

The zref-clever package implementation*

Gustavo Barros[†]

2021-09-13

Contents

1	Initial setup	2
2	Dependencies	2
3	zref setup	2
4	Plumbing	6
4.1	Reference types	6
4.2	Messages	7
4.3	Translations aux	8
4.4	Options	9
5	Type format	19
5.1	\zcRefTypeSetup	19
5.2	\zcDeclareTranslations	22
6	\zcref	26
7	\zcpageref	27
8	Sorting	27
9	Typesetting	34
10	Fallback translations	54
11	Localization	55
	Index	57

*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.

```
1 <*package>
   Identify the internal prefix (LaTeX3 DocStrip convention).
2 <@@=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 (`ltxcmdhooks`), which is bound to be useful for our purposes, and was released with the 2021-06-01 kernel.

```
3 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
4 \IfFormatAtLeastTF{2021-06-01}
5 {}
6 {%
7   \PackageError{zref-clever}{LaTeX kernel too old}
8   {%
9     'zref-clever' requires a LaTeX kernel newer than 2021-06-01.%
10    \MessageBreak Loading will abort!%
11   }%
12   \endinput
13 }%
   Identify the package.
14 \ProvidesExplPackage {zref-clever} {2021-09-13} {0.1.0-alpha}
15 {Do-what-I-mean cross-references based on zref}
```

2 Dependencies

```
16 \RequirePackage { zref-base }
17 \RequirePackage { zref-user }
18 \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.

```

19 \zref@newprop { zc@counter } { \@currentcounter }
20 \zref@addprop \ZREF@mainlist { zc@counter }
21 \zref@newprop { zc@type }
22 { \exp_args:Nne \prop_item:Nn \l__zrefclever_counter_type_prop { \@currentcounter } }
23 \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.

```

24 \zref@newprop { zc@thecnt } { \use:c { the \@currentcounter } }
25 \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`.

```

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

```

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

```

28 \zref@newprop* { zc@pgval } [0] { \int_use:c { c@page } }
29 \zref@addprop \ZREF@mainlist { zc@pgval }
30 \int_new:N \g__zrefclever_abspage_int
31 \AddToHook { shipout/before } { \int_gincr:N \g__zrefclever_abspage_int }
32 \zref@newprop* { zc@abspg } [0] { \int_use:N \g__zrefclever_abspage_int }
33 \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_resettters_seq`, and for each counter retrieves its “enclosing counters” recursively. There is one relevant caveat to this procedure: `\l__zrefclever_counter_resettters_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 $\{\langle counter \rangle\}$ 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.

```

34 \cs_new:Npn \__zrefclever_get_enclosing_counters:n #1
35 {
36   \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
37   {
38     { \__zrefclever_counter_reset_by:n {#1} }
39     \__zrefclever_get_enclosing_counters:e
40     { \__zrefclever_counter_reset_by:n {#1} }
41   }
42 }
43 \cs_new:Npn \__zrefclever_get_enclosing_counters_value:n #1
44 {
45   \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
46   {
47     { \int_use:N \cs:w c@ \__zrefclever_counter_reset_by:n {#1} \cs_end: }
48     \__zrefclever_get_enclosing_counters_value:e
49     { \__zrefclever_counter_reset_by:n {#1} }
50   }
51 }

```

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’).

```

52 \cs_generate_variant:Nn \__zrefclever_get_enclosing_counters:n { V , e }
53 \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 $\{\langle counter \rangle\}$.

```

54 \cs_new:Npn \__zrefclever_counter_reset_by:n #1
55 {
56   \bool_if:nTF
57   { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} }
58   { \prop_item:Nn \l__zrefclever_counter_resetby_prop {#1} }
59   {
60     \seq_map_tokens:Nn \l__zrefclever_counter_resettters_seq
61     { \__zrefclever_counter_reset_by_aux:nn {#1} }
62   }
63 }
64 \cs_new:Npn \__zrefclever_counter_reset_by_aux:nn #1#2

```

```

65 {
66   \cs_if_exist:cT { c@ #2 }
67   {
68     \tl_if_empty:cF { cl@ #2 }
69     {
70       \tl_map_tokens:cn { cl@ #2 }
71       { \__zrefclever_counter_reset_by_auxi:nnn {#2} {#1} }
72     }
73   }
74 }
75 \cs_new:Npn \__zrefclever_counter_reset_by_auxi:nnn #1#2#3
76 {
77   \str_if_eq:nnT {#2} {#3}
78   { \tl_map_break:n { \seq_map_break:n {#1} } }
79 }

