The zref-clever package implementation*

Gustavo Barros†

2021-09-13

Contents

*This file describes v0.1.0-alpha, last revised 2021-09-13.
†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 also presume expl3 (which made to the kernel in the 2020-02-02 release). Hence, since we would not be able to go much backwards without special handling anyway, we make the cut with the inclusion of the new hook management system (ltcmdhooks), which is bound to be useful for our purposes, and was released with the 2021-06-01 kernel.

```
\providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
\IfFormatAtLeastTF{2021-06-01}
  {}
  {%
    \PackageError{zref-clever}{LaTeX kernel too old}
      {%
        'zref-clever' requires a LaTeX kernel newer than 2021-06-01.%
        \MessageBreak Loading will abort!%
      }%
    \endinput
  }%
```

Identify the package.

```
\ProvidesExplPackage {zref-clever} {2021-09-13} {0.1.0-alpha}
  {Do-what-I-mean cross-references based on zref}
```

2 Dependencies

```
\RequirePackage { zref-base }
\RequirePackage { zref-user }
\RequirePackage { translations }
```

3 zref setup

We are (mainly) interested in three basic label elements: the reference itself, the page, and the counter. The 'page' and 'counter' are respectively handled by modules zref-base and zref-counter. The zref-abspage also provides the 'abspage' property which gives us a safe and easy way to sort labels on page references. But the reference itself, stored by zref in the 'default' field, is somewhat a disputed real estate. In particular, the use of `\labelformat` 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 `zc@thecnt` for reserved use. We also store the "type" of the label at this point (see Section 4.1).

However, the zref-abspage module is very simple, but loads atbegshi, which is no longer needed with a recent kernel, which we require here anyway. So we can spare this additional dependency by providing the property internally. Since the job of zref-counter is also trivial, we do that too, and thus ensure that almost all needed data is stored

in "internal" properties. The only exception is the `page` property, which is available by default (no extra module required), and is not tampered with by `\labelformat`, as the `default` property is. Another exception which we don't need to handle at the data provision side, but need to cater for in the retrieval side, are the `url` / `urluse` properties from the zref-xr module, which are added to the labels imported from external documents, and needed to construct hyperlinks to them.

```
\zref@newprop { zc@counter } { \@currentcounter }
\zref@addprop \ZREF@mainlist { zc@counter }
\zref@newprop { zc@type }
  { \exp_args:NNe \prop_item:Nn \l__zrefclever_counter_type_prop { \@currentcounter } }
\zref@addprop \ZREF@mainlist { zc@type }
```

Provide `zc@thecnt` property, based on the definition of `\@currentlabel` done inside `\refstepcounter` in 'texdoc source2e', section 'ltcounts.dtx'. We just drop the `\p@...` prefix.

```
\zref@newprop { zc@thecnt } { \use:c { the \@currentcounter } }
\zref@addprop \ZREF@mainlist { zc@thecnt }
```

At this point, the basic properties of interest are handled. However, the moment where the label is set is a privileged one, because at this point we have a lot of raw information available. Information which may be difficult to retrieve later on by parsing the reference printed value of the counter, which we stored in `zc@thecnt` above. Hence, we seize the opportunity to store some of that information in a way which eases significantly the task of processing the reference later on: i) the counter *value*, as a number; ii) the counter (and value) of the set of counters which may trigger a reset of the current counter.

The first one is trivial, `\c@`⟨*counter*⟩ contains the counter's numerical value (see 'texdoc source2e', section 'ltcounts.dtx'), we just store it in `zc@cntval`.

```
\zref@newprop { zc@cntval } [0] { \int_use:c { c@ \@currentcounter } }
\zref@addprop \ZREF@mainlist { zc@cntval }
```

And we need the numeric value for the page and abspage.

```
\zref@newprop* { zc@pgval } [0] { \int_use:c { c@page } }
\zref@addprop \ZREF@mainlist { zc@pgval }
\int_new:N \g__zrefclever_abspage_int
\AddToHook { shipout/before } { \int_gincr:N \g__zrefclever_abspage_int }
\zref@newprop* { zc@abspg } [0] { \int_use:N \g__zrefclever_abspage_int }
\zref@addprop \ZREF@mainlist { zc@abspg }
```

The second one is trickier. 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}`, again see section 'ltcounts.dtx' in 'source2e'). Besides, there may be a chain of resetting counters, which must be taken into account. The procedure below examines a set of counters, those included in `\l__zrefclever_counter_resetters_seq`, and for each counter retrieves its "enclosing counters" recursively. 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.

__zrefclever_get_enclosing_counters:n
__zrefclever_get_enclosing_counters_value:n

Recursively generate a *sequence* of "enclosing counters" and values, for a given {⟨*counter*⟩} and leave it in the input stream. These functions must be expandable, since they get called from `\zref@newprop` and are the ones 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.

```
\cs_new:Npn \__zrefclever_get_enclosing_counters:n #1
  {
    \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
      {
        { \__zrefclever_counter_reset_by:n {#1} }
        \__zrefclever_get_enclosing_counters:e
          { \__zrefclever_counter_reset_by:n {#1} }
      }
  }
\cs_new:Npn \__zrefclever_get_enclosing_counters_value:n #1
  {
    \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
      {
        { \int_use:N \cs:w c@ \__zrefclever_counter_reset_by:n {#1} \cs_end: }
        \__zrefclever_get_enclosing_counters_value:e
          { \__zrefclever_counter_reset_by:n {#1} }
      }
  }
```

Both `e` and `f` expansions work for this particular recursive call. For the time being, I'll stay with the `e` variant, since conceptually it is what I want (`x` itself is not expandable), and this package is unlikely to be used within the context of older kernels for which the performance penalty of the `e` expansion would ensue (see also https://tex.stackexchange.com/q/611370/#comment1529282_611385, thanks Enrico Gregorio, aka 'egreg').

```
\cs_generate_variant:Nn \__zrefclever_get_enclosing_counters:n { V , e }
\cs_generate_variant:Nn \__zrefclever_get_enclosing_counters_value:n { V , e }
```

(*End definition for* `\__zrefclever_get_enclosing_counters:n` *and* `\__zrefclever_get_enclosing_counters_value:n`.)

__zrefclever_counter_reset_by:n
__zrefclever_counter_reset_by_aux:nn
__zrefclever_counter_reset_by_auxi:nnn

Auxiliary functions for `\__zrefclever_get_enclosing_counters:n` and `\__zrefclever_get_enclosing_counters_value:n`. They are broken in parts to be able to use the expandable mapping functions. In particular `\__zrefclever_counter_reset_by:n` leaves in the stream the "enclosing counter" which resets {⟨*counter*⟩}.

```
\cs_new:Npn \__zrefclever_counter_reset_by:n #1
  {
    \bool_if:nTF
      { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} }
      { \prop_item:Nn  \l__zrefclever_counter_resetby_prop {#1} }
      {
        \seq_map_tokens:Nn \l__zrefclever_counter_resetters_seq
          { \__zrefclever_counter_reset_by_aux:nn {#1} }
      }
  }
\cs_new:Npn \__zrefclever_counter_reset_by_aux:nn #1#2
```

```
  {
    \cs_if_exist:cT { c@ #2 }
      {
        \tl_if_empty:cF { cl@ #2 }
          {
            \tl_map_tokens:cn { cl@ #2 }
              { \__zrefclever_counter_reset_by_auxi:nnn {#2} {#1} }
          }
      }
  }
\cs_new:Npn \__zrefclever_counter_reset_by_auxi:nnn #1#2#3
  {
    \str_if_eq:nnT {#2} {#3}
      { \tl_map_break:n { \seq_map_break:n {#1} } }
  }
```

(*End definition for* `\__zrefclever_counter_reset_by:n`, `\__zrefclever_counter_reset_by_aux:nn`, *and* `\__zrefclever_counter_reset_by_auxi:nnn`.)

Finally, add `zc@enclcnt` and `zc@enclval` to zref's `main` property list.

```
\zref@newprop { zc@enclcnt }
  { \__zrefclever_get_enclosing_counters:V \@currentcounter }
\zref@newprop { zc@enclval }
  { \__zrefclever_get_enclosing_counters_value:V \@currentcounter }
\zref@addprop \ZREF@mainlist { zc@enclcnt }
\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, the "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 can be used for that. But we can decide whether two labels can be compressed or not based on this format: if they are identical, we can compress them, otherwise, we can't. cleveref actually resets the counter to "1" with `\setcounter`, which is a global operation, and restores it in sequence. Here we adopt a more cautious approach of locally redefining `\c@page` to return "1", thus avoiding any global spillovers of this trick. Since this operation is not expandable we cannot run it directly from the property definition. Hence, we use a shipout hook, and set `\g__zrefclever_page_format_tl`, which can then be retrieved by the starred definition of `\zref@newprop*{zc@pgfmt}`.

__zrefclever_page_numbering:

```
\tl_new:N \g__zrefclever_page_format_tl
\cs_new_protected:Npx \__zrefclever_page_format_aux: { \int_eval:n { 1 } }
\AddToHook { shipout / before }
  {
    \group_begin:
```

```
    \cs_set_eq:NN \c@page \__zrefclever_page_format_aux:
    \exp_args:NNx \tl_gset:Nn \g__zrefclever_page_format_tl { \thepage }
    \group_end:
  }
\zref@newprop* { zc@pgfmt } { \g__zrefclever_page_format_tl }
\zref@addprop \ZREF@mainlist { zc@pgfmt }
```

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

4 Plumbing

4.1 Reference types

Let's start with a bit of terminology, to avoid confusion. A "reference type" is the basic zref-clever setup unit for specifying how a cross-reference group of a certain kind is to be typeset. Though, usually, it will have the same name as the underlying LaTeX *counter*, they are conceptually different. zref-clever defines *reference types* and an association between each *counter* and its *type*, it does not define the counters themselves, which are defined by your document. One *reference type* can be associated with one or more *counters*, but each counter can only have one *type* (for a given label...), and that determines how the reference is typeset. References to different *counters* of the same *type* are grouped together, and treated alike by zref-clever. A *reference type* may exist even when the *counter* it is associated with is not actually defined, and this inconsequential. In practice, the contrary may also happen, a *counter* may be defined but we have no *type* for it, but this must be handled by zref-clever as a "missing type" error (at least, if we try to refer to it).

A *reference type* can be associated with multiple counters because we may want to refer to different document elements, with different *counters*, with a single name, as a single *type*. One prominent case of this are sectioning commands. `\section`, `\subsection`, and `\subsubsection` have each their counter, but we'd like to refer to all of them by "section". The same for `\paragraph` and `\subparagraph`. There is one relevant subtlety to grouping multiple counters under the same type: in order for us to be able to meaningfully sort and compress this group, the set of counters contained therein cannot be arbitrary. Indeed, all of the *counters* grouped in the same *type* must belong to the same counter reset chain, and must be nested within each other (they cannot even just share the same parent). The need to check this has some implications to the data we store in the label. Since we cannot do this verification when we set up the *reference type*, because at this point we could only check existing counters, and they may be defined "later" or "never", the counter reset chain must be stored (names and values) with the label itself (this is done in properties `zc@enclcnt` and `zc@enclval`).

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

Regarding `\newtheorem`, cleveref deals with this by redefining its internals and retrieving the environment's name, to infer the type and do an "automatic definition" of theorem-like environments with a reasonable default. But even then, it can only provide the singular form of the cross-reference name, and if the plural is ever needed, the name has to be provided manually anyway. It also imposes the restriction of `\newtheorem` only being used in the preamble, which in itself would be good practice, but `\newtheorem`

is documented to be allowed anywhere in the document (see `texdoc source2e`, section 'ltthm.dtx', comment at the definition of `\newtheorem`). And the calls to `\newtheorem` must also come after cleveref is loaded. And for this to work, either ntheorem or amsthm must be loaded (as stated in the "Non-Bugs" section of the documentation). This automatism is, of course, a good thing, but the restrictions are considerable.

A related mechanism cleveref provides for overriding individual labels is by adding optional arguments to both `\label` and `\refstepcounter` which receives a "counter override label type" and stores that *instead* of regular counter with the `\newlabel` in the `.aux` file. This affords for a fully manual "one time" counter override for that particular label.

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

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

The use case for the optional argument for `\label` and `\refstepcounter` I do not quite grasp, and it does introduce ample opportunity for users to shoot themselves in the foot. Still, an equivalent could be provided by, for example, defining a document command `\zclabel[`⟨*type*⟩`]{`⟨*label*⟩`}` which makes a group, sets the type variable, and calls `\zlabel` with ⟨*label*⟩. Anyway, for the time being, I'll provide the higher level infrastructure, covering `\newtheorem` and `\appendix`, and only introduce a manual override of the sort if the need indeed arises and is well justified.

4.2 Messages

