of the
                    % current enclosing counter in the loop, if they are
                    % equal, we are still in the loop, if they are not, a
                    % sorting decision can be made directly.
                    \int_compare:nNnTF
                      { \tl_head:N \l__zrefclever_label_enclval_a_tl }
                        =
                      { \tl_head:N \l__zrefclever_label_enclval_b_tl }
                      {
                        \tl_set:Nx \l__zrefclever_label_enclval_a_tl
                          { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
                        \tl_set:Nx \l__zrefclever_label_enclval_b_tl
                          { \tl_tail:N \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_head:N \l__zrefclever_label_enclval_b_tl }
```

```
                              { \sort_return_swapped: }
                              { \sort_return_same:    }
                          }
                      }
                  }
              }
          }
      }
```

(*End definition for* `\__zrefclever_sort_default_same_type:nn`.)

_zrefclever_sort_default_different_types:nn

`\__zrefclever_sort_default_different_types:nn {⟨label a⟩} {⟨label b⟩}`

```
\cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
  {
```

Retrieve sort priorities for ⟨*label a*⟩ and ⟨*label b*⟩. `\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
      {
        \tl_if_eq:nnTF {##2} {{othertypes}}
          {
            \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
              { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
            \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
              { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
          }
          {
            \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
              { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
              {
                \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
                  { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
              }
          }
      }
```

Then do the actual sorting.

```
    \bool_if:nTF
      {
        \int_compare_p:nNn
          { \l__zrefclever_sort_prior_a_int } <
          { \l__zrefclever_sort_prior_b_int }
      }
      { \sort_return_same: }
      {
        \bool_if:nTF
          {
            \int_compare_p:nNn
              { \l__zrefclever_sort_prior_a_int } >
              { \l__zrefclever_sort_prior_b_int }
          }
```

```
            { \sort_return_swapped: }
            {
              % Sort priorities are equal: the type that occurs first in
              % 'labels', as given by the user, is kept (or brought) forward.
              \seq_map_inline:Nn \l__zrefclever_label_types_seq
                {
                  \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
                    { \seq_map_break:n { \sort_return_same: } }
                    {
                      \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
                        { \seq_map_break:n { \sort_return_swapped: } }
                    }
                }
            }
        }
  }
```

(*End definition for* `\__zrefclever_sort_default_different_types:nn`.)

`\__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 `\sort_return_same:` or `\sort_return_swapped:`. Compared to the sorting of default labels, this is a piece of cake (thanks to abspage).

```
\__zrefclever_sort_page:nn {⟨label a⟩} {⟨label b⟩}
```

```
\cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
  {
    \int_compare:nNnTF
      { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
        >
      { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
      { \sort_return_swapped: }
      { \sort_return_same:    }
  }
```

(*End definition for* `\__zrefclever_sort_page:nn`.)

8 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 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 `.dtx` file.

While processing the label stack (kept in `\l__zrefclever_typeset_labels_seq`), `\__zrefclever_typeset_refs:` "sees" two labels, and two labels only, the "current" one (kept in `\l__zrefclever_label_a_tl`), and the "next" one (kept in `\l__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 `\l__zrefclever_last_of_type_bool`), or the stack itself finishes (has no more elements, signaled by `\l__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 `\l__zrefclever_type_first_label_tl`, with `\l__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": `\l__zrefclever_typeset_queue_curr_tl` and `\l__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" (`\l__zrefclever_type_count_int`) and one for the "label in the current type block" (`\l__zrefclever_label_count_int`).

Range compression, in particular, relies heavily on counting to be able do distinguish relevant cases. `\l__zrefclever_range_count_int` counts the number of elements in the current sequential "streak", and `\l__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 `\l__zrefclever_range_beg_label_tl`). `\l__zrefclever_next_maybe_range_bool` signals when "next" is potentially a range with "current", and `\l__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

\l__zrefclever_typeset_labels_seq
\l__zrefclever_typeset_last_bool
\l__zrefclever_last_of_type_bool

Auxiliary variables for `\__zrefclever_typeset_refs:` main stack control.

```
\seq_new:N \l__zrefclever_typeset_labels_seq
\bool_new:N \l__zrefclever_typeset_last_bool
\bool_new:N \l__zrefclever_last_of_type_bool
```

(*End definition for* `\l__zrefclever_typeset_labels_seq`*,* `\l__zrefclever_typeset_last_bool`*, and* `\l__zrefclever_last_of_type_bool`*.*)

\l__zrefclever_type_count_int
\l__zrefclever_label_count_int
\l__zrefclever_ref_count_int

Auxiliary variables for `\__zrefclever_typeset_refs:` main counters.

```
\int_new:N \l__zrefclever_type_count_int
\int_new:N \l__zrefclever_label_count_int
\int_new:N \l__zrefclever_ref_count_int
```

(*End definition for* `\l__zrefclever_type_count_int`*,* `\l__zrefclever_label_count_int`*, and* `\l__zrefclever_ref_count_int`*.*)

\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.

```
\tl_new:N \l__zrefclever_label_a_tl
\tl_new:N \l__zrefclever_label_b_tl
\tl_new:N \l__zrefclever_typeset_queue_prev_tl
\tl_new:N \l__zrefclever_typeset_queue_curr_tl
\tl_new:N \l__zrefclever_type_first_label_tl
\tl_new:N \l__zrefclever_type_first_label_type_tl
```

(*End definition for* `\l__zrefclever_label_a_tl` *and others.*)

\l__zrefclever_type_name_tl
\l__zrefclever_name_in_link_bool
\l__zrefclever_type_name_missing_bool
\l__zrefclever_name_format_tl
\l__zrefclever_name_format_fallback_tl
\l__zrefclever_type_name_gender_seq

Auxiliary variables for `\__zrefclever_typeset_refs:` type name handling.

```
\tl_new:N \l__zrefclever_type_name_tl
\bool_new:N \l__zrefclever_name_in_link_bool
\bool_new:N \l__zrefclever_type_name_missing_bool
\tl_new:N \l__zrefclever_name_format_tl
\tl_new:N \l__zrefclever_name_format_fallback_tl
\seq_new:N \l__zrefclever_type_name_gender_seq
```

(*End definition for* `\l__zrefclever_type_name_tl` *and others.*)

\l__zrefclever_range_count_int
\l__zrefclever_range_same_count_int
\l__zrefclever_range_beg_label_tl
\l__zrefclever_range_beg_is_first_bool
\l__zrefclever_range_end_ref_tl
\l__zrefclever_next_maybe_range_bool
\l__zrefclever_next_is_same_bool

Auxiliary variables for `\__zrefclever_typeset_refs:` range handling.

```
\int_new:N \l__zrefclever_range_count_int
\int_new:N \l__zrefclever_range_same_count_int
\tl_new:N \l__zrefclever_range_beg_label_tl
\bool_new:N \l__zrefclever_range_beg_is_first_bool
\tl_new:N \l__zrefclever_range_end_ref_tl
\bool_new:N \l__zrefclever_next_maybe_range_bool
\bool_new:N \l__zrefclever_next_is_same_bool
```

(*End definition for* `\l__zrefclever_range_count_int` *and others.*)

\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_namefont_tl
\l__zrefclever_reffont_tl
\l__zrefclever_endrangefunc_tl
\l__zrefclever_endrangeprop_tl
\l__zrefclever_cap_bool
\l__zrefclever_abbrev_bool
\l__zrefclever_rangetopair_bool

Auxiliary variables for `\__zrefclever_typeset_refs:` separators, and font and other options.

```
\tl_new:N \l__zrefclever_tpairsep_tl
\tl_new:N \l__zrefclever_tlistsep_tl
\tl_new:N \l__zrefclever_tlastsep_tl
\tl_new:N \l__zrefclever_namesep_tl
\tl_new:N \l__zrefclever_pairsep_tl
\tl_new:N \l__zrefclever_listsep_tl
\tl_new:N \l__zrefclever_lastsep_tl
\tl_new:N \l__zrefclever_rangesep_tl
\tl_new:N \l__zrefclever_namefont_tl
\tl_new:N \l__zrefclever_reffont_tl
\tl_new:N \l__zrefclever_endrangefunc_tl
\tl_new:N \l__zrefclever_endrangeprop_tl
\bool_new:N \l__zrefclever_cap_bool
\bool_new:N \l__zrefclever_abbrev_bool
\bool_new:N \l__zrefclever_rangetopair_bool
```

(*End definition for* `\l__zrefclever_tpairsep_tl` *and others.*)

\l__zrefclever_refbounds_first_seq
\l__zrefclever_refbounds_first_sg_seq
\l__zrefclever_refbounds_first_pb_seq
\l__zrefclever_refbounds_first_rb_seq
\l__zrefclever_refbounds_mid_seq
\l__zrefclever_refbounds_mid_rb_seq
\l__zrefclever_refbounds_mid_re_seq
\l__zrefclever_refbounds_last_seq
\l__zrefclever_refbounds_last_pe_seq
\l__zrefclever_refbounds_last_re_seq
\l__zrefclever_type_first_refbounds_seq
l__zrefclever_type_first_refbounds_set_bool

Auxiliary variables for `\__zrefclever_typeset_refs:`: advanced reference format options.

```
\seq_new:N \l__zrefclever_refbounds_first_seq
\seq_new:N \l__zrefclever_refbounds_first_sg_seq
\seq_new:N \l__zrefclever_refbounds_first_pb_seq
\seq_new:N \l__zrefclever_refbounds_first_rb_seq
\seq_new:N \l__zrefclever_refbounds_mid_seq
\seq_new:N \l__zrefclever_refbounds_mid_rb_seq
\seq_new:N \l__zrefclever_refbounds_mid_re_seq
\seq_new:N \l__zrefclever_refbounds_last_seq
\seq_new:N \l__zrefclever_refbounds_last_pe_seq
\seq_new:N \l__zrefclever_refbounds_last_re_seq
\seq_new:N \l__zrefclever_type_first_refbounds_seq
\bool_new:N \l__zrefclever_type_first_refbounds_set_bool
```

(*End definition for* `\l__zrefclever_refbounds_first_seq` *and others.*)

\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`.

```
\bool_new:N \l__zrefclever_verbose_testing_bool
```

(*End definition for* `\l__zrefclever_verbose_testing_bool`.)

Main functions

__zrefclever_typeset_refs:

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
    \tl_clear:N \l__zrefclever_range_end_ref_tl
    \int_zero:N \l__zrefclever_label_count_int
    \int_zero:N \l__zrefclever_type_count_int
    \int_zero:N \l__zrefclever_ref_count_int
    \int_zero:N \l__zrefclever_range_count_int
    \int_zero:N \l__zrefclever_range_same_count_int
    \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
    \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool

    % Get type block options (not type-specific).
    \__zrefclever_get_rf_opt_tl:nxxN { tpairsep }
      { \l__zrefclever_label_type_a_tl }
      { \l__zrefclever_ref_language_tl }
      \l__zrefclever_tpairsep_tl
    \__zrefclever_get_rf_opt_tl:nxxN { tlistsep }
      { \l__zrefclever_label_type_a_tl }
      { \l__zrefclever_ref_language_tl }
      \l__zrefclever_tlistsep_tl
    \__zrefclever_get_rf_opt_tl:nxxN { tlastsep }
      { \l__zrefclever_label_type_a_tl }
      { \l__zrefclever_ref_language_tl }
      \l__zrefclever_tlastsep_tl

    % Process label stack.
    \bool_set_false:N \l__zrefclever_typeset_last_bool
    \bool_until_do:Nn \l__zrefclever_typeset_last_bool
      {
        \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
          \l__zrefclever_label_a_tl
        \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
          {
            \tl_clear:N \l__zrefclever_label_b_tl
            \bool_set_true:N \l__zrefclever_typeset_last_bool
          }
          {
            \seq_get_left:NN \l__zrefclever_typeset_labels_seq
              \l__zrefclever_label_b_tl
          }

        \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
          {
            \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
            \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
          }
          {
            \__zrefclever_extract_default:NVnn
              \l__zrefclever_label_type_a_tl
              \l__zrefclever_label_a_tl { zc@type } { zc@missingtype }
            \__zrefclever_extract_default:NVnn
              \l__zrefclever_label_type_b_tl
              \l__zrefclever_label_b_tl { zc@type } { zc@missingtype }
          }

        % 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 }
                  { \bool_set_true:N \l__zrefclever_last_of_type_bool }
                  {
                    % Neither is undefined, we must check the types.
                    \tl_if_eq:NNTF
                      \l__zrefclever_label_type_a_tl
                      \l__zrefclever_label_type_b_tl
                      { \bool_set_false: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 }
          {}
          {
            \tl_if_eq:NnT \l__zrefclever_label_type_a_tl { zc@missingtype }
              {
                \msg_warning:nnx { zref-clever } { missing-type }
                  { \l__zrefclever_label_a_tl }
              }
            \zref@ifrefcontainsprop
              { \l__zrefclever_label_a_tl }
              { \l__zrefclever_ref_property_tl }
              { }
              {
                \msg_warning:nnxx { zref-clever } { missing-property }
                  { \l__zrefclever_ref_property_tl }
                  { \l__zrefclever_label_a_tl }
              }
          }

        % Get possibly type-specific separators, refbounds, font and other
        % options, once per type.
        \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
```