```

(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.

```

80 \zref@newprop { zc@enclcnt }
81 { \__zrefclever_get_enclosing_counters:V \@currentcounter }
82 \zref@newprop { zc@enclval }
83 { \__zrefclever_get_enclosing_counters_value:V \@currentcounter }
84 \zref@addprop \ZREF@mainlist { zc@enclcnt }
85 \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:`

```

86 \tl_new:N \g__zrefclever_page_format_tl
87 \cs_new_protected:Npx \__zrefclever_page_format_aux: { \int_eval:n { 1 } }
88 \AddToHook { shipout / before }
89 {
90   \group_begin:

```

```

91 \cs_set_eq:NN \c@page \__zrefclever_page_format_aux:
92 \exp_args:NNx \tl_gset:Nn \g__zrefclever_page_format_tl { \thepage }
93 \group_end:
94 }
95 \zref@newprop* { zc@pgfmt } { \g__zrefclever_page_format_tl }
96 \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:~

```

```

100     Don't know what to do with #1.
101   }
102   \msg_new:nnn { zref-clever } { option-not-type-specific }
103   {
104     Option~'#1'~is-not-type-specific~\msg_line_context:..~
105     Set-it-in~'\exp_not:N \zcDeclareTranslations'~before-first~'type'~switch-
106     or-as-package-option.
107   }
108   \msg_new:nnn { zref-clever } { option-only-type-specific }
109   {
110     No~type~specified-for~option~'#1'~\msg_line_context:..~
111     Set-it-after~'type'~switch-or-in~'\exp_not:N \zcRefTypeSetup'.
112   }
113   \msg_new:nnn { zref-clever } { countertype-requires-value }
114   { The~'countertype'~key~'#1'~requires-a~value. }
115   \msg_new:nnn { zref-clever } { counterresetby-requires-value }
116   { The~'counterresetby'~key~'#1'~requires-a~value. }
117   \msg_new:nnn { zref-clever } { missing-zref-titleref }
118   {
119     Option~'ref=title'~requested~\msg_line_context:..~
120     But~package~'zref-titleref'~is-not-loaded,~falling-back-to-default~'ref'.
121   }
122   \msg_new:nnn { zref-clever } { hyperref-preamble-only }
123   {
124     Option~'hyperref'~only~available~in~the~preamble. \iow_newline:
125     Use-the-starred-version-of~'\noexpand\zcheck'~instead.
126   }
127   \msg_new:nnn { zref-clever } { missing-hyperref }
128   { Missing~'hyperref'~package. \iow_newline: Setting~'hyperref=false'. }
129   \msg_new:nnn { zref-clever } { counters-not-nested }
130   { Counters~not~nested~for~labels~'#1'~and~'#2'~\msg_line_context:.. }
131   \msg_new:nnn { zref-clever } { missing-type }
132   { Reference~type~undefined~for~label~'#1'~\msg_line_context:.. }
133   \msg_new:nnn { zref-clever } { missing-name }
134   { Name~undefined~for~type~'#1'~\msg_line_context:.. }
135   \msg_new:nnn { zref-clever } { single-element-range }
136   { 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.

```

137   \prg_new_conditional:Npnn \__zrefclever_if_translation:nn #1#2 { p , TF }
138   {
139     \IfTranslation {#1} {#2}
140     { \prg_return_true: }
141     { \prg_return_false: }
142   }
143   \prg_generate_conditional_variant:Nnn \__zrefclever_if_translation:nn { xx } { p , TF }
144   \cs_new_protected:Npn \__zrefclever_get_translation_for:nnn #1#2#3
145   { \SaveTranslationFor{#1}{#2}{#3} }
146   \cs_generate_variant:Nn \__zrefclever_get_translation_for:nnn { nxx }
147   \cs_new_protected:Npn \__zrefclever_declare_translation:nnn #1#2#3
148   { \declaretranslation {#1} {#2} {#3} }

```



```

149 \cs_generate_variant:Nn \__zrefclever_declare_translation:nnn { xxn , xxx }
150
151 % <lang><key><transl>
152 \cs_new_protected:Npn \__zrefclever_add_default_translation:nnn #1#2#3
153 { \addtranslation {#1} { zrefclever-default- #2 } {#3} }
154
155 % <lang><type><key><transl>
156 \cs_new_protected:Npn \__zrefclever_add_type_translation:nnnn #1#2#3#4
157 { \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.