```
97 \msg_new:nnn { zref-clever } { type-name-length }
98   {
99     Type~'name'~can~only~receive~2~or~4~values~\msg_line_context:.~
```

```
    Don't~know~what~to~do~with~#1.
  }
\msg_new:nnn { zref-clever } { option-not-type-specific }
  {
    Option~'#1'~is~not~type-specific~\msg_line_context:.~
    Set~it~in~'\exp_not:N \zcDeclareTranslations'~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~or~in~'\exp_not:N \zcRefTypeSetup'.
  }
\msg_new:nnn  { zref-clever } { countertype-requires-value }
  { The~'countertype'~key~'#1'~requires~a~value. }
\msg_new:nnn  { zref-clever } { counterresetby-requires-value }
  { The~'counterresetby'~key~'#1'~requires~a~value. }
\msg_new:nnn { zref-clever } { missing-zref-titleref }
  {
    Option~'ref=title'~requested~\msg_line_context:.~
    But~package~'zref-titleref'~is~not~loaded,~falling-back~to~default~'ref'.
  }
\msg_new:nnn { zref-clever } { hyperref-preamble-only }
  {
    Option~'hyperref'~only~available~in~the~preamble. \iow_newline:
    Use~the~starred~version~of~'\noexpand\zcheck'~instead.
  }
\msg_new:nnn { zref-clever } { missing-hyperref }
  { Missing~'hyperref'~package. \iow_newline: Setting~'hyperref=false'. }
\msg_new:nnn { zref-clever } { counters-not-nested }
  { Counters~not~nested~for~labels~'#1'~and~'#2'~\msg_line_context:. }
\msg_new:nnn { zref-clever } { missing-type }
  { Reference~type~undefined~for~label~'#1'~\msg_line_context:. }
\msg_new:nnn { zref-clever } { missing-name }
  { Name~undefined~for~type~'#1'~\msg_line_context:. }
\msg_new:nnn { zref-clever } { single-element-range }
  { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:. }
```

4.3 Translations aux

Some wrappers around translations functions, so that we can generate variants with expansion control for arguments, or for convenience.

```
\prg_new_conditional:Npnn \__zrefclever_if_translation:nn #1#2 { p , TF }
  {
    \IfTranslation {#1} {#2}
      { \prg_return_true:  }
      { \prg_return_false: }
  }
\prg_generate_conditional_variant:Nnn \__zrefclever_if_translation:nn { xx } { p , TF }
\cs_new_protected:Npn \__zrefclever_get_translation_for:nnn #1#2#3
  { \SaveTranslationFor{#1}{#2}{#3} }
\cs_generate_variant:Nn \__zrefclever_get_translation_for:nnn { nxx }
\cs_new_protected:Npn \__zrefclever_declare_translation:nnn #1#2#3
  { \declaretranslation {#1} {#2} {#3} }
```

```
\cs_generate_variant:Nn \__zrefclever_declare_translation:nnn { xxn , xxx }

% <lang><key><transl>
\cs_new_protected:Npn \__zrefclever_add_default_translation:nnn #1#2#3
  { \addtranslation {#1} { zrefclever-default- #2 } {#3} }

% <lang><type><key><transl>
\cs_new_protected:Npn \__zrefclever_add_type_translation:nnnn #1#2#3#4
  { \addtranslation {#1} { zrefclever-type- #2 - #3 } {#4} }
```

Functions for use in dictionary files. The dictionary file commands cannot rely on expl3 syntax, so we define "document" ones.

```
% <key><transl>
\NewDocumentCommand \zcDicDefaultTransl { m m }
  { \ProvideDictTranslation { zrefclever-default- #1 } {#2} }
% <type><key><transl>
\NewDocumentCommand \zcDicTypeTransl { m m m }
  { \ProvideDictTranslation { zrefclever-type- #1 - #2 } {#3} }
```

4.4 Options

countertype option

\l__zrefclever_counter_type_prop

Variable storing a mapping from "counter" to "reference type".

```
\prop_new:N \l__zrefclever_counter_type_prop
```

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

```
\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} }
  }
\keys_define:nn { zref-clever }
  {
    countertype .code:n =
      {
        \keyval_parse:nnn
          { \msg_warning:nnn { zref-clever } { countertype-requires-value } }
          { \__zrefclever_prop_put_non_empty:Nnn \l__zrefclever_counter_type_prop }
          {#1}
      } ,
    countertype .value_required:n = true ,
    countertype .initial:n =
      {
        part          = part ,
        chapter       = chapter ,
        section       = section ,
        subsection    = section ,
        subsubsection = section ,
        paragraph     = paragraph ,
        subparagraph  = paragraph ,
        figure        = figure ,
        table         = table ,
```

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

counterresetters option

\l__zrefclever_counter_resetters_seq Stores the list of counters which are potential "enclosing counters" for other counters.

```
\seq_new:N \l__zrefclever_counter_resetters_seq
```

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

```
\keys_define:nn { zref-clever }
  {
    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 ,
      },
    typesort .value_required:n = true ,
  }
```

counterresetby option

\l__zrefclever_counter_resetby_prop Variable storing a mapping from "counter" to the counter which resets it.

```
\prop_new:N \l__zrefclever_counter_resetby_prop
```

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

```
\keys_define:nn { zref-clever }
  {
    counterresetby .code:n =
      {
        \keyval_parse:nnn
          { \msg_warning:nnn { zref-clever } { counterresetby-requires-value } }
          { \__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. TODO This list should probably be extended for 'enumitem', conditioned on it being loaded.

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

`ref` option

Stores whether this reference is to the page, or to the default counter.

```
\tl_new:N \l__zrefclever_ref_property_tl
\bool_new:N \l__zrefclever_page_ref_bool
\keys_define:nn { zref-clever }
  {
    ref .choice: ,
    ref / zc@thecnt .code:n =
      {
        \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
        \bool_set_false:N \l__zrefclever_page_ref_bool
      } ,
    ref / page .code:n =
      {
        \tl_set:Nn \l__zrefclever_ref_property_tl { page }
        \bool_set_true:N \l__zrefclever_page_ref_bool
      } ,
    ref / title .code:n =
      {
        \AddToHook { begindocument }
          {
            \@ifpackageloaded { zref-titleref }
              {
                \tl_set:Nn \l__zrefclever_ref_property_tl { title }
                \bool_set_false:N \l__zrefclever_page_ref_bool
              }
              {
                \msg_warning:nn { zref-clever } { missing-zref-titleref }
                \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
                \bool_set_false:N \l__zrefclever_page_ref_bool
              }
          }
      } ,
    ref .initial:n = zc@thecnt ,
    ref .value_required:n = true ,
    page .meta:n = { ref = page },
    page .value_forbidden:n = true ,
  }

\AddToHook { begindocument }
```

```
{
  \@ifpackageloaded { zref-titleref }
    {
      \keys_define:nn { zref-clever }
        {
          ref / title .code:n =
            {
              \tl_set:Nn \l__zrefclever_ref_property_tl { title }
              \bool_set_false:N \l__zrefclever_page_ref_bool
            }
        }
    }
    {
      \keys_define:nn { zref-clever }
        {
          ref / title .code:n =
            {
              \msg_warning:nn { zref-clever } { missing-zref-titleref }
              \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
              \bool_set_false:N \l__zrefclever_page_ref_bool
            }
        }
    }
}
```

Currently, we restrict 'ref=' to these two (or three) alternatives, but there might be a case for making this more flexible. The infrastructure can already handle receiving an arbitrary property, as long as one is satisfied with sorting and compressing from the default counter. If more flexibility is granted, 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 (see https://github.com/ho-tex/zref/issues/13, thanks Ulrike Fischer). Therefore, before adding anything to `\l__zrefclever_ref_property_tl`, check if first here with `\zref@ifpropundefined`: close it at the door.

typeset option

```
\bool_new:N \l__zrefclever_typeset_ref_bool
\bool_new:N \l__zrefclever_typeset_name_bool
\keys_define:nn { zref-clever }
  {
    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 ,
  }
```

`sort` option

User option, sort labels ranges or not

```
\bool_new:N \l__zrefclever_typeset_sort_bool
\keys_define:nn { zref-clever }
  {
    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

```
\seq_new:N \l__zrefclever_typesort_seq
\keys_define:nn { zref-clever }
  {
    typesort .code:n =
      {
        \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
        % Reverse the sequence, since the sort priorities are computed in the
        % negative range, so that we can implicitly rely on `0' being the
        % ``last value''.
        \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

User option, compress ranges or not

```
\bool_new:N \l__zrefclever_typeset_compress_bool
\keys_define:nn { zref-clever }
  {
    comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
    comp .initial:n = true ,
    comp .default:n = true ,
    nocomp .meta:n = { comp = false },
```

```
    nocomp .value_forbidden:n = true ,
  }
```

range option

```
\bool_new:N \l__zrefclever_typeset_range_bool
\keys_define:nn { zref-clever }
  {
    range .bool_set:N = \l__zrefclever_typeset_range_bool ,
    range .initial:n = false ,
    range .default:n = true ,
  }
```

hyperref option

\l__zrefclever_use_hyperref_bool
\l__zrefclever_warn_hyperref_bool

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

(*End definition for* `\l__zrefclever_use_hyperref_bool` *and* `\l__zrefclever_warn_hyperref_bool`.)

```
\AddToHook { begindocument }
  {
    \@ifpackageloaded { hyperref }
      {
        \bool_if:NT \l__zrefclever_use_hyperref_bool
          { \RequirePackage { zref-hyperref } }
      }
      {
        \bool_if:NT \l__zrefclever_warn_hyperref_bool
          { \msg_warning:nn { zref-clever } { missing-hyperref } }
        \bool_set_false:N \l__zrefclever_use_hyperref_bool
      }
    \keys_define:nn { zref-clever }
      {
```

```
        hyperref .code:n =
          { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }
      }
  }
```

nameinlink option

\l__zrefclever_nameinlink_tl

```
\str_new:N \l__zrefclever_nameinlink_str
\keys_define:nn { zref-clever }
  {
    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 ,
  }
```