```
        {
          \__zrefclever_get_rf_opt_tl:nxxN { namesep }
            { \l__zrefclever_label_type_a_tl }
            { \l__zrefclever_ref_language_tl }
            \l__zrefclever_namesep_tl
          \__zrefclever_get_rf_opt_tl:nxxN { pairsep }
            { \l__zrefclever_label_type_a_tl }
            { \l__zrefclever_ref_language_tl }
            \l__zrefclever_pairsep_tl
          \__zrefclever_get_rf_opt_tl:nxxN { listsep }
            { \l__zrefclever_label_type_a_tl }
            { \l__zrefclever_ref_language_tl }
            \l__zrefclever_listsep_tl
          \__zrefclever_get_rf_opt_tl:nxxN { lastsep }
            { \l__zrefclever_label_type_a_tl }
            { \l__zrefclever_ref_language_tl }
            \l__zrefclever_lastsep_tl
          \__zrefclever_get_rf_opt_tl:nxxN { rangesep }
            { \l__zrefclever_label_type_a_tl }
            { \l__zrefclever_ref_language_tl }
            \l__zrefclever_rangesep_tl
          \__zrefclever_get_rf_opt_tl:nxxN { namefont }
            { \l__zrefclever_label_type_a_tl }
            { \l__zrefclever_ref_language_tl }
            \l__zrefclever_namefont_tl
          \__zrefclever_get_rf_opt_tl:nxxN { reffont }
            { \l__zrefclever_label_type_a_tl }
            { \l__zrefclever_ref_language_tl }
            \l__zrefclever_reffont_tl
          \__zrefclever_get_rf_opt_tl:nxxN { endrangefunc }
            { \l__zrefclever_label_type_a_tl }
            { \l__zrefclever_ref_language_tl }
            \l__zrefclever_endrangefunc_tl
          \__zrefclever_get_rf_opt_tl:nxxN { endrangeprop }
            { \l__zrefclever_label_type_a_tl }
            { \l__zrefclever_ref_language_tl }
            \l__zrefclever_endrangeprop_tl
          \__zrefclever_get_rf_opt_bool:nnxxN { cap } { false }
            { \l__zrefclever_label_type_a_tl }
            { \l__zrefclever_ref_language_tl }
            \l__zrefclever_cap_bool
          \__zrefclever_get_rf_opt_bool:nnxxN { abbrev } { false }
            { \l__zrefclever_label_type_a_tl }
            { \l__zrefclever_ref_language_tl }
            \l__zrefclever_abbrev_bool
          \__zrefclever_get_rf_opt_bool:nnxxN { rangetopair } { true }
            { \l__zrefclever_label_type_a_tl }
            { \l__zrefclever_ref_language_tl }
            \l__zrefclever_rangetopair_bool
          \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first }
            { \l__zrefclever_label_type_a_tl }
            { \l__zrefclever_ref_language_tl }
            \l__zrefclever_refbounds_first_seq
          \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-sg }
```

```
              { \l__zrefclever_label_type_a_tl }
              { \l__zrefclever_ref_language_tl }
              \l__zrefclever_refbounds_first_sg_seq
            \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-pb }
              { \l__zrefclever_label_type_a_tl }
              { \l__zrefclever_ref_language_tl }
              \l__zrefclever_refbounds_first_pb_seq
            \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-rb }
              { \l__zrefclever_label_type_a_tl }
              { \l__zrefclever_ref_language_tl }
              \l__zrefclever_refbounds_first_rb_seq
            \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid }
              { \l__zrefclever_label_type_a_tl }
              { \l__zrefclever_ref_language_tl }
              \l__zrefclever_refbounds_mid_seq
            \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-rb }
              { \l__zrefclever_label_type_a_tl }
              { \l__zrefclever_ref_language_tl }
              \l__zrefclever_refbounds_mid_rb_seq
            \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-re }
              { \l__zrefclever_label_type_a_tl }
              { \l__zrefclever_ref_language_tl }
              \l__zrefclever_refbounds_mid_re_seq
            \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last }
              { \l__zrefclever_label_type_a_tl }
              { \l__zrefclever_ref_language_tl }
              \l__zrefclever_refbounds_last_seq
            \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-pe }
              { \l__zrefclever_label_type_a_tl }
              { \l__zrefclever_ref_language_tl }
              \l__zrefclever_refbounds_last_pe_seq
            \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-re }
              { \l__zrefclever_label_type_a_tl }
              { \l__zrefclever_ref_language_tl }
              \l__zrefclever_refbounds_last_re_seq
          }

        % Here we send this to a couple of auxiliary functions.
        \bool_if:NTF \l__zrefclever_last_of_type_bool
          % There exists no next label of the same type as the current.
          { \__zrefclever_typeset_refs_last_of_type: }
          % There exists a next label of the same type as the current.
          { \__zrefclever_typeset_refs_not_last_of_type: }
      }
  }
```

(*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
          \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
            \l__zrefclever_refbounds_first_sg_seq
          \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
        }

        % The last is the second: we have a pair (if not repeated).
        % Test: 'zc-typeset01.lvt': "Last of type: pair"
        { 1 }
        {
          \int_compare:nNnTF { \l__zrefclever_range_same_count_int } = { 1 }
            {
              \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
                \l__zrefclever_refbounds_first_sg_seq
              \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
            }
            {
              \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                {
                  \exp_not:V \l__zrefclever_pairsep_tl
                  \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
                    \l__zrefclever_refbounds_last_pe_seq
                }
              \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
                \l__zrefclever_refbounds_first_pb_seq
              \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
            }
        }
      }
      % 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:nnF { \l__zrefclever_range_count_int }
          {
            % There was no range going on.
            % Test: 'zc-typeset01.lvt': "Last of type: not range"
            { 0 }
            {
              \int_compare:nNnTF { \l__zrefclever_ref_count_int } < { 2 }
                {
                  \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                    {
                      \exp_not:V \l__zrefclever_pairsep_tl
                      \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
                        \l__zrefclever_refbounds_last_pe_seq
                    }
                }
                {
                  \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                    {
                      \exp_not:V \l__zrefclever_lastsep_tl
                      \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
                        \l__zrefclever_refbounds_last_seq
                    }
                }
            }
            % Last in the range is also the second in it.
            % Test: 'zc-typeset01.lvt': "Last of type: pair in sequence"
            { 1 }
            {
              \int_compare:nNnTF
                { \l__zrefclever_range_same_count_int } = { 1 }
                {
                  % We know 'range_beg_is_first_bool' is false, since this is
                  % the second element in the range, but the third or more in
                  % the type list.
                  \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                    {
                      \exp_not:V \l__zrefclever_pairsep_tl
                      \__zrefclever_get_ref:VN
                        \l__zrefclever_range_beg_label_tl
                        \l__zrefclever_refbounds_last_pe_seq
                    }
                  \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
                    \l__zrefclever_refbounds_first_pb_seq
                  \bool_set_true:N
                    \l__zrefclever_type_first_refbounds_set_bool
                }
                {
                  \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                    {
                      \exp_not:V \l__zrefclever_listsep_tl
                      \__zrefclever_get_ref:VN
                        \l__zrefclever_range_beg_label_tl
                        \l__zrefclever_refbounds_mid_seq
                      \exp_not:V \l__zrefclever_lastsep_tl
                      \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
                        \l__zrefclever_refbounds_last_seq
                    }
                }
            }
          }
          % 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_is_first_bool’ is true, it means it was also
                  % the first of the type, and hence its typesetting was
                  % already handled, and we just have to set refbounds.
                  \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
                    {
                      \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
                        \l__zrefclever_refbounds_first_sg_seq
                      \bool_set_true:N
                        \l__zrefclever_type_first_refbounds_set_bool
                    }
                    {
                      \int_compare:nNnTF
                        { \l__zrefclever_ref_count_int } < { 2 }
                        {
                          \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                            {
                              \exp_not:V \l__zrefclever_pairsep_tl
                              \__zrefclever_get_ref:VN
                                \l__zrefclever_range_beg_label_tl
                                \l__zrefclever_refbounds_last_pe_seq
                            }
                        }
                        {
                          \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                            {
                              \exp_not:V \l__zrefclever_lastsep_tl
                              \__zrefclever_get_ref:VN
                                \l__zrefclever_range_beg_label_tl
                                \l__zrefclever_refbounds_last_seq
                            }
                        }
                    }
                }
                % A ‘range’, but with no skipped value, treat as pair if range
                % started with first of type, otherwise as list.
                % Test: ‘zc-typeset01.lvt’: "Last of type: range to pair"
                { 1 }
                {
                  % Ditto.
                  \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
                    {
```

```
            \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
              \l__zrefclever_refbounds_first_pb_seq
            \bool_set_true:N
              \l__zrefclever_type_first_refbounds_set_bool
            \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
              {
                \exp_not:V \l__zrefclever_pairsep_tl
                \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
                  \l__zrefclever_refbounds_last_pe_seq
              }
          }
          {
            \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
              {
                \exp_not:V \l__zrefclever_listsep_tl
                \__zrefclever_get_ref:VN
                  \l__zrefclever_range_beg_label_tl
                  \l__zrefclever_refbounds_mid_seq
              }
            \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
              {
                \exp_not:V \l__zrefclever_lastsep_tl
                \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
                  \l__zrefclever_refbounds_last_seq
              }
          }
      }
    }
    {
      % An actual range.
      % Test: ‘zc-typeset01.lvt’: "Last of type: range"
      % Ditto.
      \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
        {
          \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
            \l__zrefclever_refbounds_first_rb_seq
          \bool_set_true:N
            \l__zrefclever_type_first_refbounds_set_bool
        }
        {
          \int_compare:nNnTF
            { \l__zrefclever_ref_count_int } < { 2 }
            {
              \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                {
                  \exp_not:V \l__zrefclever_pairsep_tl
                  \__zrefclever_get_ref:VN
                    \l__zrefclever_range_beg_label_tl
                    \l__zrefclever_refbounds_mid_rb_seq
                }
              \seq_set_eq:NN
                \l__zrefclever_type_first_refbounds_seq
                \l__zrefclever_refbounds_first_pb_seq
              \bool_set_true:N
```

```
                        \l__zrefclever_type_first_refbounds_set_bool
                    }
                    {
                      \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                        {
                          \exp_not:V \l__zrefclever_lastsep_tl
                          \__zrefclever_get_ref:VN
                            \l__zrefclever_range_beg_label_tl
                            \l__zrefclever_refbounds_mid_rb_seq
                        }
                    }
                }
              \bool_lazy_and:nnTF
                { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
                { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
                {
                  \use:c { \l__zrefclever_endrangefunc_tl :VVN }
                    \l__zrefclever_range_beg_label_tl
                    \l__zrefclever_label_a_tl
                    \l__zrefclever_range_end_ref_tl
                  \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                    {
                      \exp_not:V \l__zrefclever_rangesep_tl
                      \__zrefclever_get_ref_endrange:VVN
                        \l__zrefclever_label_a_tl
                        \l__zrefclever_range_end_ref_tl
                        \l__zrefclever_refbounds_last_re_seq
                    }
                }
                {
                  \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                    {
                      \exp_not:V \l__zrefclever_rangesep_tl
                      \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
                        \l__zrefclever_refbounds_last_re_seq
                    }
                }
            }
        }
    }

  % 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:nnx { zref-clever } { single-element-range }
                { \l__zrefclever_type_first_label_type_tl }
            }
        }
        {
          \bool_set_false:N \l__zrefclever_next_maybe_range_bool
          \bool_if:NT \l__zrefclever_rangetopair_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"
          \bool_if:NTF \l__zrefclever_next_maybe_range_bool
            {
              \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
                {
                  \exp_not:V \l__zrefclever_pairsep_tl
                  \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
                    \l__zrefclever_refbounds_last_pe_seq
                }
              \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
                \l__zrefclever_refbounds_first_pb_seq
              \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
            }
            {
              \bool_lazy_and:nnTF
                { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
                { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
                {
                  % We must get ‘type_first_label_tl’ instead of
                  % ‘range_beg_label_tl’ here, since it is not necessary
                  % that the first of type was actually starting a range for
                  % the ‘range’ option to be used.
                  \use:c { \l__zrefclever_endrangefunc_tl :VVN }
                    \l__zrefclever_type_first_label_tl
                    \l__zrefclever_label_a_tl
                    \l__zrefclever_range_end_ref_tl
                  \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
                    {
                      \exp_not:V \l__zrefclever_rangesep_tl
                      \__zrefclever_get_ref_endrange:VVN
                        \l__zrefclever_label_a_tl
                        \l__zrefclever_range_end_ref_tl
                        \l__zrefclever_refbounds_last_re_seq
                    }
                }
                {
                  \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
                    {
                      \exp_not:V \l__zrefclever_rangesep_tl
            \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
              \l__zrefclever_refbounds_last_re_seq
          }
        }
      \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
        \l__zrefclever_refbounds_first_rb_seq
      \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
    }
  }
  }

  % If none of the special cases for the first of type refbounds have been
  % set, do it.
  \bool_if:NF \l__zrefclever_type_first_refbounds_set_bool
    {
      \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
        \l__zrefclever_refbounds_first_seq
    }

  % 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
        { \__zrefclever_get_ref_first: }
    }
    {
      \bool_if:NTF \l__zrefclever_typeset_ref_bool
        {
          % Test: ‘zc-typeset01.lvt’: "Last of type: option typeset ref"
          \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
            {
              \__zrefclever_get_ref:VN \l__zrefclever_type_first_label_tl
                \l__zrefclever_type_first_refbounds_seq
            }
        }
        {
          \bool_if:NTF \l__zrefclever_typeset_name_bool
            {
              % Test: ‘zc-typeset01.lvt’: "Last of type: option typeset name"
              \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
                      \__zrefclever_hyperlink:nnn
                        {
                          \__zrefclever_extract_url_unexp:V
                            \l__zrefclever_type_first_label_tl
                        }
```

```
             {
              \__zrefclever_extract_unexp:Vnn
               \l__zrefclever_type_first_label_tl
               { anchor } { }
             }
             { \exp_not:V \l__zrefclever_type_name_tl }
            \exp_not:N \group_end:
           }
           {
            \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:
           }
         }
       }
       {
        % 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_curr_tl }

  % Wrap up loop, or prepare for next iteration.
  \bool_if:NTF \l__zrefclever_typeset_last_bool
   {
    % We are finishing, typeset the current queue.
    \int_case:nnF { \l__zrefclever_type_count_int }
     {
      % Single type.
      % Test: ‘zc-typeset01.lvt’: "Last of type: single type"
      { 0 }
      { \l__zrefclever_typeset_queue_curr_tl }
      % Pair of types.
      % Test: ‘zc-typeset01.lvt’: "Last of type: pair of types"
      { 1 }
      {
              \l__zrefclever_tpairsep_tl
              \l__zrefclever_typeset_queue_curr_tl
            }
          }
          {
            % Last in list of types.
            % Test: ‘zc-typeset01.lvt’: "Last of type: list of types"
            \l__zrefclever_tlastsep_tl
            \l__zrefclever_typeset_queue_curr_tl
          }
        % And nudge in case of multitype reference.
        \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_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
          \l__zrefclever_typeset_queue_curr_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
        \tl_clear:N \l__zrefclever_range_end_ref_tl
        \int_zero:N \l__zrefclever_label_count_int
        \int_zero:N \l__zrefclever_ref_count_int
        \int_incr:N \l__zrefclever_type_count_int
        \int_zero:N \l__zrefclever_range_count_int
        \int_zero:N \l__zrefclever_range_same_count_int
        \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
        \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
      }
  }
```