```

158 % <key><transl>
159 \NewDocumentCommand \zcDicDefaultTransl { m m }
160 { \ProvideDictTranslation { zrefclever-default- #1 } {#2} }
161 % <type><key><transl>
162 \NewDocumentCommand \zcDicTypeTransl { m m m }
163 { \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”.

```

164 \prop_new:N \l__zrefclever_counter_type_prop

(End definition for \l__zrefclever_counter_type_prop.)

165 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
166 {
167   \tl_if_empty:nTF {#3}
168   { \prop_remove:Nn #1 {#2} }
169   { \prop_put:Nnn #1 {#2} {#3} }
170 }
171 \keys_define:nn { zref-clever }
172 {
173   countertype .code:n =
174   {
175     \keyval_parse:nnn
176     { \msg_warning:nnn { zref-clever } { countertype-requires-value } }
177     { \__zrefclever_prop_put_non_empty:Nnn \l__zrefclever_counter_type_prop }
178     {#1}
179   } ,
180   countertype .value_required:n = true ,
181   countertype .initial:n =
182   {
183     part          = part ,
184     chapter       = chapter ,
185     section       = section ,
186     subsection    = section ,
187     subsubsection = section ,
188     paragraph     = paragraph ,
189     subparagraph  = paragraph ,
190     figure        = figure ,
191     table         = table ,

```

```

192         equation      = equation ,
193         enumi          = item ,
194         enumii         = item ,
195         enumiii        = item ,
196         enumiv         = item ,
197     } ,
198 }

```

counterresetters option

`\l_zrefclever_counter_resetters_seq` Stores the list of counters which are potential “enclosing counters” for other counters.

```

199 \seq_new:N \l__zrefclever_counter_resetters_seq

```

(End definition for `\l__zrefclever_counter_resetters_seq`.)

```

200 \keys_define:nn { zref-clever }
201 {
202     counterresetters .code:n =
203     {
204         \clist_map_inline:nn {#1}
205         {
206             \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
207             { \seq_put_right:Nn \l__zrefclever_counter_resetters_seq {##1} }
208         }
209     } ,
210     counterresetters .initial:n =
211     {
212         part ,
213         chapter ,
214         section ,
215         subsection ,
216         subsubsection ,
217         paragraph ,
218         subparagraph ,
219     },
220     typesort .value_required:n = true ,
221 }

```

counterresetby option

`\l_zrefclever_counter_resetby_prop` Variable storing a mapping from “counter” to the counter which resets it.

```

222 \prop_new:N \l__zrefclever_counter_resetby_prop

```

(End definition for `\l__zrefclever_counter_resetby_prop`.)

```

223 \keys_define:nn { zref-clever }
224 {
225     counterresetby .code:n =
226     {
227         \keyval_parse:nnn
228         { \msg_warning:nnn { zref-clever } { counterresetby-requires-value } }
229         { \__zrefclever_prop_put_non_empty:Nnn \l__zrefclever_counter_resetby_prop }
230         {#1}
231     } ,
232     counterresetby .value_required:n = true ,

```

```

233     counterresetby .initial:n =
234     {

```

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.

```

235         enumii = enumi ,
236         enumiii = enumii ,
237         enumiv = enumiii ,
238     } ,
239 }

```

ref option

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

```

240 \tl_new:N \l__zrefclever_ref_property_tl
241 \bool_new:N \l__zrefclever_page_ref_bool
242 \keys_define:nn { zref-clever }
243 {
244     ref .choice: ,
245     ref / zc@thecnt .code:n =
246     {
247         \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
248         \bool_set_false:N \l__zrefclever_page_ref_bool
249     } ,
250     ref / page .code:n =
251     {
252         \tl_set:Nn \l__zrefclever_ref_property_tl { page }
253         \bool_set_true:N \l__zrefclever_page_ref_bool
254     } ,
255     ref / title .code:n =
256     {
257         \AddToHook { begindocument }
258         {
259             \@ifpackageloaded { zref-titleref }
260             {
261                 \tl_set:Nn \l__zrefclever_ref_property_tl { title }
262                 \bool_set_false:N \l__zrefclever_page_ref_bool
263             }
264             {
265                 \msg_warning:nn { zref-clever } { missing-zref-titleref }
266                 \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
267                 \bool_set_false:N \l__zrefclever_page_ref_bool
268             }
269         }
270     } ,
271     ref .initial:n = zc@thecnt ,
272     ref .value_required:n = true ,
273     page .meta:n = { ref = page } ,
274     page .value_forbidden:n = true ,
275 }
276
277 \AddToHook { begindocument }