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

cap capfirst options

```
\bool_new:N \l__zrefclever_capitalize_bool
\bool_new:N \l__zrefclever_capitalize_first_bool
\keys_define:nn { zref-clever }
  {
    cap .bool_set:N = \l__zrefclever_capitalize_bool ,
    cap .initial:n = false ,
    cap .default:n = true ,
    nocap .meta:n = { cap = false },
    nocap .value_forbidden:n = true ,

    capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,
    capfirst .initial:n = false ,
    capfirst .default:n = true ,

    C .meta:n =
      { capfirst = true , noabbrevfirst = true },
    C .value_forbidden:n = true ,
  }
```

abbrev noabbrevfirst option

```
\bool_new:N \l__zrefclever_abbrev_bool
\bool_new:N \l__zrefclever_noabbrev_first_bool
\keys_define:nn { zref-clever }
  {
    abbrev .bool_set:N = \l__zrefclever_abbrev_bool ,
    abbrev .initial:n = false ,
```

```
    abbrev .default:n = true ,
    noabbrev .meta:n = { abbrev = false },
    noabbrev .value_forbidden:n = true ,

    noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
    noabbrevfirst .initial:n = false ,
    noabbrevfirst .default:n = true ,
  }
```

lang option

```
\tl_new:N \l__zrefclever_ref_language_tl
\NewHook { zref-clever / reflanguage }
\keys_define:nn { zref-clever }
  {
    lang .code:n =
      {
        \AddToHook { zref-clever / reflanguage }
          {
            \str_case:nnF {#1}
              {
                { main }
                {
                  \tl_set_eq:NN
                    \l__zrefclever_ref_language_tl \l__zrefclever_main_language_tl
                }

                { current }
                {
                  \tl_set_eq:NN
                    \l__zrefclever_ref_language_tl \l__zrefclever_current_language_tl
                }
              }
              { \tl_set:Nn \l__zrefclever_ref_language_tl {#1} }
          }
      } ,
    lang .initial:n = main ,
    lang .value_required:n = true ,
  }
```

\AtEndOfPackage so that it comes after \ProcessKeysOptions.

```
\AtEndOfPackage
  {
    \AddToHook { zref-clever / reflanguage }
      {
        \keys_define:nn { zref-clever }
          {
            lang .code:n =
              {
                \str_case:nnF {#1}
                  {
                    { main }
                    {
                      \tl_set_eq:NN
                        \l__zrefclever_ref_language_tl \l__zrefclever_main_language_tl
```

```
                    }

                    { current }
                    {
                      \tl_set_eq:NN
                        \l__zrefclever_ref_language_tl \l__zrefclever_current_language_tl
                    }
                  }
                  { \tl_set:Nn \l__zrefclever_ref_language_tl {#1} }
              } ,
            lang .initial:n = main ,
            lang .value_required:n = true ,
          }
      }
  }
```

See https://tex.stackexchange.com/a/233178 (including Javier Bezos' comment). Also https://tex.stackexchange.com/a/281220 (including PLK's comments).

```
\AddToHook { begindocument / before }
  {
    % An internal alias for \pkg{translations}'s internal macro
    % \cs{@trnslt@current@language}.
    \tl_set_eq:NN \l__zrefclever_current_language_tl \@trnslt@current@language
    % Getting main languages and, for each babel/polyglossia loaded language,
    % load corresponding zref-clever dictionary.
    \@ifpackageloaded{babel}
      {
        \tl_set_eq:NN \l__zrefclever_main_language_tl \bbl@main@language
        \clist_map_inline:Nn \bbl@loaded
          {
            % Funny enough, \pkg{translations} also loads its basic
            % dictionaries for all languages loaded by babel or polyglossia.
            % First, there is no way to disable this, even if we don't need
            % them at all here.  Second, \pkg{translations} sends messages of
            % its own missing dictionaries to 'info' and everyone else's to
            % 'warning'\dots{}  So we have to control ourselves for missing
            % dictionaries and load them only if available.
            \exp_args:Nx \file_if_exist:nT
              { zref-clever- \@trnslt@language {#1} .trsl }
              { \LoadDictionaryFor {#1} { zref-clever } }
          }
      }
      {
        \@ifpackageloaded{polyglossia}
          {
            \tl_set_eq:NN \l__zrefclever_main_language_tl \xpg@main@language
            \clist_map_inline:Nn \xpg@loaded
              {
                \exp_args:Nx \file_if_exist:nT
                  { zref-clever- \@trnslt@language {#1} .trsl }
                  { \LoadDictionaryFor {#1} { zref-clever } }
              }
          }
          {
            \tl_new:N \l__zrefclever_main_language_tl
```

```
            \tl_set:Nn \l__zrefclever_main_language_tl { english }
            \LoadDictionaryFor { english } { zref-clever }
          }
      }
    % *Then* we execute the package options stored in the 'reflanguage' hook.
    \UseHook { zref-clever / reflanguage }
  }
```

`note` option

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

Reference options

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

Only not necessarily type-specific options are pertinent here.

```
\prop_new:N \l__zrefclever_ref_options_prop
\clist_map_inline:nn
  {
    % Not type-specific options.
    tpairsep ,
    tlistsep ,
    tlastsep ,
    notesep ,
    % Possibly type-specific options.
    namefont ,
    namesep ,
    pairsep ,
    listsep ,
    lastsep ,
    rangesep ,
    reffont ,
    refpre ,
    refpos ,
    reffont-in ,
    refpre-in ,
    refpos-in ,
  }
  {
    \keys_define:nn { zref-clever }
      {
        #1 .default:V = \c_novalue_tl ,
        #1 .code:n =
          {
            \tl_if_novalue:nTF {##1}
              { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
              { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
          } ,
      }
```

```
}
```

Package options

Process load-time package options (https://tex.stackexchange.com/a/15840).

```
\RequirePackage { l3keys2e }
\ProcessKeysOptions { zref-clever }
```

\zcsetup Provide \zcsetup.

```
\NewDocumentCommand \zcsetup { m }
  { \keys_set:nn { zref-clever } {#1} }
```

(*End definition for* \zcsetup.)

5 Type format

5.1 \zcRefTypeSetup

\l__zrefclever_setup_type_tl \l__zrefclever_setup_language_tl Variables storing the language and type to be used in \zcRefTypeSetup and \zcDeclareTranslations.

```
\tl_new:N \l__zrefclever_setup_type_tl
\tl_new:N \l__zrefclever_setup_language_tl
```

(*End definition for* \l__zrefclever_setup_type_tl *and* \l__zrefclever_setup_language_tl.)

\zcRefTypeSetup Provide \zcRefTypeSetup.

```
\NewDocumentCommand \zcRefTypeSetup { m m }
  {
    \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
      { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
    \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
    \keys_set:nn { zref-clever / typesetup } {#2}
  }
```

(*End definition for* \zcRefTypeSetup.)

Inside \zcRefTypeSetup any of the options *can* receive empty values, and those values, if they exist in the property list, will override translations, regardless of their emptiness. In principle, we could live with the situation of, once a setting has made \l__zrefclever_type_<type>_options_prop or \l__zrefclever_ref_options_prop it stays there forever, and can only be overridden by a new value at the same precedence level or a higher one. But it would be nice if an user can "unset" an option at either of those to go back to the lower precedence level of the translations at any given point. So both in \zcRefTypeSetup and in setting reference options, we leverage the distinction of an "empty valued key" (key= or key=) from a "key with no value" (key). This distinction is captured internally by the lower-level key parsing, but must be made explicit at \keys_set:nn by means of the .default: property of the key in \keys_define:nn. For the technique, see https://tex.stackexchange.com/q/614690 (thanks to Jonathan P. Spratte, aka Skillmon, and Phelype Oleinik).

name: a special convenience "short" way to set name options. Necessarily type-specific options.

```
\keys_define:nn { zref-clever / typesetup }
  {
    name .default:V = \c_novalue_tl ,
    name .code:n =
```

```
    {
      \tl_if_novalue:nTF {#1}
        {
          \clist_map_inline:nn
            {
              name-sg ,
              name-pl ,
              Name-sg ,
              Name-pl ,
              name-ab-sg ,
              name-ab-pl ,
              Name-ab-sg ,
              Name-ab-pl ,
            }
            {
              \prop_remove:cn
                { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
                {##1}
            }
        }
        {
          \int_case:nnF { \clist_count:n {#1} }
            {
              { 2 }
              {
                \clist_map_inline:nn
                  {
                    name-sg ,
                    Name-sg ,
                    name-ab-sg ,
                    Name-ab-sg ,
                  }
                  {
                    \prop_put:cnx
                      { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
                      {##1} { \clist_item:nn {#1} { 1 } }
                  }
                \clist_map_inline:nn
                  {
                    name-pl ,
                    Name-pl ,
                    name-ab-pl ,
                    Name-ab-pl ,
                  }
                  {
                    \prop_put:cnx
                      { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
                      {##1} { \clist_item:nn {#1} { 2 } }
                  }
              }

              { 4 }
              {
                % Make the first pair the capitalized ones, so as to make
```

```
                % them "feel" the default for the single pair case.
                \prop_put:cnx
                  { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
                  { Name-sg } { \clist_item:nn {#1} { 1 } }
                \prop_put:cnx
                  { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
                  { Name-ab-sg } { \clist_item:nn {#1} { 1 } }
                \prop_put:cnx
                  { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
                  { Name-pl } { \clist_item:nn {#1} { 2 } }
                \prop_put:cnx
                  { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
                  { Name-ab-pl } { \clist_item:nn {#1} { 2 } }
                \prop_put:cnx
                  { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
                  { name-sg } { \clist_item:nn {#1} { 3 } }
                \prop_put:cnx
                  { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
                  { name-ab-sg } { \clist_item:nn {#1} { 3 } }
                \prop_put:cnx
                  { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
                  { name-pl } { \clist_item:nn {#1} { 4 } }
                \prop_put:cnx
                  { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
                  { name-ab-pl } { \clist_item:nn {#1} { 4 } }
              }
            }
            {
              \msg_warning:nnx { zref-clever } { type-name-length }
                { \clist_count:n {#1} }
            }
        }
    } ,
}
```

Not type-specific options.

```
\clist_map_inline:nn
  {
    tpairsep ,
    tlistsep ,
    tlastsep ,
    notesep ,
  }
  {
    \keys_define:nn { zref-clever / typesetup }
      {
        #1 .code:n =
          {
            \msg_warning:nnn { zref-clever } { option-not-type-specific } {#1}
          } ,
      }
  }
```

Possibly or necessarily type-specific options.

```
\clist_map_inline:nn
```

```
  {
    % Possibly type-specific options.
    namefont ,
    namesep ,
    pairsep ,
    listsep ,
    lastsep ,
    rangesep ,
    reffont ,
    refpre ,
    refpos ,
    reffont-in ,
    refpre-in ,
    refpos-in ,
    % Necessarily type-specific options.
    name-sg ,
    name-pl ,
    Name-sg ,
    Name-pl ,
    name-ab-sg ,
    name-ab-pl ,
    Name-ab-sg ,
    Name-ab-pl ,
  }
  {
    \keys_define:nn { zref-clever / typesetup }
      {
        #1 .default:V = \c_novalue_tl ,
        #1 .code:n =
          {
            \tl_if_novalue:nTF {##1}
              {
                \prop_remove:cn
                  { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
                  {#1}
              }
              {
                \prop_put:cnn
                  { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
                  {#1} {##1}
              }
          } ,
      }
  }
```

5.2 \zcDeclareTranslations

\zcDeclareTranslations Provide \zcDeclareTranslations.