(*End definition for* `\__zrefclever_typeset_refs_last_of_type:`.)

__zrefclever_typeset_refs_not_last_of_type: Handles typesetting when the current label is not the last of its type.

```
\cs_new_protected:Npn \__zrefclever_typeset_refs_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
        \int_incr:N \l__zrefclever_ref_count_int

        % If the next label may be part of a range, signal it (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_next_maybe_range_bool
          {
            \bool_set_true:N \l__zrefclever_range_beg_is_first_bool
            \tl_set:NV \l__zrefclever_range_beg_label_tl
              \l__zrefclever_label_a_tl
            \tl_clear:N \l__zrefclever_range_end_ref_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 }
          }
      }
      {
        % Current label is neither the first (nor the last) of its type.
        \bool_if:NTF \l__zrefclever_next_maybe_range_bool
          {
            % Starting, or continuing a range.
            \int_compare:nNnTF
              { \l__zrefclever_range_count_int } = { 0 }
              {
                % There was no range going, we are starting one.
                \tl_set:NV \l__zrefclever_range_beg_label_tl
                  \l__zrefclever_label_a_tl
                \tl_clear:N \l__zrefclever_range_end_ref_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 }
              }
              {
                % Second or more in the range, but not the last.
                \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 }
              }
          }
          {
            % Next element is not in sequence: there was no range, or we are
```

```
            % closing one.
            \int_case:nnF { \l__zrefclever_range_count_int }
              {
                % There was no range going on.
                % Test: ‘zc-typeset01.lvt’: "Not last of type: no range"
                { 0 }
                {
                  \int_incr:N \l__zrefclever_ref_count_int
                  \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                    {
                      \exp_not:V \l__zrefclever_listsep_tl
                      \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
                        \l__zrefclever_refbounds_mid_seq
                    }
                }
                % Last is second in the range: if ‘range_same_count’ is also
                % ‘1’, it’s a repetition (drop it), otherwise, it’s a "pair
                % within a list", treat as list.
                % Test: ‘zc-typeset01.lvt’: "Not last of type: range pair to one"
                % Test: ‘zc-typeset01.lvt’: "Not last of type: range pair"
                { 1 }
                {
                  \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
                    {
                      \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
                        \l__zrefclever_refbounds_first_seq
                      \bool_set_true:N
                        \l__zrefclever_type_first_refbounds_set_bool
                    }
                    {
                      \int_incr:N \l__zrefclever_ref_count_int
                      \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                        {
                          \exp_not:V \l__zrefclever_listsep_tl
                          \__zrefclever_get_ref:VN
                            \l__zrefclever_range_beg_label_tl
                            \l__zrefclever_refbounds_mid_seq
                        }
                    }
                  \int_compare:nNnF
                    { \l__zrefclever_range_same_count_int } = { 1 }
                    {
                      \int_incr:N \l__zrefclever_ref_count_int
                      \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                        {
                          \exp_not:V \l__zrefclever_listsep_tl
                          \__zrefclever_get_ref:VN
                            \l__zrefclever_label_a_tl
                            \l__zrefclever_refbounds_mid_seq
                        }
                    }
                }
              }
              {
```

```
            % 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 }
                {
                  \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
                    {
                      \seq_set_eq:NN
                        \l__zrefclever_type_first_refbounds_seq
                        \l__zrefclever_refbounds_first_seq
                      \bool_set_true:N
                        \l__zrefclever_type_first_refbounds_set_bool
                    }
                    {
                      \int_incr:N \l__zrefclever_ref_count_int
                      \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                        {
                          \exp_not:V \l__zrefclever_listsep_tl
                          \__zrefclever_get_ref:VN
                            \l__zrefclever_range_beg_label_tl
                            \l__zrefclever_refbounds_mid_seq
                        }
                    }
                }
                % Test: ‘zc-typeset01.lvt’: "Not last of type: range to pair"
                { 1 }
                {
                  \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
                    {
                      \seq_set_eq:NN
                        \l__zrefclever_type_first_refbounds_seq
                        \l__zrefclever_refbounds_first_seq
                      \bool_set_true:N
                        \l__zrefclever_type_first_refbounds_set_bool
                    }
                    {
                      \int_incr:N \l__zrefclever_ref_count_int
                      \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                        {
                          \exp_not:V \l__zrefclever_listsep_tl
                          \__zrefclever_get_ref:VN
                            \l__zrefclever_range_beg_label_tl
                            \l__zrefclever_refbounds_mid_seq
                        }
                    }
                  \int_incr:N \l__zrefclever_ref_count_int
                  \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                    {
                      \exp_not:V \l__zrefclever_listsep_tl
                      \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
                        \l__zrefclever_refbounds_mid_seq
                    }
                }
            }
            {
              % Test: ‘zc-typeset01.lvt’: "Not last of type: range"
              \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
                {
                  \seq_set_eq:NN
                    \l__zrefclever_type_first_refbounds_seq
                    \l__zrefclever_refbounds_first_rb_seq
                  \bool_set_true:N
                    \l__zrefclever_type_first_refbounds_set_bool
                }
                {
                  \int_incr:N \l__zrefclever_ref_count_int
                  \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                    {
                      \exp_not:V \l__zrefclever_listsep_tl
                      \__zrefclever_get_ref:VN
                        \l__zrefclever_range_beg_label_tl
                        \l__zrefclever_refbounds_mid_rb_seq
                    }
                }
              % For the purposes of the serial comma, and thus for the
              % distinction of ‘lastsep’ and ‘pairsep’, a "range" counts
              % as one.  Since ‘range_beg’ has already been counted
              % (here or with the first of type), we refrain from
              % incrementing ‘ref_count_int’.
              \bool_lazy_and:nnTF
                { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
                { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
                {
                  \use:c { \l__zrefclever_endrangefunc_tl :VVN }
                    \l__zrefclever_range_beg_label_tl
                    \l__zrefclever_label_a_tl
                    \l__zrefclever_range_end_ref_tl
                  \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                    {
                      \exp_not:V \l__zrefclever_rangesep_tl
                      \__zrefclever_get_ref_endrange:VVN
                        \l__zrefclever_label_a_tl
                        \l__zrefclever_range_end_ref_tl
                        \l__zrefclever_refbounds_mid_re_seq
                    }
                }
                {
                  \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                    {
                      \exp_not:V \l__zrefclever_rangesep_tl
```

```
                              \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
                                \l__zrefclever_refbounds_mid_re_seq
                            }
                        }
                    }
                }
              % We just closed a range, reset ‘range_beg_is_first’ in case a
              % second range for the same type occurs, in which case its
              % ‘range_beg’ will no longer be ‘first’.
              \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
              % Reset counters.
              \int_zero:N \l__zrefclever_range_count_int
              \int_zero:N \l__zrefclever_range_same_count_int
            }
        }
      % Step label counter for next iteration.
      \int_incr:N \l__zrefclever_label_count_int
    }
```

(*End definition for* `\__zrefclever_typeset_refs_not_last_of_type:`*.*)

Auxiliary functions

`\__zrefclever_get_ref:nN` and `\__zrefclever_get_ref_first:` are the two functions which actually build the reference blocks for typesetting. `\__zrefclever_get_ref:nN` 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:nN` 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 `n` 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.

```
\cs_new_protected:Npn \__zrefclever_ref_default:
  { \zref@default }
\cs_new_protected:Npn \__zrefclever_name_default:
  { \zref@default }
```

(*End definition for* `\__zrefclever_ref_default:` *and* `\__zrefclever_name_default:`*.*)

`\__zrefclever_get_ref:nN` Handles a complete reference block to be accumulated in the "queue", including refbounds, and hyperlinking. For use with all labels, except the first of its type, which is done by `\__zrefclever_get_ref_first:`, and the last of a range, which is done by `\__zrefclever_get_ref_endrange:nnN`.

```
\__zrefclever_get_ref:nN {⟨label⟩} {⟨refbounds⟩}
```

```
\cs_new:Npn \__zrefclever_get_ref:nN #1#2
  {
    \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
      {
        \bool_if:nTF
          {
            \l__zrefclever_hyperlink_bool &&
            ! \l__zrefclever_link_star_bool
          }
          {
            \seq_item:Nn #2 { 1 }
            \__zrefclever_hyperlink:nnn
              { \__zrefclever_extract_url_unexp:n {#1} }
              { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
              {
                \seq_item:Nn #2 { 2 }
                \exp_not:N \group_begin:
                \exp_not:V \l__zrefclever_reffont_tl
                \__zrefclever_extract_unexp:nvn {#1}
                  { l__zrefclever_ref_property_tl } { }
                \exp_not:N \group_end:
                \seq_item:Nn #2 { 3 }
              }
            \seq_item:Nn #2 { 4 }
          }
          {
            \seq_item:Nn #2 { 1 }
            \seq_item:Nn #2 { 2 }
            \exp_not:N \group_begin:
            \exp_not:V \l__zrefclever_reffont_tl
            \__zrefclever_extract_unexp:nvn {#1}
              { l__zrefclever_ref_property_tl } { }
            \exp_not:N \group_end:
            \seq_item:Nn #2 { 3 }
            \seq_item:Nn #2 { 4 }
          }
      }
      { \__zrefclever_ref_default: }
  }
\cs_generate_variant:Nn \__zrefclever_get_ref:nN { VN }
```

(*End definition for* `\__zrefclever_get_ref:nN`*.*)

`\__zrefclever_get_ref_endrange:nnN`

```
\__zrefclever_get_ref_endrange:nnN {⟨label⟩} {⟨reference⟩} {⟨refbounds⟩}
```

```
\cs_new:Npn \__zrefclever_get_ref_endrange:nnN #1#2#3
```

```
  {
    \str_if_eq:nnTF {#2} { zc@missingproperty }
      { \__zrefclever_ref_default: }
      {
        \bool_if:nTF
          {
            \l__zrefclever_hyperlink_bool &&
            ! \l__zrefclever_link_star_bool
          }
          {
            \seq_item:Nn #3 { 1 }
            \__zrefclever_hyperlink:nnn
              { \__zrefclever_extract_url_unexp:n {#1} }
              { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
              {
                \seq_item:Nn #3 { 2 }
                \exp_not:N \group_begin:
                \exp_not:V \l__zrefclever_reffont_tl
                \exp_not:n {#2}
                \exp_not:N \group_end:
                \seq_item:Nn #3 { 3 }
              }
            \seq_item:Nn #3 { 4 }
          }
          {
            \seq_item:Nn #3 { 1 }
            \seq_item:Nn #3 { 2 }
            \exp_not:N \group_begin:
            \exp_not:V \l__zrefclever_reffont_tl
            \exp_not:n {#2}
            \exp_not:N \group_end:
            \seq_item:Nn #3 { 3 }
            \seq_item:Nn #3 { 4 }
          }
      }
  }
\cs_generate_variant:Nn \__zrefclever_get_ref_endrange:nnN { VVN }
```

(*End definition for* `\__zrefclever_get_ref_endrange:nnN`.)