```

```

278 {
279   \@ifpackageloaded { zref-titleref }
280   {
281     \keys_define:nn { zref-clever }
282     {
283       ref / title .code:n =
284       {
285         \tl_set:Nn \l__zrefclever_ref_property_tl { title }
286         \bool_set_false:N \l__zrefclever_page_ref_bool
287       }
288     }
289   }
290   {
291     \keys_define:nn { zref-clever }
292     {
293       ref / title .code:n =
294       {
295         \msg_warning:nn { zref-clever } { missing-zref-titleref }
296         \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
297         \bool_set_false:N \l__zrefclever_page_ref_bool
298       }
299     }
300   }
301 }

```

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

```

302 \bool_new:N \l__zrefclever_typeset_ref_bool
303 \bool_new:N \l__zrefclever_typeset_name_bool
304 \keys_define:nn { zref-clever }
305 {
306   typeset .choice: ,
307   typeset / both .code:n =
308   {
309     \bool_set_true:N \l__zrefclever_typeset_ref_bool
310     \bool_set_true:N \l__zrefclever_typeset_name_bool
311   } ,
312   typeset / ref .code:n =
313   {
314     \bool_set_true:N \l__zrefclever_typeset_ref_bool
315     \bool_set_false:N \l__zrefclever_typeset_name_bool
316   } ,
317   typeset / name .code:n =

```

```

318     {
319         \bool_set_false:N \l__zrefclever_typeset_ref_bool
320         \bool_set_true:N \l__zrefclever_typeset_name_bool
321     } ,
322     typeset .initial:n = both ,
323     typeset .value_required:n = true ,
324
325     noname .meta:n = { typeset = ref },
326     noname .value_forbidden:n = true ,
327 }

```

sort option

User option, sort labels ranges or not

```

328 \bool_new:N \l__zrefclever_typeset_sort_bool
329 \keys_define:nn { zref-clever }
330 {
331     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
332     sort .initial:n = true ,
333     sort .default:n = true ,
334     nosort .meta:n = { sort = false },
335     nosort .value_forbidden:n = true ,
336 }

```

typesort option

```

337 \seq_new:N \l__zrefclever_typesort_seq
338 \keys_define:nn { zref-clever }
339 {
340     typesort .code:n =
341     {
342         \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
343         % Reverse the sequence, since the sort priorities are computed in the
344         % negative range, so that we can implicitly rely on '0' being the
345         % 'last value'.
346         \seq_reverse:N \l__zrefclever_typesort_seq
347     } ,
348     typesort .initial:n =
349     { part , chapter , section , paragraph },
350     typesort .value_required:n = true ,
351     notypesort .code:n =
352     { \seq_clear:N \l__zrefclever_typesort_seq } ,
353     notypesort .value_forbidden:n = true ,
354 }

```

comp option

User option, compress ranges or not

```

355 \bool_new:N \l__zrefclever_typeset_compress_bool
356 \keys_define:nn { zref-clever }
357 {
358     comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
359     comp .initial:n = true ,
360     comp .default:n = true ,
361     nocomp .meta:n = { comp = false },

```

```

362     nocomp .value_forbidden:n = true ,
363 }

```

range option

```

364 \bool_new:N \l__zrefclever_typeset_range_bool
365 \keys_define:nn { zref-clever }
366 {
367     range .bool_set:N = \l__zrefclever_typeset_range_bool ,
368     range .initial:n = false ,
369     range .default:n = true ,
370 }

```

hyperref option

```

\l__zrefclever_use_hyperref_bool
\l__zrefclever_warn_hyperref_bool