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

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

```
\keys_define:nn { zref-clever / translations }
  {
    type .code:n =
      {
        \tl_if_empty:nTF {#1}
          { \tl_clear:N \l__zrefclever_setup_type_tl }
          {
            \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
              { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
            \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
          }
      } ,
  }
```

`name`: a special convenience "short" way to set name options. Necessarily type-specific options.

```
\keys_define:nn { zref-clever / translations }
  {
    name .value_required:n = true ,
    name .code:n =
      {
        \tl_if_empty:NTF \l__zrefclever_setup_type_tl
          {
            \msg_warning:nnn { zref-clever }
              { option-only-type-specific } {#1}
          }
          {
            \int_case:nnF { \clist_count:n {#1} }
              {
                { 2 }
                {
                  \clist_map_inline:nn
                    {
                      name-sg ,
                      Name-sg ,
                      name-ab-sg ,
                      Name-ab-sg ,
                    }
                    {
                      \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_
                        { zrefclever-type- \l__zrefclever_setup_type_tl - ##1 }
                        { \clist_item:nn {#1} { 1 } }
                    }
                  \clist_map_inline:nn
                    {
                      name-pl ,
                      Name-pl ,
                      name-ab-pl ,
                      Name-ab-pl ,
                    }
                    {
                      \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_
                        { zrefclever-type- \l__zrefclever_setup_type_tl - ##1 }
```

```
                        { \clist_item:nn {#1} { 2 } }
                    }
                }

                { 4 }
                {
                  % Make the first pair the capitalized ones, so as to make
                  % them "feel" the default for the single pair case.
                  \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_tl }
                    { zrefclever-type- \l__zrefclever_setup_type_tl -Name-sg }
                    { \clist_item:nn {#1} { 1 } }
                  \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_tl }
                    { zrefclever-type- \l__zrefclever_setup_type_tl -Name-ab-sg }
                    { \clist_item:nn {#1} { 1 } }
                  \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_tl }
                    { zrefclever-type- \l__zrefclever_setup_type_tl -Name-pl }
                    { \clist_item:nn {#1} { 2 } }
                  \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_tl }
                    { zrefclever-type- \l__zrefclever_setup_type_tl -Name-ab-pl }
                    { \clist_item:nn {#1} { 2 } }
                  \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_tl }
                    { zrefclever-type- \l__zrefclever_setup_type_tl -name-sg }
                    { \clist_item:nn {#1} { 3 } }
                  \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_tl }
                    { zrefclever-type- \l__zrefclever_setup_type_tl -name-ab-sg }
                    { \clist_item:nn {#1} { 3 } }
                  \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_tl }
                    { zrefclever-type- \l__zrefclever_setup_type_tl -name-pl }
                    { \clist_item:nn {#1} { 4 } }
                  \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_tl }
                    { zrefclever-type- \l__zrefclever_setup_type_tl -name-ab-pl }
                    { \clist_item:nn {#1} { 4 } }
                }
              }
              {
                \msg_warning:nnx { zref-clever } { type-name-length }
                  { \clist_count:n {#1} }
              }
          }
      } ,
  }
```

Not type-specific options.

```
\clist_map_inline:nn
  {
    tpairsep ,
    tlistsep ,
    tlastsep ,
    notesep ,
  }
  {
    \keys_define:nn { zref-clever / translations }
      {
        #1 .value_required:n = true ,
        #1 .code:n =
```

```
          {
            \tl_if_empty:NTF \l__zrefclever_setup_type_tl
              {
                \__zrefclever_declare_translation:xxn { \l__zrefclever_setup_language_tl }
                  { zrefclever-default- #1 } {##1}
              }
              {
                \msg_warning:nnn { zref-clever }
                  { option-not-type-specific } {#1}
              }
          } ,
      }
  }
```

Possibly type-specific options.

```
\clist_map_inline:nn
  {
    namesep ,
    pairsep ,
    listsep ,
    lastsep ,
    rangesep ,
    refpre ,
    refpos ,
    refpre-in ,
    refpos-in ,
  }
  {
    \keys_define:nn { zref-clever / translations }
      {
        #1 .value_required:n = true ,
        #1 .code:n =
          {
            \tl_if_empty:NTF \l__zrefclever_setup_type_tl
              {
                \__zrefclever_declare_translation:xxn { \l__zrefclever_setup_language_tl }
                  { zrefclever-default- #1 } {##1}
              }
              {
                \__zrefclever_declare_translation:xxn { \l__zrefclever_setup_language_tl }
                  { zrefclever-type- \l__zrefclever_setup_type_tl - #1 } {##1}
              }
          } ,
      }
  }
```

Necessarily type-specific options.

```
\clist_map_inline:nn
  {
    name-sg ,
    name-pl ,
    Name-sg ,
    Name-pl ,
    name-ab-sg ,
    name-ab-pl ,
```

```
    Name-ab-sg ,
    Name-ab-pl ,
  }
  {
    \keys_define:nn { zref-clever / translations }
      {
        #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}
              }
              {
                \__zrefclever_declare_translation:xxn { \l__zrefclever_setup_language_tl }
                  { zrefclever-type- \l__zrefclever_setup_type_tl - #1 } {##1}
              }
          } ,
      }
  }
```

6 \zcref

\zcref

\zcref⟨*⟩[⟨*options*⟩]{⟨*labels*⟩}

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

(*End definition for* \zcref.)

\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.)

__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:
      \keys_set:nn { zref-clever } {#3}
      \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
      \bool_set:Nn \l__zrefclever_link_star_bool {#2}
      \bool_lazy_or:nnT
        { \l__zrefclever_typeset_sort_bool }
        { \l__zrefclever_typeset_range_bool }
        { \__zrefclever_sort_labels: }
      \__zrefclever_typeset_refs:
```

```
      % Typeset \texttt{note}.
      \l__zrefclever_notesep_tl
      \l__zrefclever_zcref_note_tl
    \group_end:
  }
```

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

7 \zcpageref

\zcpageref

`\zcpageref`⟨*⟩[⟨*options*⟩]{⟨*labels*⟩}

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

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

8 Sorting

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

\l__zrefclever_label_a_tl
\l__zrefclever_label_b_tl
\l__zrefclever_label_type_a_tl
\l__zrefclever_label_type_b_tl
\l__zrefclever_label_enclcnt_a_tl
\l__zrefclever_label_enclcnt_b_tl
\l__zrefclever_label_enclval_a_tl
\l__zrefclever_label_enclval_b_tl

Aux variables, for use in sorting and typesetting. I could probably let go some of them in favor of `tmpa`/`tmpb`, but they do improve code readability.

```
\tl_new:N \l__zrefclever_label_a_tl
\tl_new:N \l__zrefclever_label_b_tl
\tl_new:N \l__zrefclever_label_type_a_tl
\tl_new:N \l__zrefclever_label_type_b_tl
\tl_new:N \l__zrefclever_label_enclcnt_a_tl
\tl_new:N \l__zrefclever_label_enclcnt_b_tl
\tl_new:N \l__zrefclever_label_enclval_a_tl
\tl_new:N \l__zrefclever_label_enclval_b_tl
```

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

\l__zrefclever_label_types_seq

Stores the order in which reference types appear in the label list supplied by the user in `\zcref`. This order is required as a "last resort" sort criterion between the reference types, for use in `\__zrefclever_sort_default_aux: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
    \bool_if:NF \l__zrefclever_page_ref_bool
      {
        \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.  The
                % reference to the "default" property/counter or to the page
                % are quite different from our perspective, they rely on
                % different fields and even use different information for
                % sorting, so we branch them here to specialized functions.
                \bool_if:NTF \l__zrefclever_page_ref_bool
                  { \__zrefclever_sort_page_aux:nn {##1} {##2} }
                  { \__zrefclever_sort_default_aux:nn {##1} {##2} }
              }
          }
      }
  }
```

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

`\__zrefclever_label_type_put_new_right:n` Auxiliary function used to store "new" label types (in order) as the sorting proceeds. It is expected to be run inside `\__zrefclever_sort_labels:`, and stores new types in `\l__zrefclever_label_types_seq`.

`\__zrefclever_label_type_put_new_right:n {`⟨*label*⟩`}`

```
\cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
  {
```

```
    \tl_set:Nx \l__zrefclever_label_type_a_tl
      { \zref@extractdefault {#1} { zc@type } { \c_empty_tl } }
    \tl_if_empty:NF \l__zrefclever_label_type_a_tl
      {
        \seq_if_in:NVF \l__zrefclever_label_types_seq \l__zrefclever_label_type_a_tl
          {
            \seq_put_right:NV
              \l__zrefclever_label_types_seq \l__zrefclever_label_type_a_tl
          }
      }
  }
```

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

`\l__zrefclever_sort_decided_bool` Auxiliary variable for `\__zrefclever_sort_default_aux: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`.)