`\__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.

```
\cs_new:Npn \__zrefclever_get_ref_first:
  {
    \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
      { \__zrefclever_ref_default: }
      {
        \bool_if:NTF \l__zrefclever_name_in_link_bool
```

```
            {
              \zref@ifrefcontainsprop
                { \l__zrefclever_type_first_label_tl }
                { \l__zrefclever_ref_property_tl }
                {
                  \__zrefclever_hyperlink:nnn
                    {
                      \__zrefclever_extract_url_unexp:V
                        \l__zrefclever_type_first_label_tl
                    }
                    {
                      \__zrefclever_extract_unexp:Vnn
                        \l__zrefclever_type_first_label_tl { anchor } { }
                    }
                    {
                      \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
                      \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
                      \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
                      \exp_not:N \group_begin:
                      \exp_not:V \l__zrefclever_reffont_tl
                      \__zrefclever_extract_unexp:Vvn
                        \l__zrefclever_type_first_label_tl
                        { l__zrefclever_ref_property_tl } { }
                      \exp_not:N \group_end:
                      \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
                    }
                  \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
                }
                {
                  \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
                  \__zrefclever_ref_default:
                }
            }
            {
              \bool_if:nTF \l__zrefclever_type_name_missing_bool
                {
                  \__zrefclever_name_default:
                  \exp_not:V \l__zrefclever_namesep_tl
                }
                {
                  \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:
                  \tl_if_empty:NF \l__zrefclever_type_name_tl
                    { \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_hyperlink_bool &&
         ! \l__zrefclever_link_star_bool
        }
        {
         \seq_item:Nn
          \l__zrefclever_type_first_refbounds_seq { 1 }
         \__zrefclever_hyperlink:nnn
          {
           \__zrefclever_extract_url_unexp:V
            \l__zrefclever_type_first_label_tl
          }
          {
           \__zrefclever_extract_unexp:Vnn
            \l__zrefclever_type_first_label_tl { anchor } { }
          }
          {
           \seq_item:Nn
            \l__zrefclever_type_first_refbounds_seq { 2 }
           \exp_not:N \group_begin:
           \exp_not:V \l__zrefclever_reffont_tl
           \__zrefclever_extract_unexp:Vvn
            \l__zrefclever_type_first_label_tl
            { l__zrefclever_ref_property_tl } { }
           \exp_not:N \group_end:
           \seq_item:Nn
            \l__zrefclever_type_first_refbounds_seq { 3 }
          }
         \seq_item:Nn
          \l__zrefclever_type_first_refbounds_seq { 4 }
        }
        {
         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
         \exp_not:N \group_begin:
         \exp_not:V \l__zrefclever_reffont_tl
         \__zrefclever_extract_unexp:Vvn
          \l__zrefclever_type_first_label_tl
          { l__zrefclever_ref_property_tl } { }
         \exp_not:N \group_end:
         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
        }
      }
      { \__zrefclever_ref_default: }
    }
  }
}
```

(*End definition for* `\__zrefclever_get_ref_first:`.)

`\__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 `\l__zrefclever_typeset_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:
  {
    \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
      {
        \tl_clear:N \l__zrefclever_type_name_tl
        \bool_set_true:N \l__zrefclever_type_name_missing_bool
      }
      {
        \tl_if_eq:NnTF
          \l__zrefclever_type_first_label_type_tl { zc@missingtype }
          {
            \tl_clear:N \l__zrefclever_type_name_tl
            \bool_set_true:N \l__zrefclever_type_name_missing_bool
          }
          {
            % Determine whether we should use capitalization, abbreviation,
            % and plural.
            \bool_lazy_or:nnTF
              { \l__zrefclever_cap_bool }
              {
                \l__zrefclever_capfirst_bool &&
                \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
              }
              { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
              { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
            % If the queue is empty, we have a singular, otherwise, plural.
            \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
              { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
              { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
            \bool_lazy_and:nnTF
              { \l__zrefclever_abbrev_bool }
              {
                ! \int_compare_p:nNn
                    { \l__zrefclever_type_count_int } = { 0 } ||
                ! \l__zrefclever_noabbrev_first_bool
              }
              {
                \tl_set:NV \l__zrefclever_name_format_fallback_tl
                  \l__zrefclever_name_format_tl
                \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
                      {
                        { \l__zrefclever_nudge_comptosing_bool }
                        { \tl_if_empty_p: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_rf_opt_seq:nxxN { gender }
                      { \l__zrefclever_type_first_label_type_tl }
                      { \l__zrefclever_ref_language_tl }
                      \l__zrefclever_type_name_gender_seq
                    \seq_if_in:NVF
                      \l__zrefclever_type_name_gender_seq
                      \l__zrefclever_ref_gender_tl
                      {
                        \seq_if_empty:NTF \l__zrefclever_type_name_gender_seq
                          {
                            \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_ref_language_tl }
                          }
                          {
                            \msg_warning:nnxxxx { zref-clever }
                              { nudge-gender-mismatch }
                              { \l__zrefclever_type_first_label_type_tl }
                              { \l__zrefclever_ref_gender_tl }
                              {
                                \seq_use:Nn
                                  \l__zrefclever_type_name_gender_seq { ,~ }
                              }
                              { \l__zrefclever_ref_language_tl }
                          }
                      }
                  }
              }

            \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
              {
                \__zrefclever_opt_tl_get:cNF
                  {
                    \__zrefclever_opt_varname_type:een
                      { \l__zrefclever_type_first_label_type_tl }
                      { \l__zrefclever_name_format_tl }
                      { 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
                      }
                    \__zrefclever_opt_tl_get:cNF
                      {
                        \__zrefclever_opt_varname_lang_type:eeen
                          { \l__zrefclever_ref_language_tl }
                          { \l__zrefclever_type_first_label_type_tl }
                          { \l__zrefclever_name_format_tl }
                          { tl }
                      }
                      \l__zrefclever_type_name_tl
                      {
                        \tl_clear:N \l__zrefclever_type_name_tl
                        \bool_set_true:N \l__zrefclever_type_name_missing_bool
                        \msg_warning:nnxx { zref-clever } { missing-name }
                          { \l__zrefclever_name_format_tl }
                          { \l__zrefclever_type_first_label_type_tl }
                      }
                  }
              }
              {
                \__zrefclever_opt_tl_get:cNF
                  {
                    \__zrefclever_opt_varname_type:een
                      { \l__zrefclever_type_first_label_type_tl }
                      { \l__zrefclever_name_format_tl }
                      { tl }
                  }
                  \l__zrefclever_type_name_tl
                  {
                    \__zrefclever_opt_tl_get:cNF
                      {
                        \__zrefclever_opt_varname_type:een
                          { \l__zrefclever_type_first_label_type_tl }
                          { \l__zrefclever_name_format_fallback_tl }
                          { 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
                          }
                        \__zrefclever_opt_tl_get:cNF
                          {
                            \__zrefclever_opt_varname_lang_type:eeen
                              { \l__zrefclever_ref_language_tl }
                              { \l__zrefclever_type_first_label_type_tl }
                              { \l__zrefclever_name_format_tl }
                              { tl }
                          }
                          \l__zrefclever_type_name_tl
                          {
                            \__zrefclever_opt_tl_get:cNF
                              {
                                \__zrefclever_opt_varname_lang_type:eeen
                                  { \l__zrefclever_ref_language_tl }
                                  { \l__zrefclever_type_first_label_type_tl }
                                  { \l__zrefclever_name_format_fallback_tl }
                                  { tl }
                              }
                              \l__zrefclever_type_name_tl
                              {
                                \tl_clear:N \l__zrefclever_type_name_tl
                                \bool_set_true:N
                                  \l__zrefclever_type_name_missing_bool
                                \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_hyperlink_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
          {
            { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
            {
              \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
              \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
            }
            {
              \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
              \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
              \l__zrefclever_typeset_last_bool &&
              \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
            }
          }
          { \bool_set_true:N \l__zrefclever_name_in_link_bool }
          { \bool_set_false:N \l__zrefclever_name_in_link_bool }
      }
  }
```

(*End definition for* `\__zrefclever_type_name_setup:`.)

`\__zrefclever_hyperlink:nnn`

This avoids using the internal `\hyper@@link`, using only public hyperref commands (see https://github.com/latex3/hyperref/issues/229#issuecomment-1093870142, thanks Ulrike Fisher).

`\__zrefclever_hyperlink:nnn {⟨url/file⟩} {⟨anchor⟩} {⟨text⟩}`

```
\cs_new_protected:Npn \__zrefclever_hyperlink:nnn #1#2#3
  {
    \tl_if_empty:nTF {#1}
      { \hyperlink {#2} {#3} }
      { \hyper@linkfile {#3} {#1} {#2} }
  }
```

(*End definition for* `\__zrefclever_hyperlink:nnn`.)

`\__zrefclever_extract_url_unexp:n`

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

```
\cs_new:Npn \__zrefclever_extract_url_unexp:n #1
  {
    \zref@ifpropundefined { urluse }
      { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
      {
        \zref@ifrefcontainsprop {#1} { urluse }
          { \__zrefclever_extract_unexp:nnn {#1} { urluse } { } }
          { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
      }
  }
\cs_generate_variant:Nn \__zrefclever_extract_url_unexp:n { V }
```

(*End definition for* `\__zrefclever_extract_url_unexp:n`.)

`\__zrefclever_labels_in_sequence:nn`

Auxiliary function to `\__zrefclever_typeset_refs_not_last_of_type:`. Sets `\l__zrefclever_next_maybe_range_bool` to true if ⟨*label b*⟩ comes in immediate sequence from ⟨*label a*⟩. And sets both `\l__zrefclever_next_maybe_range_bool` and `\l__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 {⟨label a⟩} {⟨label b⟩}`

```
\cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
  {
    \exp_args:Nxx \tl_if_eq:nnT
      { \__zrefclever_extract_unexp:nnn {#1} { externaldocument } { } }
      { \__zrefclever_extract_unexp:nnn {#2} { externaldocument } { } }
      {
        \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
          {
            \exp_args:Nxx \tl_if_eq:nnT
              { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
              { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
              {
                \int_compare:nNnTF
                  { \__zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
                    =
                  { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
                  { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
                  {
                    \int_compare:nNnT
                      { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
                        =
                      { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
                      {
                        \bool_set_true:N \l__zrefclever_next_maybe_range_bool
                        \bool_set_true:N \l__zrefclever_next_is_same_bool
                      }
                  }
              }
          }
          {
            \exp_args:Nxx \tl_if_eq:nnT
              { \__zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
              { \__zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
              {
                \exp_args:Nxx \tl_if_eq:nnT
                  { \__zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
                  { \__zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
                  {
                    \int_compare:nNnTF
                      { \__zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
                        =
                      { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
                      { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
                      {
                        \int_compare:nNnT
                          { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
                            =
                          { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
                          {
```

If `zc@counter`s are equal, `zc@enclval`s are equal, and `zc@enclval`s are equal, but the references themselves are different, this means that `\@currentlabel` has somehow been set manually (e.g. by an amsmath's `\tag`), in which case we have no idea what's in there, and we should not even consider this is still a range. If they are equal, though, of course it is a range, and it is the same.

```
                          \exp_args:Nxx \tl_if_eq:nnT
                            {
                              \__zrefclever_extract_unexp:nvn {#1}
                                { l__zrefclever_ref_property_tl } { }
                            }
                            {
                              \__zrefclever_extract_unexp:nvn {#2}
                                { l__zrefclever_ref_property_tl } { }
                            }
                            {
                              \bool_set_true:N
                                \l__zrefclever_next_maybe_range_bool
                              \bool_set_true:N
                                \l__zrefclever_next_is_same_bool
                            }
                        }
                    }
                }
            }
        }
    }
  }
```

(*End definition for* `\__zrefclever_labels_in_sequence:nn`.)

Finally, some functions for retrieving reference options values, according to the relevant precedence rules. They receive an ⟨*option*⟩ as argument, and store the retrieved value in an appropriate ⟨*variable*⟩. The difference between each of these functions is the data type of the option each should be used for.

__zrefclever_get_rf_opt_tl:nnnN

```
\__zrefclever_get_rf_opt_tl:nnnN {⟨option⟩}
  {⟨ref type⟩} {⟨language⟩} {⟨tl variable⟩}
```

```
\cs_new_protected:Npn \__zrefclever_get_rf_opt_tl:nnnN #1#2#3#4
  {
    % First attempt: general options.
    \__zrefclever_opt_tl_get:cNF
      { \__zrefclever_opt_varname_general:nn {#1} { tl } }
      #4
      {
        % If not found, try type specific options.
        \__zrefclever_opt_tl_get:cNF
          { \__zrefclever_opt_varname_type:nnn {#2} {#1} { tl } }
          #4
          {
            % If not found, try type- and language-specific.
            \__zrefclever_opt_tl_get:cNF
              { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { tl } }
              #4
              {
                % If not found, try language-specific default.
                \__zrefclever_opt_tl_get:cNF
                  { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { tl } }
                  #4
                  {
                    % If not found, try fallback.
                    \__zrefclever_opt_tl_get:cNF
                      { \__zrefclever_opt_varname_fallback:nn {#1} { tl } }
                      #4
                      { \tl_clear:N #4 }
                  }
              }
          }
      }
  }
\cs_generate_variant:Nn \__zrefclever_get_rf_opt_tl:nnnN { nxxN }
```

(*End definition for* `\__zrefclever_get_rf_opt_tl:nnnN`.)

__zrefclever_get_rf_opt_seq:nnnN

```
\__zrefclever_get_rf_opt_seq:nnnN {⟨option⟩}
  {⟨ref type⟩} {⟨language⟩} {⟨seq variable⟩}
```

```
\cs_new_protected:Npn \__zrefclever_get_rf_opt_seq:nnnN #1#2#3#4
  {
    % First attempt: general options.
    \__zrefclever_opt_seq_get:cNF
      { \__zrefclever_opt_varname_general:nn {#1} { seq } }
      #4
      {
        % If not found, try type specific options.
        \__zrefclever_opt_seq_get:cNF
          { \__zrefclever_opt_varname_type:nnn {#2} {#1} { seq } }
          #4
          {
            % If not found, try type- and language-specific.
            \__zrefclever_opt_seq_get:cNF
```