```

```

371 \bool_new:N \l__zrefclever_use_hyperref_bool
372 \bool_new:N \l__zrefclever_warn_hyperref_bool
373 \keys_define:nn { zref-clever }
374 {
375     hyperref .choice: ,
376     hyperref / auto .code:n =
377     {
378         \bool_set_true:N \l__zrefclever_use_hyperref_bool
379         \bool_set_false:N \l__zrefclever_warn_hyperref_bool
380     } ,
381     hyperref / true .code:n =
382     {
383         \bool_set_true:N \l__zrefclever_use_hyperref_bool
384         \bool_set_true:N \l__zrefclever_warn_hyperref_bool
385     } ,
386     hyperref / false .code:n =
387     {
388         \bool_set_false:N \l__zrefclever_use_hyperref_bool
389         \bool_set_false:N \l__zrefclever_warn_hyperref_bool
390     } ,
391     hyperref .initial:n = auto ,
392     hyperref .default:n = auto
393 }

```

(End definition for \l__zrefclever_use_hyperref_bool and \l__zrefclever_warn_hyperref_bool.)

```

394 \AddToHook { begindocument }
395 {
396     \@ifpackageloaded { hyperref }
397     {
398         \bool_if:NT \l__zrefclever_use_hyperref_bool
399         { \RequirePackage { zref-hyperref } }
400     }
401     {
402         \bool_if:NT \l__zrefclever_warn_hyperref_bool
403         { \msg_warning:nn { zref-clever } { missing-hyperref } }
404         \bool_set_false:N \l__zrefclever_use_hyperref_bool
405     }
406     \keys_define:nn { zref-clever }
407     {

```

```

408         hyperref .code:n =
409         { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }
410     }
411 }

```

nameinlink option

`\l__zrefclever_nameinlink_tl`

```

412 \str_new:N \l__zrefclever_nameinlink_str
413 \keys_define:nn { zref-clever }
414 {
415     nameinlink .choice: ,
416     nameinlink / true .code:n =
417     { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
418     nameinlink / false .code:n =
419     { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
420     nameinlink / single .code:n =
421     { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
422     nameinlink / tsingle .code:n =
423     { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
424     nameinlink .initial:n = tsingle ,
425     nameinlink .default:n = true ,
426 }

```

(End definition for \l__zrefclever_nameinlink_tl.)

cap capfirst options

```

427 \bool_new:N \l__zrefclever_capitalize_bool
428 \bool_new:N \l__zrefclever_capitalize_first_bool
429 \keys_define:nn { zref-clever }
430 {
431     cap .bool_set:N = \l__zrefclever_capitalize_bool ,
432     cap .initial:n = false ,
433     cap .default:n = true ,
434     nocap .meta:n = { cap = false },
435     nocap .value_forbidden:n = true ,
436
437     capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,
438     capfirst .initial:n = false ,
439     capfirst .default:n = true ,
440
441     C .meta:n =
442     { capfirst = true , noabbrevfirst = true },
443     C .value_forbidden:n = true ,
444 }

```

abbrev noabbrevfirst option

```

445 \bool_new:N \l__zrefclever_abbrev_bool
446 \bool_new:N \l__zrefclever_noabbrevfirst_bool
447 \keys_define:nn { zref-clever }
448 {
449     abbrev .bool_set:N = \l__zrefclever_abbrev_bool ,
450     abbrev .initial:n = false ,

```

```

451     abbrev .default:n = true ,
452     noabbrev .meta:n = { abbrev = false },
453     noabbrev .value_forbidden:n = true ,
454
455     noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
456     noabbrevfirst .initial:n = false ,
457     noabbrevfirst .default:n = true ,
458 }

```

lang option

```

459 \tl_new:N \l__zrefclever_ref_language_tl
460 \NewHook { zref-clever / reflanguage }
461 \keys_define:nn { zref-clever }
462 {
463     lang .code:n =
464     {
465         \AddToHook { zref-clever / reflanguage }
466         {
467             \str_case:nnF {#1}
468             {
469                 { main }
470                 {
471                     \tl_set_eq:NN
472                     \l__zrefclever_ref_language_tl \l__zrefclever_main_language_tl
473                 }
474
475                 { current }
476                 {
477                     \tl_set_eq:NN
478                     \l__zrefclever_ref_language_tl \l__zrefclever_current_language_tl
479                 }
480             }
481             { \tl_set:Nn \l__zrefclever_ref_language_tl {#1} }
482         }
483     } ,
484     lang .initial:n = main ,
485     lang .value_required:n = true ,
486 }