`\tl_reverse_items:V` Variant not provided by the kernel.

```
\cs_generate_variant:Nn \tl_reverse_items:n { V }
```

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

`\__zrefclever_sort_default_aux:nn` The heavy-lifting function for sorting of existing 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_aux:nn {`⟨*label a*⟩`} {`⟨*label b*⟩`}`

```
\cs_new_protected:Npn \__zrefclever_sort_default_aux:nn #1#2
  {
    \tl_set:Nx \l__zrefclever_label_type_a_tl
      { \zref@extractdefault {#1} { zc@type } { \c_empty_tl } }
    \tl_set:Nx \l__zrefclever_label_type_b_tl
      { \zref@extractdefault {#2} { zc@type } { \c_empty_tl } }

    \bool_if:nTF
      {
        % The second label has a type, but the first doesn't, leave the
        % undefined first (to be more visible).
        \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
        ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
      }
      { \sort_return_same: }
      {
        \bool_if:nTF
          {
            % The first label has a type, but the second doesn't, bring the
            % second forward.
            ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
            \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
```

```
    }
    { \sort_return_swapped: }
    {
      \bool_if:nTF
        {
          % Both labels have a type\dots{}
          ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
          ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
        }
        {
          \tl_if_eq:NNTF \l__zrefclever_label_type_a_tl \l__zrefclever_label_type_b_tl
            {
              % \dots{} and it's the same type.
              \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
                { \zref@extractdefault {#1} { zc@enclcnt } { \c_empty_tl } }
              \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
                { \tl_reverse_items:V \l__zrefclever_label_enclcnt_a_tl }
              \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
                { \zref@extractdefault {#2} { zc@enclcnt } { \c_empty_tl } }
              \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
                { \tl_reverse_items:V \l__zrefclever_label_enclcnt_b_tl }
              \tl_set:Nx \l__zrefclever_label_enclval_a_tl
                { \zref@extractdefault {#1} { zc@enclval } { \c_empty_tl } }
              \tl_set:Nx \l__zrefclever_label_enclval_a_tl
                { \tl_reverse_items:V \l__zrefclever_label_enclval_a_tl }
              \tl_set:Nx \l__zrefclever_label_enclval_b_tl
                { \zref@extractdefault {#2} { zc@enclval } { \c_empty_tl } }
              \tl_set:Nx \l__zrefclever_label_enclval_b_tl
                { \tl_reverse_items:V \l__zrefclever_label_enclval_b_tl }

              \bool_set_false:N \l__zrefclever_sort_decided_bool
              % CHECK should I replace the tmp variables here?
              \tl_clear:N \l_tmpa_tl
              \tl_clear:N \l_tmpb_tl
              \bool_until_do:Nn \l__zrefclever_sort_decided_bool
                {
                  \tl_set:Nx \l_tmpa_tl
                    { \tl_head:N \l__zrefclever_label_enclcnt_a_tl }
                  \tl_set:Nx \l_tmpb_tl
                    { \tl_head:N \l__zrefclever_label_enclcnt_b_tl }

                  \bool_if:nTF
                    {
                      % Both are empty, meaning: neither labels have any
                      % (further) ``enclosing counters'' (left).
                      \tl_if_empty_p:V \l_tmpa_tl &&
                      \tl_if_empty_p:V \l_tmpb_tl
                    }
                    {
                      \exp_args:Nxx \tl_if_eq:nnTF
                        { \zref@extractdefault {#1} { zc@counter } { } }
                        { \zref@extractdefault {#2} { zc@counter } { } }
                        {
                          \bool_set_true:N \l__zrefclever_sort_decided_bool
```

```
      \int_compare:nNnTF
        { \zref@extractdefault {#1} { zc@cntval } {-1} }
          >
        { \zref@extractdefault {#2} { zc@cntval } {-1} }
        { \sort_return_swapped: }
        { \sort_return_same:    }
    }
    {
      \msg_warning:nnnn { zref-clever }
        { counters-not-nested } {#1} {#2}
      \bool_set_true:N \l__zrefclever_sort_decided_bool
      \sort_return_same:
    }
}
{
  \bool_if:nTF
    {
      % ‘a’ is empty (and ‘b’ is not), meaning: ‘b’
      % is (possibly) nested in ‘a’.
      \tl_if_empty_p:V \l_tmpa_tl
    }
    {
      \tl_set:Nx \l_tmpa_tl
        { {\zref@extractdefault {#1} { zc@counter } { }} }
      \exp_args:NNx \tl_if_in:NnTF
        \l__zrefclever_label_enclcnt_b_tl { \l_tmpa_tl }
        {
          \bool_set_true:N \l__zrefclever_sort_decided_bool
          \sort_return_same:
        }
        {
          \msg_warning:nnnn { zref-clever }
            { counters-not-nested } {#1} {#2}
          \bool_set_true:N \l__zrefclever_sort_decided_bool
          \sort_return_same:
        }
    }
    {
      \bool_if:nTF
        {
          % ‘b’ is empty (and ‘a’ is not), meaning:
          % ‘a’ is (possibly) nested in ‘b’.
          \tl_if_empty_p:V \l_tmpb_tl
        }
        {
          \tl_set:Nx \l_tmpb_tl
            { {\zref@extractdefault {#2} { zc@counter } { }} }
          \exp_args:NNx \tl_if_in:NnTF
            \l__zrefclever_label_enclcnt_a_tl { \l_tmpb_tl }
            {
              \bool_set_true:N \l__zrefclever_sort_decided_bool
              \sort_return_swapped:
            }
            {
```

```
                \msg_warning:nnnn { zref-clever }
                  { counters-not-nested } {#1} {#2}
                \bool_set_true:N \l__zrefclever_sort_decided_bool
                \sort_return_same:
              }
          }
          {
            % Neither is empty, meaning: we can
            % (possibly) 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.
            \tl_if_eq:NNTF \l_tmpa_tl \l_tmpb_tl
              {
                \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_enclcnt_a_tl
                      { \tl_tail:N \l__zrefclever_label_enclcnt_a_tl
                    \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
                      { \tl_tail:N \l__zrefclever_label_enclcnt_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_boo
                    \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:    }
                  }
              }
              {
                \msg_warning:nnnn { zref-clever }
                  { counters-not-nested } {#1} {#2}
                \bool_set_true:N \l__zrefclever_sort_decided_bool
                \sort_return_same:
              }
          }
      }
    }
  }
  {
    % \dots{} and they are different types.
    \int_zero:N \l__zrefclever_sort_prior_a_int
    \int_zero:N \l__zrefclever_sort_prior_b_int
```

```
        % \cs{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''.
        \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 } }
                  }
              }
          }
        \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 for different types:
                % the type that occurs first in \meta{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: } }
                      }
                  }
              }
          }
      }
  }
```

```
            {
              % Neither of the labels has a type.  We can't do much of
              % meaningful here, but if it's the same counter, compare it.
              \exp_args:Nxx \tl_if_eq:nnTF
                { \zref@extractdefault {#1} { zc@counter } { } }
                { \zref@extractdefault {#2} { zc@counter } { } }
                {
                  \int_compare:nNnTF
                    { \zref@extractdefault {#1} { zc@cntval } {-1} }
                      >
                    { \zref@extractdefault {#2} { zc@cntval } {-1} }
                    { \sort_return_swapped: }
                    { \sort_return_same:    }
                }
                { \sort_return_same: }
            }
        }
    }
}
```

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

`\__zrefclever_sort_page_aux:nn` The sorting function for sorting of existing 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_aux:nn {⟨label a⟩} {⟨label b⟩}`

```
\cs_new_protected:Npn \__zrefclever_sort_page_aux:nn #1#2
  {
    \int_compare:nNnTF
      { \zref@extractdefault {#1} { zc@abspg } {-1} }
        >
      { \zref@extractdefault {#2} { zc@abspg } {-1} }
      { \sort_return_swapped: }
      { \sort_return_same:    }
  }
```

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

9 Typesetting

About possible alternatives to signal compression inhibition for individual labels, see https://tex.stackexchange.com/q/611370 (thanks Enrico Gregorio, Phelype Oleinik, and Steven B. Segletes). Yet another alternative would be to receive an optional argument with the label(s) not to be compressed. This would be a repetition, but would keep the syntax "clean". All in all, and rethinking this here, probably the best is simply to not allow individual inhibition of compression. We can already control compression of each individual call of `\zcref` with existing options, this should be enough. I don't think the small extra flexibility this would grant is worth the syntax disruption it entails. Anyway, I have kept a "handle" to deal with this in case the need arises, in the form of `\l__zrefclever_range_inhibit_next_bool`, which is currently no-op, but is in place.

Typesetting variables

\l__zrefclever_typeset_last_bool
\l__zrefclever_last_of_type_bool

Auxiliary variables for `\__zrefclever_typeset_refs:`. `\l__zrefclever_typeset_last_bool` signals if the label list is over so that we can leave the loop. `\l__zrefclever_last_of_type_bool` signals if we are processing the last label of the current reference type.

```
\bool_new:N \l__zrefclever_typeset_last_bool
\bool_new:N \l__zrefclever_last_of_type_bool
```

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

\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:`. They store, respectively the "previous" and the "current" reference type information while they are being processed, since we cannot typeset them directly, given we can only know certain things when the (next) type list is over. The "queue" stores all references but the first of the type, and they are stored ready to be typeset. The "first_label" stores the *label* of the first reference for the type, because the name can only be determined at the end, and its (potential) hyperlink must be handled at that point.

```
\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_typeset_queue_prev_tl` *and others.*)

\l__zrefclever_label_count_int
\l__zrefclever_type_count_int

Main counters for `\__zrefclever_typeset_refs:`. They track the state of the parsing of the labels list. `\l__zrefclever_label_count_int` is stepped for every reference/label in the list, and reset at the start of a new type. `\l__zrefclever_type_count_int` is stepped at every reference type change.

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

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

\l__zrefclever_range_count_int
\l__zrefclever_range_same_count_int
\l__zrefclever_range_beg_label_tl
\l__zrefclever_next_maybe_range_bool
\l__zrefclever_next_is_same_bool
\l__zrefclever_range_inhibit_next_bool

Range related auxiliary variables for `\__zrefclever_typeset_refs:`. `\l__zrefclever_range_count_int` counts how many references/labels are in the current ongoing range. `\l__zrefclever_range_same_count_int` counts how many of the references in the current ongoing range are repeated ones. `\l__zrefclever_range_beg_label_tl` stores the label of the reference that starts a range. `\l__zrefclever_next_maybe_range_bool` signals whether the next element is in sequence to the current one. `\l__zrefclever_next_is_same_bool` signals whether the next element repeats the current one. `\l__zrefclever_range_inhibit_next_bool` allows to control/track compression inhibition of the next label.

```
\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_next_maybe_range_bool
\bool_new:N \l__zrefclever_next_is_same_bool
\bool_new:N \l__zrefclever_range_inhibit_next_bool
```

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

Aux variables for __zrefclever_typeset_refs:. Store separators and refpre/pos options.

```
\tl_new:N \l__zrefclever_namefont_tl
\tl_new:N \l__zrefclever_reffont_out_tl
\tl_new:N \l__zrefclever_reffont_in_tl

\tl_new:N \l__zrefclever_namesep_tl
\tl_new:N \l__zrefclever_rangesep_tl
\tl_new:N \l__zrefclever_pairsep_tl
\tl_new:N \l__zrefclever_listsep_tl
\tl_new:N \l__zrefclever_lastsep_tl
% 't' for ''type''
\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_notesep_tl
\tl_new:N \l__zrefclever_refpre_out_tl
\tl_new:N \l__zrefclever_refpos_out_tl
\tl_new:N \l__zrefclever_refpre_in_tl
\tl_new:N \l__zrefclever_refpos_in_tl
```

(*End definition for .*)

\l__zrefclever_type_name_tl
\l__zrefclever_name_in_link_bool
\l__zrefclever_name_format_tl

Auxiliary variables for __zrefclever_get_ref_first: and __zrefclever_type_name_setup:.

```
\tl_new:N \l__zrefclever_type_name_tl
\bool_new:N \l__zrefclever_name_in_link_bool
\tl_new:N \l__zrefclever_name_format_tl
```

(*End definition for* \l__zrefclever_type_name_tl, \l__zrefclever_name_in_link_bool, *and* \l__zrefclever_name_format_tl.)

Main typesetting functions

__zrefclever_typeset_refs:

Main typesetting function for \zcref.