```
              { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { seq } }
              #4
              {
                % If not found, try language-specific default.
                \__zrefclever_opt_seq_get:cNF
                  { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { seq } }
                  #4
                  {
                    % If not found, try fallback.
                    \__zrefclever_opt_seq_get:cNF
                      { \__zrefclever_opt_varname_fallback:nn {#1} { seq } }
                      #4
                      { \seq_clear:N #4 }
                  }
              }
          }
      }
  }
\cs_generate_variant:Nn \__zrefclever_get_rf_opt_seq:nnnN { nxxN }
```

(*End definition for* `\__zrefclever_get_rf_opt_seq:nnnN`*.*)

`\__zrefclever_get_rf_opt_bool:nnnnN`

```
 \__zrefclever_get_rf_opt_bool:nN {⟨option⟩} {⟨default⟩}
   {⟨ref type⟩} {⟨language⟩}  {⟨bool variable⟩}
```

```
\cs_new_protected:Npn \__zrefclever_get_rf_opt_bool:nnnnN #1#2#3#4#5
  {
    % First attempt: general options.
    \__zrefclever_opt_bool_get:cNF
      { \__zrefclever_opt_varname_general:nn {#1} { bool } }
      #5
      {
        % If not found, try type specific options.
        \__zrefclever_opt_bool_get:cNF
          { \__zrefclever_opt_varname_type:nnn {#3} {#1} { bool } }
          #5
          {
            % If not found, try type- and language-specific.
            \__zrefclever_opt_bool_get:cNF
              { \__zrefclever_opt_varname_lang_type:nnnn {#4} {#3} {#1} { bool } }
              #5
              {
                % If not found, try language-specific default.
                \__zrefclever_opt_bool_get:cNF
                  { \__zrefclever_opt_varname_lang_default:nnn {#4} {#1} { bool } }
                  #5
                  {
                    % If not found, try fallback.
                    \__zrefclever_opt_bool_get:cNF
                      { \__zrefclever_opt_varname_fallback:nn {#1} { bool } }
                      #5
                      { \use:c { bool_set_ #2 :N } #5 }
                  }
              }
          }
    }
  }
\cs_generate_variant:Nn \__zrefclever_get_rf_opt_bool:nnnnN { nnxxN }
```

(*End definition for* `\__zrefclever_get_rf_opt_bool:nnnnN`*.*)

9 Compatibility

This section is meant to aggregate any "special handling" needed for LaTeX 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
          {
            countertype =
              {
                chapter       = appendix ,
                section       = appendix ,
                subsection    = appendix ,
                subsubsection = appendix ,
                paragraph     = appendix ,
                subparagraph  = appendix ,
              }
          }
      }
  }
```

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 `\scantokens` 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 }
      {
        \newcounter { zc@appendix }
        \newcounter { zc@save@appendix }
        \setcounter { zc@appendix } { 0 }
        \setcounter { zc@save@appendix } { 0 }
        \cs_if_exist:cTF { chapter }
          {
            \__zrefclever_zcsetup:n
              { counterresetby = { chapter = zc@appendix } }
          }
          {
            \cs_if_exist:cT { section }
              {
                \__zrefclever_zcsetup:n
                  { counterresetby = { section = zc@appendix } }
              }
          }
        \AddToHook { env / appendices / begin }
          {
            \stepcounter { zc@save@appendix }
            \setcounter { zc@appendix } { \value { zc@save@appendix } }
            \__zrefclever_zcsetup:n
              {
                countertype =
                  {
                    chapter       = appendix ,
                    section       = appendix ,
                    subsection    = appendix ,
                    subsubsection = appendix ,
                    paragraph     = appendix ,
                    subparagraph  = appendix ,
                  }
              }
          }
        \AddToHook { env / appendices / end }
          { \setcounter { zc@appendix } { 0 } }
        \AddToHook { cmd / appendix / before }
          {
            \stepcounter { zc@save@appendix }
            \setcounter { zc@appendix } { \value { zc@save@appendix } }
          }
        \AddToHook { env / subappendices / begin }
          {
            \__zrefclever_zcsetup:n
              {
                countertype =
                  {
                    section       = appendix ,
                    subsection    = appendix ,
                    subsubsection = appendix ,
                    paragraph     = appendix ,
                    subparagraph  = appendix ,
                  } ,
              }
          }
        \msg_info:nnn { zref-clever } { compat-package } { appendix }
      }
  }
```

9.3 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 ⟨*label*⟩. 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 ⟨*subtitle*⟩ argument of the subcaptioning commands, namely: `\subcaption`, `\contsubcaption`, `\subbottom`, `\contsubbottom`, `\subtop`. This functionality is implemented by memoir by setting a *second label* with prefix `sub@`⟨*label*⟩, 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 memoir does not really consider the possibility of some other command setting labels. `\@memsubcaption` 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. memoir'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 memoir's appendix features "emulates" the appendix package, hence the corresponding compatibility module is loaded for memoir 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 { memoir }
  {
    \__zrefclever_if_class_loaded:nT { memoir }
      {
```

Add subfigure and subtable support out of the box. Technically, this is not "default" behavior for memoir, 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 ,
              } ,
            counterresetby =
              {
                subfigure = figure ,
                subtable  = table ,
              } ,
          }
```

Support for caption memoir features that require that ⟨*label*⟩ 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 }
          { \__zrefclever_memoir_both_labels: }
        \AddToHook { env / sidecontcaption / begin }
          { \__zrefclever_memoir_both_labels: }
        \AddToHook{ cmd / bitwonumcaption / before }
          { \group_begin: \__zrefclever_memoir_both_labels: }
        \AddToHook{ cmd / bitwonumcaption / after }
          { \group_end: }
        \AddToHook{ cmd / bionenumcaption / before }
          { \group_begin: \__zrefclever_memoir_both_labels: }
        \AddToHook{ cmd / bionenumcaption / after }
          { \group_end: }
        \AddToHook{ cmd / bicaption / before }
          { \group_begin: \__zrefclever_memoir_both_labels: }
        \AddToHook{ cmd / bicaption / after }
          { \group_end: }
```

Support for `subcaption` reference.

```
        \zref@newprop { subcaption }
          { \cs_if_exist_use:c { @@thesub \@captype } }
        \AddToHook{ cmd / @memsubcaption / before }
          { \zref@localaddprop \ZREF@mainlist { subcaption } }
```

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 }
          { \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { mpfootnote } }
        \AddToHook { cmd / @makefntext / before }
          {
            \__zrefclever_zcsetup:x
              { currentcounter = \l__zrefclever_memoir_footnote_type_tl }
          }
        \AddToHook { cmd / @makesidefntext / before }
          { \__zrefclever_zcsetup:n { currentcounter = sidefootnote } }
        \__zrefclever_zcsetup:n
          {
            countertype =
              {
                sidefootnote = footnote ,
                pagenote = endnote ,
              } ,
          }
        \AddToHook { file / \jobname.ent / before }
          { \__zrefclever_zcsetup:x { currentcounter = pagenote } }
        \msg_info:nnn { zref-clever } { compat-class } { memoir }
      }
  }
```

9.4 amsmath

About this, see https://tex.stackexchange.com/a/402297.

```
\__zrefclever_compat_module:nn { amsmath }
```

```
    {
      \__zrefclever_if_package_loaded:nT { amsmath }
        {
```

First, we define a function for label setting inside 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 *must* use the lower level `\zref@label` in this context, and hence also handle protection with `\zref@wrapper@babel`, because `\zlabel` makes itself no-op when `\label` is equal to `\ltx@gobble`, and that's precisely the case inside the `multline` environment (and, damn!, I took a beating of this detail...). See also `https://github.com/ho-tex/zref/issues/4` and `https://github.com/latex3/latex2e/issues/272`.

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

Then we must store the original value of `\ltx@label`, which is the macro actually responsible for setting the labels inside amsmath's math environments. And, after that, redefine it to be `\__zrefclever_ltxlabel:n` instead. We must handle hyperref here, which comes very late in the preamble, and which loads nameref at `begindocument` (though this has changed recently 2022-05-16, see `https://github.com/latex3/hyperref/commit/a011ba9308a1b047dc151796de557da0bb22abaa`), which in turn, lets `\ltx@label` be `\label`. This has to come after nameref. Other classes packages also redefine `\ltx@label`, which may cause some trouble. A `grep` on `texmf-dist` returns hits for: `thm-restate.sty`, `smartref.sty`, `jmlrbook.cls`, `cleveref.sty`, `cryptocode.sty`, `nameref.sty`, `easyeqn.sty`, `empheq.sty`, `ntheorem.sty`, `nccmath.sty`, `nwejm.cls`, `nwejmart.cls`, `aguplus.sty`, `aguplus.cls`, `agupp.sty`, `amsmath.hyp`, `amsmath.sty` (surprise!), `amsmath.4ht`, `nameref.4ht`, `frenchle.sty`, `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 (amsmath itself, and nameref). A number of those I'm really not acquainted with. For 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 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 cleveref to load it last, or at least very late, and besides I don't see much of an use case for using both cleveref and 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 `https://github.com/latex3/hyperref/issues/212`.

```
          \__zrefclever_if_package_loaded:nTF { hyperref }
            {
              \AddToHook { package / nameref / after }
                {
                  \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
                  \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
                }
            }
            {
              \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
```

```
        \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
      }
```

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 refstepped, 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).

```
    \bool_new:N \l__zrefclever_amsmath_subequations_bool
    \AddToHook { env / subequations / begin }
      {
        \__zrefclever_zcsetup:x
          {
            counterresetby =
              {
                parentequation =
                  \__zrefclever_counter_reset_by:n { equation } ,
                equation = parentequation ,
              } ,
            currentcounter = parentequation ,
            countertype = { parentequation = equation } ,
          }
        \bool_set_true:N \l__zrefclever_amsmath_subequations_bool
      }
```

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. We also arrange, at this point, for the provision of the subeq property, for the convenience of referring to them directly or to build terse ranges with the endrange option.

```
    \zref@newprop { subeq } { \alph { equation } }
    \clist_map_inline:nn
      {
        equation ,
        equation* ,
        align ,
        align* ,
        alignat ,
        alignat* ,
        flalign ,
        flalign* ,
        xalignat ,
        xalignat* ,
        gather ,
        gather* ,
        multline ,
        multline* ,
      }
      {
        \AddToHook { env / #1 / begin }
          {
            \__zrefclever_zcsetup:n { currentcounter = equation }
            \bool_if:NT \l__zrefclever_amsmath_subequations_bool
              { \zref@localaddprop \ZREF@mainlist { subeq } }
          }
      }
    \msg_info:nnn { zref-clever } { compat-package } { amsmath }
  }
}
```

9.5 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.

```
\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 }
                      {
                        \protected@write \@auxout { }
                          { \string \MT@newlabel {##1} }
                      }
                      {
                        \exp_args:Nx \tl_if_eq:nnT
```

```
                              { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
                              { parentequation }
                              {
                                \protected@write \@auxout { }
                                  { \string \MT@newlabel {##1} }
                              }
                          }
                      }
                    \@esphack
                  }
                \msg_info:nnn { zref-clever } { compat-package } { mathtools }
              }
          }
      }
```

9.6 breqn

From the breqn documentation: "Use of the normal `\label` command instead of the `label` option works, I think, most of the time (untested)". Indeed, light testing suggests it does work for `\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 `label=` option without hacking the package internals, even if the case of doing so would not be specially tricky, just "not very civil".

```
\__zrefclever_compat_module:nn { breqn }
  {
    \__zrefclever_if_package_loaded:nT { breqn }
      {
```

Contrary to the practice in amsmath, which prints `\tag` even in unnumbered environments, the starred environments from breqn don't typeset any tag/number at all, even for a manually given `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 amsmath's practice, breqn uses `\stepcounter` instead of `\refstepcounter` for incrementing the equation counters (see https://tex.stackexchange.com/a/241150).

```
        \bool_new:N \l__zrefclever_breqn_dgroup_bool
        \AddToHook { env / dgroup / begin }
          {
            \__zrefclever_zcsetup:x
              {
                counterresetby =
                  {
                    parentequation =
                      \__zrefclever_counter_reset_by:n { equation } ,
                    equation = parentequation ,
                  } ,
                currentcounter = parentequation ,
                countertype = { parentequation = equation } ,
              }
            \bool_set_true:N \l__zrefclever_breqn_dgroup_bool
          }
        \zref@ifpropundefined { subeq }
          { \zref@newprop { subeq } { \alph { equation } } }
          { }
        \clist_map_inline:nn
          {
            dmath ,
            dseries ,
            darray ,
          }
          {
            \AddToHook { env / #1 / begin }
              {
                \__zrefclever_zcsetup:n { currentcounter = equation }
                \bool_if:NT \l__zrefclever_breqn_dgroup_bool
                  { \zref@localaddprop \ZREF@mainlist { subeq } }
              }
          }
        \msg_info:nnn { zref-clever } { compat-package } { breqn }
      }
  }
```

9.7 listings

```
\__zrefclever_compat_module:nn { listings }
  {
    \__zrefclever_if_package_loaded:nT { listings }
      {
        \__zrefclever_zcsetup:n
          {
            countertype =
              {
                lstlisting = listing ,
                lstnumber = line ,
              } ,
            counterresetby = { lstnumber = lstlisting } ,
          }
```

Set (also) a `\zlabel` with the label received in the `label=` option from the `lstlisting` environment. The *only* place to set this label is the `PreInit` hook. This hook, comes right after `\lst@MakeCaption` in `\lst@Init`, which runs `\refstepcounter` on `lstlisting`, so we must come after it. Also listings itself sets `\@currentlabel` to `\thelstnumber` in the `Init` hook, which comes right after the `PreInit` one in `\lst@Init`. Since, if we add to `Init` here, we go to the end of it, we'd be seeing the wrong `\@currentlabel` at that point.