```

\AtEndOfPackage so that it comes after \ProcessKeysOptions.

```

487 \AtEndOfPackage
488 {
489     \AddToHook { zref-clever / reflanguage }
490     {
491         \keys_define:nn { zref-clever }
492         {
493             lang .code:n =
494             {
495                 \str_case:nnF {#1}
496                 {
497                     { main }
498                     {
499                         \tl_set_eq:NN
500                         \l__zrefclever_ref_language_tl \l__zrefclever_main_language_tl

```



```

501         }
502
503         { current }
504         {
505             \tl_set_eq:NN
506             \l__zrefclever_ref_language_tl \l__zrefclever_current_language_tl
507         }
508     }
509     { \tl_set:Nn \l__zrefclever_ref_language_tl {#1} }
510 },
511 lang .initial:n = main ,
512 lang .value_required:n = true ,
513 }
514 }
515 }

```

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

```

516 \AddToHook { begindocument / before }
517 {
518     % An internal alias for \pkg{translations}'s internal macro
519     % \cs{@trnslt@current@language}.
520     \tl_set_eq:NN \l__zrefclever_current_language_tl \@trnslt@current@language
521     % Getting main languages and, for each babel/polyglossia loaded language,
522     % load corresponding zref-clever dictionary.
523     \ifpackageloaded{babel}
524     {
525         \tl_set_eq:NN \l__zrefclever_main_language_tl \bbl@main@language
526         \clist_map_inline:Nn \bbl@loaded
527         {
528             % Funny enough, \pkg{translations} also loads its basic
529             % dictionaries for all languages loaded by babel or polyglossia.
530             % First, there is no way to disable this, even if we don't need
531             % them at all here. Second, \pkg{translations} sends messages of
532             % its own missing dictionaries to 'info' and everyone else's to
533             % 'warning'\dots{} So we have to control ourselves for missing
534             % dictionaries and load them only if available.
535             \exp_args:Nx \file_if_exist:nT
536             { zref-clever- \@trnslt@language {#1} .trsl }
537             { \LoadDictionaryFor {#1} { zref-clever } }
538         }
539     }
540     {
541         \ifpackageloaded{polyglossia}
542         {
543             \tl_set_eq:NN \l__zrefclever_main_language_tl \xpg@main@language
544             \clist_map_inline:Nn \xpg@loaded
545             {
546                 \exp_args:Nx \file_if_exist:nT
547                 { zref-clever- \@trnslt@language {#1} .trsl }
548                 { \LoadDictionaryFor {#1} { zref-clever } }
549             }
550         }
551         {
552             \tl_new:N \l__zrefclever_main_language_tl

```

```

553         \tl_set:Nn \l__zrefclever_main_language_tl { english }
554         \LoadDictionaryFor { english } { zref-clever }
555     }
556 }
557 % *Then* we execute the package options stored in the 'reflanguage' hook.
558 \UseHook { zref-clever / reflanguage }
559 }

```

note option

```

560 \tl_new:N \l__zrefclever_zcref_note_tl
561 \keys_define:nn { zref-clever }
562 {
563     note .tl_set:N = \l__zrefclever_zcref_note_tl ,
564     note .value_required:n = true ,
565 }

```

Reference options

```

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

```

Only not necessarily type-specific options are pertinent here.

```

569 \prop_new:N \l__zrefclever_ref_options_prop
570 \clist_map_inline:nn
571 {
572     % Not type-specific options.
573     tpairsep ,
574     tlistsep ,
575     tlastsep ,
576     notesep ,
577     % Possibly type-specific options.
578     namefont ,
579     namesep ,
580     pairsep ,
581     listsep ,
582     lastsep ,
583     rangesep ,
584     reffont ,
585     refpre ,
586     refpos ,
587     reffont-in ,
588     refpre-in ,
589     refpos-in ,
590 }
591 {
592     \keys_define:nn { zref-clever }
593     {
594         #1 .default:V = \c_novalue_tl ,
595         #1 .code:n =
596         {
597             \tl_if_novalue:nTF {##1}
598             { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
599             { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
600         } ,
601     }

```

```
602 }
```

Package options

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

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

`\zcsetup` Provide `\zcsetup`.

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

(End definition for `\zcsetup`.)

5 Type format

5.1 `\zcRefTypeSetup`

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

```
\l__zrefclever_setup_language_tl
607 \tl_new:N \l__zrefclever_setup_type_tl
608 \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`.

```
609 \NewDocumentCommand \zcRefTypeSetup { m m }
610 {
611   \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
612   { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
613   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
614   \keys_set:nn { zref-clever / typesetup } {#2}
615 }
```

(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.