```
\cs_new_protected:Npn \__zrefclever_typeset_refs:
  {
    \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
    \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
    \tl_clear:N \l__zrefclever_type_first_label_tl
    \tl_clear:N \l__zrefclever_type_first_label_type_tl
    \tl_clear:N \l__zrefclever_range_beg_label_tl
    \int_zero:N \l__zrefclever_label_count_int
    \int_zero:N \l__zrefclever_type_count_int
    \int_zero:N \l__zrefclever_range_count_int
    \int_zero:N \l__zrefclever_range_same_count_int

    % Get not-type-specific separators and refpre/pos options.
    \__zrefclever_get_option_with_transl:nN {tpairsep} \l__zrefclever_tpairsep_tl
    \__zrefclever_get_option_with_transl:nN {tlistsep} \l__zrefclever_tlistsep_tl
    \__zrefclever_get_option_with_transl:nN {tlastsep} \l__zrefclever_tlastsep_tl
    \__zrefclever_get_option_with_transl:nN {notesep}  \l__zrefclever_notesep_tl

    % Set the font option for this zcref call.
```

```
    \l__zrefclever_ref_typeset_font_tl

    % Loop over the label list in sequence.
    \bool_set_false:N \l__zrefclever_typeset_last_bool
    \bool_until_do:Nn \l__zrefclever_typeset_last_bool
      {
        \seq_pop_left:NN \l__zrefclever_zcref_labels_seq \l__zrefclever_label_a_tl
        \seq_if_empty:NTF \l__zrefclever_zcref_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_zcref_labels_seq \l__zrefclever_label_b_tl }

        \bool_if:NTF \l__zrefclever_page_ref_bool
          {
            \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
            \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
          }
          {
            \tl_set:Nx \l__zrefclever_label_type_a_tl
              {
                \zref@extractdefault
                  { \l__zrefclever_label_a_tl } { zc@type } { \c_empty_tl }
              }
            \tl_set:Nx \l__zrefclever_label_type_b_tl
              {
                \zref@extractdefault
                  { \l__zrefclever_label_b_tl } { zc@type } { \c_empty_tl }
              }
          }

        % 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.
                    \bool_if:nTF
                      % Both empty: same ``type''.
                      {
                        \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
```

```
                  \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
                }
                { \bool_set_false:N \l__zrefclever_last_of_type_bool }
                {
                  \bool_if:nTF
                    % Neither empty: compare types.
                    {
                      ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
                      ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
                    }
                    {
                      \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  }
                    }
                    % One empty, the other not: different ``types''.
                    { \bool_set_true:N \l__zrefclever_last_of_type_bool }
                }
            }
        }
    }

  % Handle warnings in case of reference or type undefined.
  \zref@refused { \l__zrefclever_label_a_tl }
  \zref@ifrefundefined { \l__zrefclever_label_a_tl }
    {}
    {
      \tl_if_empty:NT \l__zrefclever_label_type_a_tl
        {
          \msg_warning:nnx { zref-clever } { missing-type }
            { \l__zrefclever_label_a_tl }
        }
    }

  % Get type-specific separators, refpre/pos and font options, once per
  % type.
  \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
    {
      \__zrefclever_get_option_plain:nN {namefont}        \l__zrefclever_namefont_tl
      \__zrefclever_get_option_plain:nN {reffont}         \l__zrefclever_reffont_out_t
      \__zrefclever_get_option_plain:nN {reffont-in}      \l__zrefclever_reffont_in_tl
      \__zrefclever_get_option_with_transl:nN {namesep}   \l__zrefclever_namesep_tl
      \__zrefclever_get_option_with_transl:nN {rangesep}  \l__zrefclever_rangesep_tl
      \__zrefclever_get_option_with_transl:nN {pairsep}   \l__zrefclever_pairsep_tl
      \__zrefclever_get_option_with_transl:nN {listsep}   \l__zrefclever_listsep_tl
      \__zrefclever_get_option_with_transl:nN {lastsep}   \l__zrefclever_lastsep_tl
      \__zrefclever_get_option_with_transl:nN {refpre}    \l__zrefclever_refpre_out_tl
      \__zrefclever_get_option_with_transl:nN {refpos}    \l__zrefclever_refpos_out_tl
      \__zrefclever_get_option_with_transl:nN {refpre-in} \l__zrefclever_refpre_in_tl
      \__zrefclever_get_option_with_transl:nN {refpos-in} \l__zrefclever_refpos_in_tl
    }

  % Here we send this to a couple of auxiliary functions for no other
```

```
        % reason than to keep this long function a little less unreadable.
        \bool_if:NTF \l__zrefclever_last_of_type_bool
          {
            % There exists no next label of the same type as the current.
            \__zrefclever_typeset_refs_aux_last_of_type:
          }
          {
            % There exists a next label of the same type as the current.
            \__zrefclever_typeset_refs_aux_not_last_of_type:
          }
      }
  }
```

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

`__zrefclever_typeset_refs_aux_last_of_type:` Handles typesetting of when the current label is the last of its type.

```
\cs_new_protected:Npn \__zrefclever_typeset_refs_aux_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.
        { 0 }
        {
          \tl_set:NV \l__zrefclever_type_first_label_tl \l__zrefclever_label_a_tl
          \tl_set:NV \l__zrefclever_type_first_label_type_tl \l__zrefclever_label_type_a_tl
        }

        % The last is the second: we have a pair (if not repeated).
        { 1 }
        {
          \int_compare:nNnF { \l__zrefclever_range_same_count_int } = {1}
            {
              \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                {
                  \exp_not:V \l__zrefclever_pairsep_tl
                  \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
                }
            }
        }
      }
      % If neither the first, nor the second: we have the last label
      % on the current type list (if not repeated).
      {
        \int_case:nnF { \l__zrefclever_range_count_int }
          {
            % There was no range going on.
            {0}
            {
              \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                {
                  \exp_not:V \l__zrefclever_lastsep_tl
                  \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
```

```
            }
        }
        % Last in the range is also the second in it.
        {1}
        {
          \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
            {
              % We know 'range_beg_label' is not empty, since this is the
              % second element in the range, but the third or more in the
              % type list.
              \exp_not:V \l__zrefclever_listsep_tl
              \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
              \int_compare:nNnF { \l__zrefclever_range_same_count_int } = {1}
                {
                  \exp_not:V \l__zrefclever_lastsep_tl
                  \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
                }
            }
        }
    }
    % 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.
          {0}
          {
            % If 'range_beg_label' is empty, it means it was also the
            % first of the type, and hence was already handled.
            \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
              {
                \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                  {
                    \exp_not:V \l__zrefclever_lastsep_tl
                    \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
                  }
              }
          }
          % A ''range'', but with no skipped value, treat as list.
          {1}
          {
            \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
              {
                % Ditto.
                \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
                  {
                    \exp_not:V \l__zrefclever_listsep_tl
                    \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
                  }
                \exp_not:V \l__zrefclever_lastsep_tl
                \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
              }
          }
```

```
            }
            {
              % An actual range.
              \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                {
                  % Ditto.
                  \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
                    {
                      \exp_not:V \l__zrefclever_lastsep_tl
                      \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
                    }
                  \exp_not:V \l__zrefclever_rangesep_tl
                  \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
                }
            }
        }
    }

  % 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 \cs{l__zrefclever_label_a_tl} 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
          \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
            { }
            {
              \__zrefclever_labels_in_sequence:nn
                { \l__zrefclever_type_first_label_tl } { \l__zrefclever_label_a_tl }
            }
          \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
            {
              \bool_if:NTF \l__zrefclever_next_maybe_range_bool
                { \exp_not:V \l__zrefclever_pairsep_tl }
                { \exp_not:V \l__zrefclever_rangesep_tl }
              \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
            }
        }
    }

  % Now that the type is finished, we can add the name and the first ref to
  % the queue.  Or, if ``typset'' option is not ``both'', handle it here
  % too.
```

```
\__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 }
      {
        \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
          { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
      }
      {
        \bool_if:nTF
          { \l__zrefclever_typeset_name_bool }
          {
            \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
              {
                \bool_if:NTF \l__zrefclever_name_in_link_bool
                  {
                    \exp_not:N \group_begin:
                    \exp_not:V \l__zrefclever_namefont_tl
                    % It's two '@s', but escaped for DocStrip.
                    \exp_not:N \hyper@@link
                      {
                        \zref@ifrefcontainsprop
                          { \l__zrefclever_type_first_label_tl } { urluse }
                          {
                            \zref@extractdefault
                              { \l__zrefclever_type_first_label_tl }
                              { urluse } {}
                          }
                          {
                            \zref@extractdefault
                              { \l__zrefclever_type_first_label_tl }
                              { url } {}
                          }
                      }
                      {
                        \zref@extractdefault
                          { \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:
                  }
              }
```

```
            }
            {
              % This case would correspond to "typeset=none" but should not
              % happen, given the options are set up to typeset at least one
              % of "ref" or "name", but a sensible fallback, equal to the
              % behavior of ``both''.
              \tl_put_left:Nx
                \l__zrefclever_typeset_queue_curr_tl { \__zrefclever_get_ref_first: }
            }
        }
    }

  % Typeset the previous type, 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
    }

  % 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.
          { 0 }
          { \l__zrefclever_typeset_queue_curr_tl }
          % Pair of types.
          { 1 }
          {
            \l__zrefclever_tpairsep_tl
            \l__zrefclever_typeset_queue_curr_tl
          }
        }
        {
          % Last in list of types.
          \l__zrefclever_tlastsep_tl
          \l__zrefclever_typeset_queue_curr_tl
        }
    }
    {
      % 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
      \int_zero:N \l__zrefclever_label_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
```

```
      }
  }
```

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

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

```
\cs_new_protected:Npn \__zrefclever_typeset_refs_aux_not_last_of_type:
  {
    % Signal if next label may form a range with the current one (of
    % course, 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_lazy_and:nnT
      { \l__zrefclever_typeset_compress_bool }
      % Currently no-op, but kept as ``handle'' to inhibit compression of
      % individual labels.
      { ! \l__zrefclever_range_inhibit_next_bool }
      {
        \zref@ifrefundefined { \l__zrefclever_label_a_tl }
          { }
          {
            \__zrefclever_labels_in_sequence:nn
              { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
          }
      }

    % Process the current label to the current queue.
    \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
      {
        % Current label is the first of its type (also not the last, but it
        % doesn't matter here): just store the label.
        \tl_set:NV \l__zrefclever_type_first_label_tl \l__zrefclever_label_a_tl
        \tl_set:NV \l__zrefclever_type_first_label_type_tl \l__zrefclever_label_type_a_tl

        % If the next label may be part of a range, we set `range_beg_label'
        % to ``empty'' (we deal with it as the ``first'', and must do it
        % there, to handle hyperlinking), but also step the range counters.
        \bool_if:NT \l__zrefclever_next_maybe_range_bool
          {
            \tl_clear:N \l__zrefclever_range_beg_label_tl
            \int_incr:N \l__zrefclever_range_count_int
            \bool_if:NT \l__zrefclever_next_is_same_bool
              { \int_incr:N \l__zrefclever_range_same_count_int }
         }
      }
      {
        % 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
        \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, meaning: there was no range, or
    % we are closing one.
    \int_case:nnF { \l__zrefclever_range_count_int }
      {
        % There was no range going on.
        {0}
        {
          \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
            {
              \exp_not:V \l__zrefclever_listsep_tl
              \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
            }
        }
        % 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.
        {1}
        {
          \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
            {
              \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
                {
                  \exp_not:V \l__zrefclever_listsep_tl
                  \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
                }
              \int_compare:nNnF { \l__zrefclever_range_same_count_int } = {1}
                {
                  \exp_not:V \l__zrefclever_listsep_tl
                  \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
                }
            }
        }
      }
      {
        % 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 }
                  {
                    {0}
                    {
                      \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                        {
                          \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
                            {
                              \exp_not:V \l__zrefclever_listsep_tl
                              \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
                            }
                        }
                    }
                    {1}
                    {
                      \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                        {
                          \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
                            {
                              \exp_not:V \l__zrefclever_listsep_tl
                              \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
                            }
                          \exp_not:V \l__zrefclever_listsep_tl
                          \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
                        }
                    }
                  }
                  {
                    \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
                      {
                        \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
                          {
                            \exp_not:V \l__zrefclever_listsep_tl
                            \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
                          }
                        \exp_not:V \l__zrefclever_rangesep_tl
                        \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
                      }
                  }
              }
            % 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_aux_not_last_of_type:`.)