```
        \lst@AddToHook { PreInit }
          { \tl_if_empty:NF \lst@label { \zlabel { \lst@label } } }
```

Set `currentcounter` to `lstnumber` in the `Init` hook, since listings itself sets `\@currentlabel` to `\thelstnumber` here. Note that listings *does use* `\refstepcounter` on `lstnumber`, but does so in the `EveryPar` hook, and there must be some grouping involved such that `\@currentcounter` ends up not being visible to the label. See section "Line numbers" of '`texdoc listings-devel`' (the `.dtx`), and search for the definition of macro `\c@lstnumber`. Indeed, the fact that listings manually sets `\@currentlabel` to `\thelstnumber` is a signal that the work of `\refstepcounter` is being restrained somehow.

```
        \lst@AddToHook { Init }
          { \__zrefclever_zcsetup:n { currentcounter = lstnumber } }
```

```
      \msg_info:nnn { zref-clever } { compat-package } { listings }
    }
  }
```

9.8 enumitem

The procedure below will "see" any changes made to the `enumerate` environment (made with enumitem's `\renewlist`) as long as it is done in the preamble. Though, technically, `\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" `enumerate` itself is to change `{`⟨*max-depth*⟩`}`. `\renewlist` *hard-codes* `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 `\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 zref-clever's perspective. Since the first four are defined by the kernel and already setup for zref-clever by default, we start from 5, and stop at the first non-existent `\c@enumN` counter.

```
\__zrefclever_compat_module:nn { enumitem }
  {
    \__zrefclever_if_package_loaded:nT { enumitem }
      {
        \int_set:Nn \l_tmpa_int { 5 }
        \bool_while_do:nn
          {
            \cs_if_exist_p:c
              { c@ enum \int_to_roman:n { \l_tmpa_int } }
          }
          {
            \__zrefclever_zcsetup:x
              {
                counterresetby =
                  {
                    enum \int_to_roman:n { \l_tmpa_int } =
                    enum \int_to_roman:n { \l_tmpa_int - 1 }
                  } ,
                countertype =
                  { enum \int_to_roman:n { \l_tmpa_int } = item } ,
              }
            \int_incr:N \l_tmpa_int
          }
        \int_compare:nNnT { \l_tmpa_int } > { 5 }
          { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
      }
  }
```

9.9 subcaption

```
\__zrefclever_compat_module:nn { subcaption }
  {
    \__zrefclever_if_package_loaded:nT { subcaption }
      {
        \__zrefclever_zcsetup:n
          {
            countertype =
              {
                subfigure = figure ,
                subtable = table ,
              } ,
            counterresetby =
              {
                subfigure = figure ,
                subtable = table ,
              } ,
          }
```

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 } }
      }
  }
```

9.10 subfig

Though subfig offers \subref (as subcaption), I could not find any reasonable place to add the subref property to zref's main list.

```
\__zrefclever_compat_module:nn { subfig }
  {
    \__zrefclever_if_package_loaded:nT { subfig }
      {
        \__zrefclever_zcsetup:n
          {
            countertype =
              {
                subfigure = figure ,
                subtable = table ,
              } ,
            counterresetby =
              {
                subfigure = figure ,
                subtable = table ,
              } ,
          }
      }
  }
⟨/package⟩
```

10 Language files

Initial values for the English, German, French, Portuguese, and Spanish language files have been provided by the author. Translations available for document elements' names in other packages have been an useful reference for the purpose, namely: babel, cleveref, translator, and translations.

10.1 Localization guidelines

Since the task of localizing zref-clever to work in different languages depends on the generous work of contributors, it is a good idea to set some guidelines not only to ease the task itself but also to document what the package expects in this regard.

The first general observation is that, contrary to a common initial reaction of those faced with the task of localizing the reference types, is that the job is not quite one of "translation". The reference type names are just the internal names used by the package to refer to them, technically, they could just as well be foobars. Of course, for practical reasons, they were chosen to be semantic. However, what we are searching for is not really the translation to the reference type name itself, but rather for the word / term / expression which is typically used to refer to the document object that the reference type is meant to represent. And terms that should work well in the contexts which cross-references are commonly used.

That said, some comments about the reference types and common pitfalls.

Sectioning: A number of reference types are provided to support referencing to document sectioning commands. Obviously, `part`, `chapter`, `section`, and `paragraph` are meant to refer to the sectioning commands of the standard classes and elsewhere, which anyone reading this is certainly acquainted with. Note that zref-clever uses – by default at least, which is what the language files cater for – the `section` reference type to refer to `\subsections` and `\subsubsections` as well, similarly, `paragraph` also refers to `\subparagraph`. The `appendix` reference type is meant to refer to any sectioning command – be them chapters, sections, or paragraphs – issued after `\appendix`, which corresponds to how the standard classes, the KOMA Script classes, and memoir deal with appendices. The `book` reference type deserves some explanation. The word "book" has a good number of meanings, and the most common one is not the one which is intended here. The Webster dictionary gives us a couple of definitions of interest: "1. A collection of sheets of paper, or similar material, blank, written, or printed, bound together; commonly, many folded and bound sheets containing continuous printing or writing." and "3. A part or subdivision of a treatise or literary work; as, the tenth book of 'Paradise Lost'." It is this third meaning which the `book` reference type is meant to support: a major subdivision of a work, much like `\part`. Even if it does not exist in the standard classes, it may exist elsewhere, in particular, it is provided by memoir.

Common numbered objects: Nothing surprising here, just being explicit. `table` and `figure` refer to the document's respective floats objects. `page` to the page number. `item` to the item number in `enumerate` environments. Similarly, `line` is meant to refer to line numbers.

Notes: zref-clever provides three reference types in this area: `footnote`, `endnote`, and `note`. The first two refer to footnotes and end notes, respectively. The third is meant as a convenience for a general "note" object, either the other two, or something else. By experience, here is one place where that initial observation of not simply translating the reference types names is particularly relevant. There's a natural temptation, because three different types exist and are somewhat close to each other, to distinguish them clearly. Duty would compel us to do so. But that may lead to less than ideal results. Different terms work well for some languages, like English and German, which have compound words for the purpose. But less so for other languages, like Portuguese, French, or Italian. For example, in a document in French which only contains footnotes, arguably a very common use case, would it be better to refer to a footnote as just "note", or be very precise with "note infrapaginale"? Of course, in a document which contains both footnotes and end notes, we may need the distinction. But is it really the better default?

True, possibly the inclusion of the `note` reference type, with no clear object to refer to, creates more noise than convenience here. If I recall correctly, my intention was to provide an easy way out for users from possible contentious localizations for `footnote` and `endnote`, but I'm not sure if it's been working like this in practice, and I should probably have refrained from adding it in the first place.

Math & Co.: A good number of reference types provided by the package are meant to cater for document objects commonly used in Mathematics and related areas. They are either straight math environments, defined by the kernel, amsmath or other packages, or environments which are normally not pre-defined by the kernel or the standard classes, but are traditionally defined by users with the kernel's `\newtheorem` or similar constructs available in the LaTeX package ecosystem. For most of them, localization should strive as much as possible to use the formal terms, jargon really, typically employed by mathematicians, logicians, and friends. Namely for the reference types: `equation`, `theorem`, `lemma`, `corollary`, `proposition`, `definition`, `proof`, `result`, and `remark`. Regarding `example`, `exercise`, and `solution` being somewhat less formal is admissible. But the chosen terms should still be fit for use in Math related contexts, and should be assumed were created by `\newtheorem` or similar, even if users may well find other uses for these types.

Code: A couple of reference types are provided for code related environments: `algorithm` and `listing`. By experience, the `listing` type has already proven to be a particularly challenging one. Formally, it should be a good default term to encompass anything which may regularly be included in a `lstlisting` environment as provided by the listings package. However, it seems that in different languages it is quite difficult to find a satisfying term for it. Though my English is decent, I'm not a native speaker, still I'm not even sure how common the term is used for the purpose even in English. It seems to be traditional enough in the LaTeX community at least. In doubt, pend to the jargon side, anglicism if need be. Since we are bound to displease mostly everyone anyway, at least we do so in a consistent manner.

Completeness and abbreviated forms: Ideally, the language file should be as complete as possible. "Complete" meaning it contains: i) the defaults for all basic separators, `namesep`, `pairsep`, `listsep`, `lastsep`, `tpairsep`, `tlistsep`, `tlastsep`, `notesep`, and `rangesep`; ii) the non-abbreviated forms of names for all the supported reference types, according to the language definitions, that is, usually for `Name-sg`, `name-sg`, `Name-pl`, `name-pl`, but only for the capitalized forms if the language was declared with `allcaps` option, and names for each declension case, if the language was declared with `declension`; iii) genders for each reference type, if the language was declared with `gender`. The language file may include some other things, like some type specific settings for separators or refbounds, and also some abbreviated name forms. In the case of abbreviated name forms, it is usual and desirable to provide some, but they should be used sparingly, only for cases where the abbreviation is a common and well established tradition for the language. The reason is that `abbrev=true` is quite a common use case, and it is easier to provide an occasional wanted abbreviated form, if the language file didn't include it, than it is to disable several unwanted ones, if the language file includes too many of them. What should be aimed at is to provide a good default abbreviations set. Unusual or disputable abbreviations should be avoided. In particular, there is no need at all to provide the same set of abbreviations for each language. It is not because English has them for a given type that some other language has to have them, and it is not because English lacks them for another type, that other languages shouldn't have them. Still, with regard to abbreviated forms, it is better to be conservative than opinionated.

babel names: As is known, babel defines a set of captions for different document objects for each supported language. In some cases, they intersect with the objects referred to with cross-references, in which case consistency with babel should be maintained as much as possible. This is specially the case for prominent and traditional objects, such as `\chaptername`, `\figurename`, `\tablename`, `\pagename`, `\partname`, and `\appendixname`. This is not set in stone, but there should be good reason to diverge from it. In particular, if a certain term is contentious in a given language, babel's default should be preferred. For example, "table" vs. "tableau" in French, or "cuadro" vs. "tabla" in Spanish.

Input encoding of language files: When zref-clever was released, the LaTeX kernel already used UTF-8 as default input encoding. Indeed, zref-clever requires a kernel even newer than the one where the default input encoding was changed. That given, UTF-8 input encoding was made a requirement of the package, and hence the language files should be in UTF-8, since it makes them easier to read and maintain than LICR.

Precedence rule for options in the language files: Any option given twice or more times has to have some precedence rule. Normally, the language files should not contain options in duplicity, but they may happen when setting some "group" `refbounds` options, in which case precedence rules become relevant. For user facing options (those set with `\zcLanguageSetup`), the option is always set, regardless of its previous state. Which means that the last value takes precedence. For the language files, we have to load them at `begindocument` (or later), since that's the point where we know from babel or polyglossia the `\languagename`. But we also don't want to override any options the user has actively set in the preamble. So the language files only set the values if they were not previously set. In other words, for them the precedence order is inverted, the first value takes precedence.

zref-vario: If you are interested in the localization of zref-clever to your language, and willing to contribute to it, you may also want to consider doing the same for the companion package zref-vario. It is actually a much simpler task than localizing zref-clever.

10.2 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   = {~and\nobreakspace} ,
listsep   = {,~} ,
lastsep   = {~and\nobreakspace} ,
tpairsep  = {~and\nobreakspace} ,
tlistsep  = {,~} ,
tlastsep  = {,~and\nobreakspace} ,
notesep   = {~} ,
rangesep  = {~to\nobreakspace} ,

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

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

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

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

type = paragraph ,
  Name-sg = Paragraph ,
  name-sg = paragraph ,
  Name-pl = Paragraphs ,
  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} ,
  rangetopair = false ,

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. ,
  refbounds-first-sg = {,(,),} ,
  refbounds = {(,,,)} ,

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 ,
  name-sg = proof ,
  Name-pl = Proofs ,
  name-pl = proofs ,

type = result ,
  Name-sg = Result ,
  name-sg = result ,
  Name-pl = Results ,
  name-pl = results ,

type = remark ,
  Name-sg = Remark ,
  name-sg = remark ,
  Name-pl = Remarks ,
  name-pl = remarks ,

type = example ,
  Name-sg = Example ,
  name-sg = example ,
  Name-pl = Examples ,
  name-pl = examples ,

type = algorithm ,
  Name-sg = Algorithm ,
  name-sg = algorithm ,
  Name-pl = Algorithms ,
  name-pl = algorithms ,

type = listing ,
  Name-sg = Listing ,
  name-sg = listing ,
  Name-pl = Listings ,
  name-pl = listings ,

type = exercise ,
  Name-sg = Exercise ,
  name-sg = exercise ,
  Name-pl = Exercises ,
  name-pl = exercises ,

type = solution ,
  Name-sg = Solution ,
  name-sg = solution ,
  Name-pl = Solutions ,
  name-pl = solutions ,
⟨/lang-english⟩
```

10.3 German

German language file has been initially provided by the author.

babel-german also has `.ldf`s for `germanb` and `ngermanb`, but they are deprecated as options and, if used, they fall back respectively to `german` and `ngerman`.

```
⟨*package⟩
\zcDeclareLanguage
  [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
  { german }
\zcDeclareLanguageAlias { ngerman      } { german }
\zcDeclareLanguageAlias { austrian     } { german }
\zcDeclareLanguageAlias { naustrian    } { german }
\zcDeclareLanguageAlias { swissgerman  } { german }
\zcDeclareLanguageAlias { nswissgerman } { german }
⟨/package⟩
⟨*lang-german⟩
namesep  = {\nobreakspace} ,
pairsep  = {~und\nobreakspace} ,
listsep  = {,~} ,
lastsep  = {~und\nobreakspace} ,
tpairsep = {~und\nobreakspace} ,
tlistsep = {,~} ,
tlastsep = {~und\nobreakspace} ,
notesep  = {~} ,
rangesep = {~bis\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 = Büchern ,
  case = G ,
    Name-sg = Buches ,
    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 = Teiles ,
    Name-pl = Teile ,

type = chapter ,
  gender = n ,
  case = N ,
    Name-sg = Kapitel ,
    Name-pl = Kapitel ,
  case = A ,
    Name-sg = Kapitel ,
    Name-pl = Kapitel ,
  case = D ,
    Name-sg = Kapitel ,
    Name-pl = Kapiteln ,
  case = G ,
    Name-sg = Kapitels ,
    Name-pl = Kapitel ,

type = section ,
  gender = m ,
  case = N ,
    Name-sg = Abschnitt ,
    Name-pl = Abschnitte ,
  case = A ,
    Name-sg = Abschnitt ,
    Name-pl = Abschnitte ,
  case = D ,
    Name-sg = Abschnitt ,
    Name-pl = Abschnitten ,
  case = G ,
    Name-sg = Abschnitts ,
    Name-pl = Abschnitte ,

type = paragraph ,
  gender = m ,
  case = N ,
    Name-sg = Absatz ,
    Name-pl = Absätze ,
  case = A ,
    Name-sg = Absatz ,
    Name-pl = Absätze ,
  case = D ,
    Name-sg = Absatz ,
    Name-pl = Absätzen ,
  case = G ,
    Name-sg = Absatzes ,
    Name-pl = Absätze ,

type = appendix ,
  gender = m ,
  case = N ,
    Name-sg = Anhang ,
    Name-pl = Anhänge ,
  case = A ,
    Name-sg = Anhang ,
    Name-pl = Anhänge ,
  case = D ,
    Name-sg = Anhang ,
    Name-pl = Anhängen ,
  case = G ,
    Name-sg = Anhangs ,
    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 = {\textendash} ,
  rangetopair = false ,

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. ,
    Name-pl-ab = Abb. ,

type = table ,
  gender = f ,
  case = N ,
    Name-sg = Tabelle ,
    Name-pl = Tabellen ,
  case = A ,
    Name-sg = Tabelle ,
    Name-pl = Tabellen ,
  case = D ,
    Name-sg = Tabelle ,
    Name-pl = Tabellen ,
  case = G ,
    Name-sg = Tabelle ,
    Name-pl = Tabellen ,

type = item ,
  gender = m ,
  case = N ,
    Name-sg = Punkt ,
    Name-pl = Punkte ,
  case = A ,
    Name-sg = Punkt ,
    Name-pl = Punkte ,
  case = D ,
    Name-sg = Punkt ,
    Name-pl = Punkten ,
  case = G ,
    Name-sg = Punktes ,
    Name-pl = Punkte ,

type = footnote ,
  gender = f ,
  case = N ,
    Name-sg = Fußnote ,
    Name-pl = Fußnoten ,
  case = A ,
    Name-sg = Fußnote ,
    Name-pl = Fußnoten ,
  case = D ,
    Name-sg = Fußnote ,
    Name-pl = Fußnoten ,
  case = G ,
    Name-sg = Fußnote ,
    Name-pl = Fußnoten ,

type = endnote ,
  gender = f ,
  case = N ,
    Name-sg = Endnote ,
    Name-pl = Endnoten ,
  case = A ,
    Name-sg = Endnote ,
    Name-pl = Endnoten ,
  case = D ,
    Name-sg = Endnote ,
    Name-pl = Endnoten ,
  case = G ,
    Name-sg = Endnote ,
    Name-pl = Endnoten ,

type = note ,
  gender = f ,
  case = N ,
    Name-sg = Anmerkung ,
    Name-pl = Anmerkungen ,
  case = A ,
    Name-sg = Anmerkung ,
    Name-pl = Anmerkungen ,
  case = D ,
    Name-sg = Anmerkung ,
    Name-pl = Anmerkungen ,
  case = G ,
    Name-sg = Anmerkung ,
    Name-pl = Anmerkungen ,

type = equation ,
  gender = f ,
  case = N ,
    Name-sg = Gleichung ,
    Name-pl = Gleichungen ,
  case = A ,
    Name-sg = Gleichung ,
    Name-pl = Gleichungen ,
  case = D ,
    Name-sg = Gleichung ,
    Name-pl = Gleichungen ,
  case = G ,
    Name-sg = Gleichung ,
    Name-pl = Gleichungen ,
  refbounds-first-sg = {,(,),} ,
  refbounds = {(,,,)} ,

type = theorem ,
  gender = n ,
  case = N ,
    Name-sg = Theorem ,
    Name-pl = Theoreme ,
  case = A ,
    Name-sg = Theorem ,
    Name-pl = Theoreme ,
  case = D ,
    Name-sg = Theorem ,
    Name-pl = Theoremen ,
  case = G ,
    Name-sg = Theorems ,
    Name-pl = Theoreme ,

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 ,
  case = G ,
    Name-sg = Übungsaufgabe ,
    Name-pl = Übungsaufgaben ,

type = solution ,
  gender = f ,
  case = N ,
    Name-sg = Lösung ,
    Name-pl = Lösungen ,
  case = A ,
    Name-sg = Lösung ,
    Name-pl = Lösungen ,
  case = D ,
    Name-sg = Lösung ,
    Name-pl = Lösungen ,
  case = G ,
    Name-sg = Lösung ,
    Name-pl = Lösungen ,
```
⟨/lang-german⟩

10.4 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) and the fr.comp.text.tex (at https://groups.google.com/g/fr.comp.text.tex/c/Fa11Tf6MFFs) mailing lists.

babel-french also has .ldfs for francais, frenchb, and canadien, but they are deprecated as options and, if used, they fall back to either french or acadian.

```
⟨*package⟩
\zcDeclareLanguage [ gender = { f , m } ] { french }
\zcDeclareLanguageAlias { acadian  } { french }
⟨/package⟩
```

```
⟨*lang-french⟩
```

```
namesep   = {\nobreakspace} ,
pairsep   = {~et\nobreakspace} ,
listsep   = {,~} ,
lastsep   = {~et\nobreakspace} ,
tpairsep = {~et\nobreakspace} ,
tlistsep = {,~} ,
tlastsep = {~et\nobreakspace} ,
notesep   = {~} ,
rangesep = {~à\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 = {-} ,
  rangetopair = false ,

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 ,
  Name-pl = Figures ,
  name-pl = figures ,

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

type = item ,
  gender = m ,
  Name-sg = Point ,
  name-sg = point ,
  Name-pl = Points ,
  name-pl = points ,

type = footnote ,
  gender = f ,
  Name-sg = Note ,
  name-sg = note ,
  Name-pl = Notes ,
  name-pl = notes ,

type = endnote ,
  gender = f ,
  Name-sg = Note ,
  name-sg = note ,
  Name-pl = Notes ,
  name-pl = notes ,

type = note ,
  gender = f ,
  Name-sg = Note ,
  name-sg = note ,
  Name-pl = Notes ,
  name-pl = notes ,

type = equation ,
  gender = f ,
  Name-sg = Équation ,
  name-sg = équation ,
  Name-pl = Équations ,
  name-pl = équations ,
  refbounds-first-sg = {,(,),} ,
  refbounds = {(,,,)} ,

type = theorem ,
  gender = m ,
  Name-sg = Théorème ,
  name-sg = théorème ,
  Name-pl = Théorèmes ,
  name-pl = théorèmes ,

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 ,
  name-pl = exemples ,

type = algorithm ,
  gender = m ,
  Name-sg = Algorithme ,
  name-sg = algorithme ,
  Name-pl = Algorithmes ,
  name-pl = algorithmes ,

type = listing ,
  gender = m ,
  Name-sg = Listing ,
  name-sg = listing ,
  Name-pl = Listings ,
  name-pl = listings ,

type = exercise ,
  gender = m ,
  Name-sg = Exercice ,
  name-sg = exercice ,
  Name-pl = Exercices ,
  name-pl = exercices ,

type = solution ,
  gender = f ,
  Name-sg = Solution ,
  name-sg = solution ,
  Name-pl = Solutions ,
  name-pl = solutions ,
```

⟨/lang-french⟩

10.5 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.

⟨*package⟩

```
\zcDeclareLanguage [ gender = { f , m } ] { portuguese }
\zcDeclareLanguageAlias { brazilian } { portuguese }
\zcDeclareLanguageAlias { brazil    } { portuguese }
\zcDeclareLanguageAlias { portuges  } { portuguese }
```

⟨/package⟩

⟨*lang-portuguese⟩

```
namesep  = {\nobreakspace} ,
pairsep  = {~e\nobreakspace} ,
listsep  = {,~} ,
lastsep  = {~e\nobreakspace} ,
tpairsep = {~e\nobreakspace} ,
tlistsep = {,~} ,
tlastsep = {~e\nobreakspace} ,
notesep  = {~} ,
rangesep = {~a\nobreakspace} ,

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} ,
  rangetopair = false ,

type = line ,
  gender = f ,
  Name-sg = Linha ,
  name-sg = linha ,
  Name-pl = Linhas ,
  name-pl = linhas ,

type = figure ,
  gender = f ,
  Name-sg = Figura ,
  name-sg = figura ,
  Name-pl = Figuras ,
  name-pl = figuras ,
  Name-sg-ab = Fig. ,
  name-sg-ab = fig. ,
  Name-pl-ab = Figs. ,
  name-pl-ab = figs. ,

type = table ,
  gender = f ,
  Name-sg = Tabela ,
  name-sg = tabela ,
  Name-pl = Tabelas ,
  name-pl = tabelas ,

type = item ,
  gender = m ,
  Name-sg = Item ,
  name-sg = item ,
  Name-pl = Itens ,
  name-pl = itens ,

type = footnote ,
  gender = f ,
  Name-sg = Nota ,
  name-sg = nota ,
  Name-pl = Notas ,
  name-pl = notas ,

type = endnote ,
  gender = f ,
  Name-sg = Nota ,
  name-sg = nota ,
  Name-pl = Notas ,
  name-pl = notas ,

type = note ,
  gender = f ,
  Name-sg = Nota ,
  name-sg = nota ,
  Name-pl = Notas ,
  name-pl = notas ,

type = equation ,
  gender = f ,
  Name-sg = Equação ,
  name-sg = equação ,
  Name-pl = Equações ,
  name-pl = equações ,
  Name-sg-ab = Eq. ,
  name-sg-ab = eq. ,
  Name-pl-ab = Eqs. ,
  name-pl-ab = eqs. ,
  refbounds-first-sg = {,(,),} ,
  refbounds = {(,,,)} ,

type = theorem ,
  gender = m ,
  Name-sg = Teorema ,
  name-sg = teorema ,
  Name-pl = Teoremas ,
  name-pl = teoremas ,

type = lemma ,
  gender = m ,
  Name-sg = Lema ,
  name-sg = lema ,
  Name-pl = Lemas ,
  name-pl = lemas ,

type = corollary ,
  gender = m ,
  Name-sg = Corolário ,
  name-sg = corolário ,
  Name-pl = Corolários ,
  name-pl = corolários ,

type = proposition ,
  gender = f ,
  Name-sg = Proposição ,
  name-sg = proposição ,
  Name-pl = Proposições ,
  name-pl = proposições ,

type = definition ,
  gender = f ,
  Name-sg = Definição ,
  name-sg = definição ,
  Name-pl = Definições ,
  name-pl = definições ,

type = proof ,
  gender = f ,
  Name-sg = Demonstração ,
  name-sg = demonstração ,
  Name-pl = Demonstrações ,
  name-pl = demonstrações ,