Aux typesetting functions

__zrefclever_get_ref:n

Auxiliary function to __zrefclever_typeset_refs:. Handles a complete "ref-block", including "pre" and "pos" elements, and *hyperlinking*. It does not handle the reference type "name", for that use __zrefclever_get_ref_first:. It should get the reference with \zref@extractdefault as usual but, if the reference is not available, should put \zref@default on the stream protected, so that it can be accumulated in the queue. \hyperlink must also be protected from expansion for the same reason.

```
\cs_new:Npn \__zrefclever_get_ref:n #1
  {
    \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
      {
        \bool_if:nTF
          { \l__zrefclever_use_hyperref_bool && ! \l__zrefclever_link_star_bool }
          {
            \exp_not:N \group_begin:
            \exp_not:V \l__zrefclever_reffont_out_tl
            \exp_not:V \l__zrefclever_refpre_out_tl
            \exp_not:N \group_begin:
            \exp_not:V \l__zrefclever_reffont_in_tl
            % It's two '@s', but escaped for DocStrip.
            \exp_not:N \hyper@@link
              {
                \zref@ifrefcontainsprop {#1} { urluse }
                  { \zref@extractdefault {#1} { urluse } {} }
                  { \zref@extractdefault {#1} { url } {} }
              }
              { \zref@extractdefault {#1} { anchor } {} }
              {
                \exp_not:V \l__zrefclever_refpre_in_tl
                \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } {}
                \exp_not:V \l__zrefclever_refpos_in_tl
              }
            \exp_not:N \group_end:
            \exp_not:V \l__zrefclever_refpos_out_tl
            \exp_not:N \group_end:
          }
          {
            \exp_not:N \group_begin:
            \exp_not:V \l__zrefclever_reffont_out_tl
            \exp_not:V \l__zrefclever_refpre_out_tl
            \exp_not:N \group_begin:
            \exp_not:V \l__zrefclever_reffont_in_tl
            \exp_not:V \l__zrefclever_refpre_in_tl
            \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } {}
            \exp_not:V \l__zrefclever_refpos_in_tl
            \exp_not:N \group_end:
            \exp_not:V \l__zrefclever_refpos_out_tl
            \exp_not:N \group_end:
          }
      }
      { \exp_not:N \zref@default }
  }
\cs_generate_variant:Nn \__zrefclever_get_ref:n { V }
```

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

`\__zrefclever_type_name_setup:` Auxiliary function to `\__zrefclever_typeset_refs:`. Sets the type name variable `\l__zrefclever_type_name_tl`. When it cannot be found, clears it.

```
\cs_new_protected:Npn \__zrefclever_type_name_setup:
  {
    \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
      { \tl_clear:N \l__zrefclever_type_name_tl }
      {
        \tl_if_empty:nTF \l__zrefclever_type_first_label_type_tl
          { \tl_clear:N \l__zrefclever_type_name_tl }
          {
```

Determine whether we should use capitalization, abbreviation, and plural.

```
            \bool_lazy_or:nnTF
              { \l__zrefclever_capitalize_bool }
              {
                \l__zrefclever_capitalize_first_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} }
            \bool_lazy_and:nnT
              { \l__zrefclever_abbrev_bool }
              {
                ! \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 } ||
                ! \l__zrefclever_noabbrev_first_bool
              }
              { \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab } }
            % 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 } }

            \prop_get:cVNF
              { l__zrefclever_type_ \l__zrefclever_type_first_label_type_tl _options_prop }
              \l__zrefclever_name_format_tl
              \l__zrefclever_type_name_tl
              {
                \__zrefclever_if_translation:xxTF
                  { \l__zrefclever_ref_language_tl }
                  {
                    zrefclever-type- \l__zrefclever_type_first_label_type_tl -
                    \l__zrefclever_name_format_tl
                  }
                  {
                    \__zrefclever_get_translation_for:nxx { \l__zrefclever_type_name_tl }
                      { \l__zrefclever_ref_language_tl }
                      {
                        zrefclever-type- \l__zrefclever_type_first_label_type_tl -
                        \l__zrefclever_name_format_tl
                      }
                  }
                  {
```

```
                  \tl_clear:N \l__zrefclever_type_name_tl
                  \msg_warning:nnx { zref-clever } { missing-name }
                    { \l__zrefclever_type_first_label_type_tl }
                }
            }
        }
    }
```

Signal whether the type name is to be included in the hyperlink or not.

```
  \bool_lazy_any:nTF
    {
      { ! \l__zrefclever_use_hyperref_bool }
      { \l__zrefclever_link_star_bool }
      { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
      { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
    }
    { \bool_set_false:N \l__zrefclever_name_in_link_bool }
    {
      \bool_lazy_any:nTF
        {
          { \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_get_ref_first:` Auxiliary function to `\__zrefclever_typeset_refs:`. Handles a complete "ref-block", including "pre" and "pos" elements, *hyperlinking*, and the reference type "name". For use on the first reference of each type.

```
\cs_new:Npn \__zrefclever_get_ref_first:
  {
    \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
      { \exp_not:N \zref@default }
      {
        \bool_if:NTF \l__zrefclever_name_in_link_bool
          {
            \zref@ifrefcontainsprop
              { \l__zrefclever_type_first_label_tl } { \l__zrefclever_ref_property_tl }
              {
                % It's two '@s', but escaped for DocStrip.
                \exp_not:N \hyper@@link
                  {
```

```
                \zref@ifrefcontainsprop
                  { \l__zrefclever_type_first_label_tl } { urluse }
                  {
                    \zref@extractdefault { \l__zrefclever_type_first_label_tl }
                      { urluse } {}
                  }
                  {
                    \zref@extractdefault { \l__zrefclever_type_first_label_tl }
                      { url } {}
                  }
              }
              {
                \zref@extractdefault { \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
                \exp_not:N \group_begin:
                \exp_not:V \l__zrefclever_reffont_out_tl
                \exp_not:V \l__zrefclever_refpre_out_tl
                \exp_not:N \group_begin:
                \exp_not:V \l__zrefclever_reffont_in_tl
                \exp_not:V \l__zrefclever_refpre_in_tl
                \zref@extractdefault { \l__zrefclever_type_first_label_tl }
                  { \l__zrefclever_ref_property_tl } {}
                \exp_not:V \l__zrefclever_refpos_in_tl
                \exp_not:N \group_end:
                % hyperlink makes it's own group, we'd like to close the
                % 'refpre-out' group after 'refpos-out', but... we close
                % it here, and give the trailing 'refpos-out' its own
                % group.  This will result that formatting given to
                % 'refpre-out' will not reach 'refpos-out', but I see no
                % alternative, and this has to be handled specially.
                \exp_not:N \group_end:
              }
            \exp_not:N \group_begin:
            % Ditto: special treatment.
            \exp_not:V \l__zrefclever_reffont_out_tl
            \exp_not:V \l__zrefclever_refpos_out_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:
            \exp_not:V \l__zrefclever_namesep_tl
            \exp_not:N \zref@default
          }
      }
```

```
{
  \tl_if_empty:NTF \l__zrefclever_type_name_tl
    {
      \exp_not:N \zref@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:
      \exp_not:V \l__zrefclever_namesep_tl
    }
  \zref@ifrefcontainsprop
    { \l__zrefclever_type_first_label_tl } { \l__zrefclever_ref_property_tl }
    {
      \bool_if:nTF
        {
          \l__zrefclever_use_hyperref_bool &&
          ! \l__zrefclever_link_star_bool
        }
        {
          \exp_not:N \group_begin:
          \exp_not:V \l__zrefclever_reffont_out_tl
          \exp_not:V \l__zrefclever_refpre_out_tl
          \exp_not:N \group_begin:
          \exp_not:V \l__zrefclever_reffont_in_tl
          % It's two '@s', but escaped for DocStrip.
          \exp_not:N \hyper@@link
            {
              \zref@ifrefcontainsprop
                { \l__zrefclever_type_first_label_tl } { urluse }
                {
                  \zref@extractdefault { \l__zrefclever_type_first_label_tl }
                    { urluse } {}
                }
                {
                  \zref@extractdefault { \l__zrefclever_type_first_label_tl }
                    { url } {}
                }
            }
            {
              \zref@extractdefault { \l__zrefclever_type_first_label_tl }
                { anchor } {}
            }
            {
              \exp_not:V \l__zrefclever_refpre_in_tl
              \zref@extractdefault { \l__zrefclever_type_first_label_tl }
                { \l__zrefclever_ref_property_tl } {}
              \exp_not:V \l__zrefclever_refpos_in_tl
            }
          \exp_not:N \group_end:
          \exp_not:V \l__zrefclever_refpos_out_tl
          \exp_not:N \group_end:
```

```
                  }
                  {
                    \exp_not:N \group_begin:
                    \exp_not:V \l__zrefclever_reffont_out_tl
                    \exp_not:V \l__zrefclever_refpre_out_tl
                    \exp_not:N \group_begin:
                    \exp_not:V \l__zrefclever_reffont_in_tl
                    \exp_not:V \l__zrefclever_refpre_in_tl
                    \zref@extractdefault { \l__zrefclever_type_first_label_tl }
                      { \l__zrefclever_ref_property_tl } {}
                    \exp_not:V \l__zrefclever_refpos_in_tl
                    \exp_not:N \group_end:
                    \exp_not:V \l__zrefclever_refpos_out_tl
                    \exp_not:N \group_end:
                  }
              }
              { \exp_not:N \zref@default }
          }
      }
  }
```

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

`\__zrefclever_get_option_with_transl:nN`