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⟩
```

10.6 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 = m ,
  Name-sg = Apéndice ,
  name-sg = apéndice ,
  Name-pl = Apéndices ,
  name-pl = apéndices ,

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

type = line ,
  gender = f ,
  Name-sg = Línea ,
  name-sg = línea ,
  Name-pl = Líneas ,
  name-pl = líneas ,

type = figure ,
  gender = f ,
  Name-sg = Figura ,
  name-sg = figura ,
  Name-pl = Figuras ,
  name-pl = figuras ,

type = table ,
  gender = m ,
  Name-sg = Cuadro ,
  name-sg = cuadro ,
  Name-pl = Cuadros ,
  name-pl = cuadros ,

type = item ,
  gender = m ,
  Name-sg = Punto ,
  name-sg = punto ,
  Name-pl = Puntos ,
  name-pl = puntos ,

type = footnote ,
  gender = f ,
  Name-sg = Nota ,
  name-sg = nota ,
  Name-pl = Notas ,
  name-pl = notas ,

type = endnote ,
  gender = f ,
  Name-sg = Nota ,
  name-sg = nota ,
  Name-pl = Notas ,
  name-pl = notas ,

type = note ,
  gender = f ,
  Name-sg = Nota ,
  name-sg = nota ,
  Name-pl = Notas ,
  name-pl = notas ,

type = equation ,
  gender = f ,
  Name-sg = Ecuación ,
  name-sg = ecuación ,
  Name-pl = Ecuaciones ,
  name-pl = ecuaciones ,
  refbounds-first-sg = {,(,),} ,
  refbounds = {(,,,)} ,

type = theorem ,
  gender = m ,
  Name-sg = Teorema ,
  name-sg = teorema ,
  Name-pl = Teoremas ,
  name-pl = teoremas ,

type = lemma ,
  gender = m ,
  Name-sg = Lema ,
  name-sg = lema ,
  Name-pl = Lemas ,
  name-pl = lemas ,

type = corollary ,
  gender = m ,
  Name-sg = Corolario ,
  name-sg = corolario ,
  Name-pl = Corolarios ,
  name-pl = corolarios ,

type = proposition ,
  gender = f ,
  Name-sg = Proposición ,
  name-sg = proposición ,
  Name-pl = Proposiciones ,
  name-pl = proposiciones ,

type = definition ,
  gender = f ,
  Name-sg = Definición ,
  name-sg = definición ,
  Name-pl = Definiciones ,
  name-pl = definiciones ,

type = proof ,
  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 ,
⟨/lang-spanish⟩
```

10.7 Dutch

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

```
⟨*package⟩
\zcDeclareLanguage [ gender = { f , m , n } ] { dutch }
⟨/package⟩
⟨*lang-dutch⟩
namesep   = {\nobreakspace} ,
pairsep   = {~en\nobreakspace} ,
listsep   = {,~} ,
lastsep   = {~en\nobreakspace} ,
tpairsep  = {~en\nobreakspace} ,
tlistsep  = {,~} ,
tlastsep  = {,~en\nobreakspace} ,
notesep   = {~} ,
rangesep  = {~t/m\nobreakspace} ,

type = book ,
  gender = n ,
  Name-sg = Boek ,
  name-sg = boek ,
  Name-pl = Boeken ,
  name-pl = boeken ,

type = part ,
  gender = n ,
  Name-sg = Deel ,
  name-sg = deel ,
  Name-pl = Delen ,
  name-pl = delen ,

type = chapter ,
  gender = n ,
  Name-sg = Hoofdstuk ,
  name-sg = hoofdstuk ,
  Name-pl = Hoofdstukken ,
  name-pl = hoofdstukken ,

type = section ,
  gender = m ,
  Name-sg = Paragraaf ,
  name-sg = paragraaf ,
  Name-pl = Paragrafen ,
  name-pl = paragrafen ,

type = paragraph ,
  gender = f ,
  Name-sg = Alinea ,
  name-sg = alinea ,
  Name-pl = Alinea's ,
  name-pl = alinea's ,

```

2022-12-27, ‘niluxv’: “bijlage” is chosen over “appendix” (plural “appendices”, gender: m, n) for consistency with babel/polyglossia. “bijlages” is also a valid plural; “bijlagen” is chosen for consistency with babel/polyglossia.

```
type = appendix ,
  gender = { f, m } ,
  Name-sg = Blage ,
  name-sg = blage ,
  Name-pl = Blagen ,
  name-pl = blagen ,

type = page ,
  gender = { f , m } ,
  Name-sg = Pagina ,
  name-sg = pagina ,
  Name-pl = Pagina's ,
  name-pl = pagina's ,
  rangesep = {\textendash} ,
  rangetopair = false ,

type = line ,
  gender = m ,
  Name-sg = Regel ,
  name-sg = regel ,
  Name-pl = Regels ,
  name-pl = regels ,

type = figure ,
  gender = { n , f , m } ,
  Name-sg = Figuur ,
  name-sg = figuur ,
  Name-pl = Figuren ,
  name-pl = figuren ,

type = table ,
  gender = { f , m } ,
  Name-sg = Tabel ,
  name-sg = tabel ,
  Name-pl = Tabellen ,
  name-pl = tabellen ,

type = item ,
  gender = n ,
  Name-sg = Punt ,
  name-sg = punt ,
  Name-pl = Punten ,
  name-pl = punten ,

type = footnote ,
  gender = { f , m } ,
  Name-sg = Voetnoot ,
  name-sg = voetnoot ,
  Name-pl = Voetnoten ,
  name-pl = voetnoten ,

type = endnote ,
  gender = { f , m } ,
  Name-sg = Eindnoot ,
  name-sg = eindnoot ,
  Name-pl = Eindnoten ,
  name-pl = eindnoten ,

type = note ,
  gender = f ,
  Name-sg = Opmerking ,
  name-sg = opmerking ,
  Name-pl = Opmerkingen ,
  name-pl = opmerkingen ,

type = equation ,
  gender = f ,
  Name-sg = Vergelking ,
  name-sg = vergelking ,
  Name-pl = Vergelkingen ,
  name-pl = vergelkingen ,
  Name-sg-ab = Vgl. ,
  name-sg-ab = vgl. ,
  Name-pl-ab = Vgl.'s ,
  name-pl-ab = vgl.'s ,
  refbounds-first-sg = {,(,),} ,
  refbounds = {(,,,)} ,

type = theorem ,
  gender = f ,
  Name-sg = Stelling ,
  name-sg = stelling ,
  Name-pl = Stellingen ,
  name-pl = stellingen ,

```

2022-01-09, '`niluxv`': 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".

```
type = lemma ,
  gender = n ,
  Name-sg = Lemma ,
  name-sg = lemma ,
  Name-pl = Lemma's ,
  name-pl = lemma's ,

type = corollary ,
  gender = n ,
  Name-sg = Gevolg ,
  name-sg = gevolg ,
  Name-pl = Gevolgen ,
  name-pl = gevolgen ,

type = proposition ,
  gender = f ,
  Name-sg = Propositie ,
  name-sg = propositie ,
  Name-pl = Proposities ,
  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 = Bews ,
  name-sg = bews ,
  Name-pl = Bewzen ,
  name-pl = bewzen ,

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 ,

```

2022-12-27, ‘niluxv’: “algoritmes” is also a valid plural. “algoritmen” is chosen to be consistent with using “bijlagen” (and not “bijlages”) as the plural of “bijlage”.

```
type = algorithm ,
  gender = { n , f , m } ,
  Name-sg = Algoritme ,
  name-sg = algoritme ,
  Name-pl = Algoritmen ,
  name-pl = algoritmen ,

```

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 ,
  gender = { f , m } ,
  Name-sg = Opgave ,
  name-sg = opgave ,
  Name-pl = Opgaven ,
  name-pl = opgaven ,

type = solution ,
  gender = f ,
  Name-sg = Oplossing ,
  name-sg = oplossing ,
  Name-pl = Oplossingen ,
  name-pl = oplossingen ,
⟨/lang-dutch⟩
```

10.8 Italian

Italian language file initially contributed by Matteo Ferrigato (issue #11), with the help of participants of the Gruppo Utilizzatori Italiani di TEX (GuIT) forum (at https://www.guitex.org/home/it/forum/5-tex-e-latex/121856-closed-zref-clever-e-localizzazione-in-

```
⟨*package⟩
\zcDeclareLanguage [ gender = { f , m } ] { italian }
⟨/package⟩
⟨*lang-italian⟩
namesep   = {\nobreakspace} ,
pairsep   = {~e\nobreakspace} ,
listsep   = {,~} ,
lastsep   = {~e\nobreakspace} ,
tpairsep  = {~e\nobreakspace} ,
tlistsep  = {,~} ,
tlastsep  = {,~e\nobreakspace} ,
notesep   = {~} ,
rangesep  = {~a\nobreakspace} ,
+refbounds-rb = {da\nobreakspace,,,} ,

type = book ,
  gender = m ,
  Name-sg = Libro ,
  name-sg = libro ,
  Name-pl = Libri ,
  name-pl = libri ,

type = part ,
  gender = f ,
  Name-sg = Parte ,
  name-sg = parte ,
  Name-pl = Parti ,
  name-pl = parti ,

type = chapter ,
  gender = m ,
  Name-sg = Capitolo ,
  name-sg = capitolo ,
  Name-pl = Capitoli ,
  name-pl = capitoli ,

type = section ,
  gender = m ,
  Name-sg = Paragrafo ,
  name-sg = paragrafo ,
  Name-pl = Paragrafi ,
  name-pl = paragrafi ,

type = paragraph ,
  gender = m ,
  Name-sg = Capoverso ,
  name-sg = capoverso ,
  Name-pl = Capoversi ,
  name-pl = capoversi ,

type = appendix ,
  gender = f ,
  Name-sg = Appendice ,
  name-sg = appendice ,
  Name-pl = Appendici ,
  name-pl = appendici ,

type = page ,
  gender = f ,
  Name-sg = Pagina ,
  name-sg = pagina ,
  Name-pl = Pagine ,
  name-pl = pagine ,
  Name-sg-ab = Pag. ,
  name-sg-ab = pag. ,
  Name-pl-ab = Pag. ,
  name-pl-ab = pag. ,
  rangesep = {\textendash} ,
  rangetopair = false ,
  +refbounds-rb = {,,,} ,

type = line ,
  gender = f ,
  Name-sg = Riga ,
  name-sg = riga ,
  Name-pl = Righe ,
  name-pl = righe ,

type = figure ,
  gender = f ,
  Name-sg = Figura ,
  name-sg = figura ,
  Name-pl = Figure ,
  name-pl = figure ,
  Name-sg-ab = Fig. ,
  name-sg-ab = fig. ,
  Name-pl-ab = Fig. ,
  name-pl-ab = fig. ,

type = table ,
  gender = f ,
  Name-sg = Tabella ,
  name-sg = tabella ,
  Name-pl = Tabelle ,
  name-pl = tabelle ,
  Name-sg-ab = Tab. ,
  name-sg-ab = tab. ,
  Name-pl-ab = Tab. ,
  name-pl-ab = tab. ,

type = item ,
  gender = m ,
  Name-sg = Punto ,
  name-sg = punto ,
  Name-pl = Punti ,
  name-pl = punti ,

type = footnote ,
  gender = f ,
  Name-sg = Nota ,
  name-sg = nota ,
  Name-pl = Note ,
  name-pl = note ,

type = endnote ,
  gender = f ,
  Name-sg = Nota ,
  name-sg = nota ,
  Name-pl = Note ,
  name-pl = note ,

type = note ,
  gender = f ,
  Name-sg = Nota ,
  name-sg = nota ,
  Name-pl = Note ,
  name-pl = note ,

type = equation ,
  gender = f ,
  Name-sg = Equazione ,
  name-sg = equazione ,
  Name-pl = Equazioni ,
  name-pl = equazioni ,
  Name-sg-ab = Eq. ,
  name-sg-ab = eq. ,
  Name-pl-ab = Eq. ,
  name-pl-ab = eq. ,
  +refbounds-rb = {da\nobreakspace(,,,)} ,
  refbounds-first-sg = {,(,),} ,
  refbounds = {(,,,)} ,

type = theorem ,
  gender = m ,
  Name-sg = Teorema ,
  name-sg = teorema ,
  Name-pl = Teoremi ,
  name-pl = teoremi ,

type = lemma ,
  gender = m ,
  Name-sg = Lemma ,
  name-sg = lemma ,
  Name-pl = Lemmi ,
  name-pl = lemmi ,

type = corollary ,
  gender = m ,
  Name-sg = Corollario ,
  name-sg = corollario ,
  Name-pl = Corollari ,
  name-pl = corollari ,

type = proposition ,
  gender = f ,
  Name-sg = Proposizione ,
  name-sg = proposizione ,
  Name-pl = Proposizioni ,
  name-pl = proposizioni ,

type = definition ,
  gender = f ,
  Name-sg = Definizione ,
  name-sg = definizione ,
  Name-pl = Definizioni ,
  name-pl = definizioni ,

type = proof ,
  gender = f ,
  Name-sg = Dimostrazione ,
  name-sg = dimostrazione ,
  Name-pl = Dimostrazioni ,
  name-pl = dimostrazioni ,

type = result ,
  gender = m ,
  Name-sg = Risultato ,
  name-sg = risultato ,
  Name-pl = Risultati ,
  name-pl = risultati ,

type = remark ,
  gender = f ,
  Name-sg = Osservazione ,
  name-sg = osservazione ,
  Name-pl = Osservazioni ,
  name-pl = osservazioni ,

type = example ,
  gender = m ,
  Name-sg = Esempio ,
  name-sg = esempio ,
  Name-pl = Esempi ,
  name-pl = esempi ,

type = algorithm ,
  gender = m ,
  Name-sg = Algoritmo ,
  name-sg = algoritmo ,
  Name-pl = Algoritmi ,
  name-pl = algoritmi ,

type = listing ,
  gender = m ,
  Name-sg = Listato ,
  name-sg = listato ,
  Name-pl = Listati ,
  name-pl = listati ,

type = exercise ,
  gender = m ,
  Name-sg = Esercizio ,
  name-sg = esercizio ,
  Name-pl = Esercizi ,
  name-pl = esercizi ,

type = solution ,
  gender = f ,
  Name-sg = Soluzione ,
  name-sg = soluzione ,
  Name-pl = Soluzioni ,
  name-pl = soluzioni ,
⟨/lang-italian⟩
```

Index

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.