```
% \Arg{option} \Arg{var to store result}
\cs_new_protected:Npn \__zrefclever_get_option_with_transl:nN #1#2
  {
    % First attempt options stored in \cs{l__zrefclever_ref_options_prop}.
    \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
      {
        % If not found, try the type specific options.
        \bool_lazy_all:nTF
          {
            { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
            {
              \prop_if_exist_p:c
                { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop }
            }
            {
              \prop_if_in_p:cn
                { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1}
            }
          }
          {
            \prop_get:cnN
              { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1} #2
          }
          {
            % If not found, try the type specific translations.
            \__zrefclever_if_translation:xxTF
              { \l__zrefclever_ref_language_tl }
              { zrefclever-type- \l__zrefclever_label_type_a_tl - #1 }
              {
                \__zrefclever_get_translation_for:nxx {#2}
```

```
                  { \l__zrefclever_ref_language_tl }
                  { zrefclever-type- \l__zrefclever_label_type_a_tl - #1 }
              }
              {
                % If not found, try general translations.  We are not
                % controlling for their existence, but we must make sure all
                % options being retrieved with
                % \cs{__zrefclever_get_option_with_transl:nN} have their values set for
                % 'English' and 'fallback'.
                \__zrefclever_get_translation_for:nxx {#2}
                  { \l__zrefclever_ref_language_tl }
                  { zrefclever-default- #1 }
              }
          }
      }
  }
```

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

`\__zrefclever_get_option_plain:nN`

```
\cs_new_protected:Npn \__zrefclever_get_option_plain:nN #1#2
  {
    % First attempt options stored in \cs{l__zrefclever_ref_options_prop}.
    \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
      {
        % If not found, try the type specific options.
        \bool_lazy_and:nnTF
          { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
          {
            \prop_if_exist_p:c
              { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop }
          }
          {
            \prop_get:cnNF
              { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1} #2
              { \tl_clear:N #2 }
          }
          { \tl_clear:N #2 }
      }
  }
```

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

`\__zrefclever_labels_in_sequence:nn` Sets `\l__zrefclever_next_maybe_range_bool` to true if label '1' comes in immediate sequence from label '2'. And sets both `\l__zrefclever_next_maybe_range_bool` and `\l__zrefclever_next_is_same_bool` if the labels are the "same".

```
\cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
  {
    \bool_if:NTF \l__zrefclever_page_ref_bool
      {
        \exp_args:Nxx \tl_if_eq:nnT
          { \zref@extractdefault {#1} { zc@pgfmt } { } }
          { \zref@extractdefault {#2} { zc@pgfmt } { } }
          {
```

```
            \int_compare:nNnTF
              { \zref@extractdefault {#1} { zc@pgval } {-2} + 1 }
                =
              { \zref@extractdefault {#2} { zc@pgval } {-1} }
              { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
              {
                \int_compare:nNnT
                  { \zref@extractdefault {#1} { zc@pgval } {-1} }
                    =
                  { \zref@extractdefault {#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
          { \zref@extractdefault {#1} { zc@counter } { } }
          { \zref@extractdefault {#2} { zc@counter } { } }
          {
            \exp_args:Nxx \tl_if_eq:nnT
              { \zref@extractdefault {#1} { zc@enclval } { } }
              { \zref@extractdefault {#2} { zc@enclval } { } }
              {
                \int_compare:nNnTF
                  { \zref@extractdefault {#1} { zc@cntval } {-2} + 1 }
                    =
                  { \zref@extractdefault {#2} { zc@cntval } {-1} }
                  { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
                  {
                    \int_compare:nNnT
                      { \zref@extractdefault {#1} { zc@cntval } {-1} }
                        =
                      { \zref@extractdefault {#2} { zc@cntval } {-1} }
                      {
                        \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`.)

10 Falback translations

All options retrieved with `\__zrefclever_get_option_with_transl:nN` must have their values set for 'fallback', since this is what will be retrieved if babel or polyglossia is loaded

and sets a language which zref-clever does not know. On the other hand type-specific options are not looked for in 'fallback'.

```
\__zrefclever_add_default_translation:nnn { fallback } { namesep   } {\nobreakspace}
\__zrefclever_add_default_translation:nnn { fallback } { pairsep   } {,~}
\__zrefclever_add_default_translation:nnn { fallback } { listsep   } {,~}
\__zrefclever_add_default_translation:nnn { fallback } { lastsep   } {,~}
\__zrefclever_add_default_translation:nnn { fallback } { tpairsep  } {,~}
\__zrefclever_add_default_translation:nnn { fallback } { tlistsep  } {,~}
\__zrefclever_add_default_translation:nnn { fallback } { tlastsep  } {,~}
\__zrefclever_add_default_translation:nnn { fallback } { notesep   } {~}
\__zrefclever_add_default_translation:nnn { fallback } { rangesep  } {\textendash}
\__zrefclever_add_default_translation:nnn { fallback } { refpre    } {}
\__zrefclever_add_default_translation:nnn { fallback } { refpos    } {}
\__zrefclever_add_default_translation:nnn { fallback } { refpre-in } {}
\__zrefclever_add_default_translation:nnn { fallback } { refpos-in } {}
```

⟨/package⟩

11 Localization

⟨*lang-english⟩

English

All options retrieved with `\__zrefclever_get_option_with_transl:nN` must have their values set for 'English', since this is what will be retrieved if no language package is loaded.

```
\ProvideDictionaryFor{English}{zref-clever}

\zcDicDefaultTransl{namesep}{\nobreakspace}
\zcDicDefaultTransl{pairsep}{~and\nobreakspace}
\zcDicDefaultTransl{listsep}{,~}
\zcDicDefaultTransl{lastsep}{~and\nobreakspace}
\zcDicDefaultTransl{tpairsep}{~and\nobreakspace}
\zcDicDefaultTransl{tlistsep}{,~}
\zcDicDefaultTransl{tlastsep}{,~and\nobreakspace}
\zcDicDefaultTransl{notesep}{~}
\zcDicDefaultTransl{rangesep}{~to\nobreakspace}
\zcDicDefaultTransl{refpre}{}
\zcDicDefaultTransl{refpos}{}
\zcDicDefaultTransl{refpre-in}{}
\zcDicDefaultTransl{refpos-in}{}

\zcDicTypeTransl{part}{name-sg}{part}
\zcDicTypeTransl{part}{name-pl}{parts}
\zcDicTypeTransl{part}{Name-sg}{Part}
\zcDicTypeTransl{part}{Name-pl}{Parts}
\zcDicTypeTransl{part}{name-ab-sg}{part}
\zcDicTypeTransl{part}{name-ab-pl}{parts}
\zcDicTypeTransl{part}{Name-ab-sg}{Part}
\zcDicTypeTransl{part}{Name-ab-pl}{Parts}

\zcDicTypeTransl{chapter}{name-sg}{chapter}
\zcDicTypeTransl{chapter}{name-pl}{chapters}
\zcDicTypeTransl{chapter}{Name-sg}{Chapter}
```

```
\zcDicTypeTransl{chapter}{Name-pl}{Chapters}
\zcDicTypeTransl{chapter}{name-ab-sg}{chapter}
\zcDicTypeTransl{chapter}{name-ab-pl}{chapters}
\zcDicTypeTransl{chapter}{Name-ab-sg}{Chapter}
\zcDicTypeTransl{chapter}{Name-ab-pl}{Chapters}

\zcDicTypeTransl{section}{name-sg}{section}
\zcDicTypeTransl{section}{name-pl}{sections}
\zcDicTypeTransl{section}{Name-sg}{Section}
\zcDicTypeTransl{section}{Name-pl}{Sections}
\zcDicTypeTransl{section}{name-ab-sg}{section}
\zcDicTypeTransl{section}{name-ab-pl}{sections}
\zcDicTypeTransl{section}{Name-ab-sg}{Section}
\zcDicTypeTransl{section}{Name-ab-pl}{Sections}

\zcDicTypeTransl{paragraph}{name-sg}{paragraph}
\zcDicTypeTransl{paragraph}{name-pl}{paragraphs}
\zcDicTypeTransl{paragraph}{Name-sg}{Paragraph}
\zcDicTypeTransl{paragraph}{Name-pl}{Paragraphs}
\zcDicTypeTransl{paragraph}{name-ab-sg}{par.}
\zcDicTypeTransl{paragraph}{name-ab-pl}{par.}
\zcDicTypeTransl{paragraph}{Name-ab-sg}{Par.}
\zcDicTypeTransl{paragraph}{Name-ab-pl}{Par.}

\zcDicTypeTransl{page}{name-sg}{page}
\zcDicTypeTransl{page}{name-pl}{pages}
\zcDicTypeTransl{page}{Name-sg}{Page}
\zcDicTypeTransl{page}{Name-pl}{Pages}
\zcDicTypeTransl{page}{name-ab-sg}{p.}
\zcDicTypeTransl{page}{name-ab-pl}{pp.}
\zcDicTypeTransl{page}{Name-ab-sg}{Page}
\zcDicTypeTransl{page}{Name-ab-pl}{Pages}

\zcDicTypeTransl{figure}{name-sg}{figure}
\zcDicTypeTransl{figure}{name-pl}{figures}
\zcDicTypeTransl{figure}{Name-sg}{Figure}
\zcDicTypeTransl{figure}{Name-pl}{Figures}
\zcDicTypeTransl{figure}{name-ab-sg}{fig.}
\zcDicTypeTransl{figure}{name-ab-pl}{figs.}
\zcDicTypeTransl{figure}{Name-ab-sg}{Fig.}
\zcDicTypeTransl{figure}{Name-ab-pl}{Figs.}

\zcDicTypeTransl{table}{name-sg}{table}
\zcDicTypeTransl{table}{name-pl}{tables}
\zcDicTypeTransl{table}{Name-sg}{Table}
\zcDicTypeTransl{table}{Name-pl}{Tables}
\zcDicTypeTransl{table}{name-ab-sg}{table}
\zcDicTypeTransl{table}{name-ab-pl}{tables}
\zcDicTypeTransl{table}{Name-ab-sg}{Table}
\zcDicTypeTransl{table}{Name-ab-pl}{Tables}

\zcDicTypeTransl{equation}{name-sg}{equation}
\zcDicTypeTransl{equation}{name-pl}{equations}
\zcDicTypeTransl{equation}{Name-sg}{Equation}
```

```
\zcDicTypeTransl{equation}{Name-pl}{Equations}
\zcDicTypeTransl{equation}{name-ab-sg}{eq.}
\zcDicTypeTransl{equation}{name-ab-pl}{eqs.}
\zcDicTypeTransl{equation}{Name-ab-sg}{Eq.}
\zcDicTypeTransl{equation}{Name-ab-pl}{Eqs.}
\zcDicTypeTransl{equation}{refpre-in}{(}
\zcDicTypeTransl{equation}{refpos-in}{)}

\zcDicTypeTransl{item}{name-sg}{item}
\zcDicTypeTransl{item}{name-pl}{items}
\zcDicTypeTransl{item}{Name-sg}{Item}
\zcDicTypeTransl{item}{Name-pl}{Items}
\zcDicTypeTransl{item}{name-ab-sg}{item}
\zcDicTypeTransl{item}{name-ab-pl}{items}
\zcDicTypeTransl{item}{Name-ab-sg}{Item}
\zcDicTypeTransl{item}{Name-ab-pl}{Items}
⟨/lang-english⟩
```

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.

N

P

R

S

T

U

Z