```
616 \keys_define:nn { zref-clever / typesetup }
617 {
618   name .default:V = \c_novalue_tl ,
619   name .code:n =
```

```

620 {
621     \tl_if_novalue:nTF {#1}
622     {
623         \clist_map_inline:nn
624         {
625             name-sg ,
626             name-pl ,
627             Name-sg ,
628             Name-pl ,
629             name-ab-sg ,
630             name-ab-pl ,
631             Name-ab-sg ,
632             Name-ab-pl ,
633         }
634         {
635             \prop_remove:cn
636             { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
637             {##1}
638         }
639     }
640     {
641         \int_case:nnF { \clist_count:n {#1} }
642         {
643             { 2 }
644             {
645                 \clist_map_inline:nn
646                 {
647                     name-sg ,
648                     Name-sg ,
649                     name-ab-sg ,
650                     Name-ab-sg ,
651                 }
652                 {
653                     \prop_put:cnx
654                     { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
655                     {##1} { \clist_item:nn {#1} { 1 } }
656                 }
657                 \clist_map_inline:nn
658                 {
659                     name-pl ,
660                     Name-pl ,
661                     name-ab-pl ,
662                     Name-ab-pl ,
663                 }
664                 {
665                     \prop_put:cnx
666                     { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
667                     {##1} { \clist_item:nn {#1} { 2 } }
668                 }
669             }
670         }
671         { 4 }
672         {
673             % Make the first pair the capitalized ones, so as to make

```

```

674 % them "feel" the default for the single pair case.
675 \prop_put:cnx
676 { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
677 { Name-sg } { \clist_item:nn {#1} { 1 } }
678 \prop_put:cnx
679 { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
680 { Name-ab-sg } { \clist_item:nn {#1} { 1 } }
681 \prop_put:cnx
682 { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
683 { Name-pl } { \clist_item:nn {#1} { 2 } }
684 \prop_put:cnx
685 { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
686 { Name-ab-pl } { \clist_item:nn {#1} { 2 } }
687 \prop_put:cnx
688 { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
689 { name-sg } { \clist_item:nn {#1} { 3 } }
690 \prop_put:cnx
691 { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
692 { name-ab-sg } { \clist_item:nn {#1} { 3 } }
693 \prop_put:cnx
694 { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
695 { name-pl } { \clist_item:nn {#1} { 4 } }
696 \prop_put:cnx
697 { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
698 { name-ab-pl } { \clist_item:nn {#1} { 4 } }
699 }
700 }
701 {
702 \msg_warning:nxx { zref-clever } { type-name-length }
703 { \clist_count:n {#1} }
704 }
705 }
706 } ,
707 }

```

Not type-specific options.

```

708 \clist_map_inline:nn
709 {
710   tpairsep ,
711   tlistsep ,
712   tlastsep ,
713   notesep ,
714 }
715 {
716   \keys_define:nn { zref-clever / typesetup }
717   {
718     #1 .code:n =
719     {
720       \msg_warning:nnn { zref-clever } { option-not-type-specific } {#1}
721     } ,
722   }
723 }

```

Possibly or necessarily type-specific options.

```

724 \clist_map_inline:nn

```

```

725 {
726   % Possibly type-specific options.
727   namefont ,
728   namesep ,
729   pairsep ,
730   listsep ,
731   lastsep ,
732   rangesep ,
733   reffont ,
734   refpre ,
735   refpos ,
736   reffont-in ,
737   refpre-in ,
738   refpos-in ,
739   % Necessarily type-specific options.
740   name-sg ,
741   name-pl ,
742   Name-sg ,
743   Name-pl ,
744   name-ab-sg ,
745   name-ab-pl ,
746   Name-ab-sg ,
747   Name-ab-pl ,
748 }
749 {
750   \keys_define:nm { zref-clever / typesetup }
751   {
752     #1 .default:V = \c_novalue_tl ,
753     #1 .code:n =
754     {
755       \tl_if_novalue:nTF {##1}
756       {
757         \prop_remove:cn
758         { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
759         {#1}
760       }
761       {
762         \prop_put:cnn
763         { l__zrefclever_type_ \l__zrefclever_setup_type_tl _options_prop }
764         {#1} {##1}
765       }
766     } ,
767   }
768 }

```

5.2 \zcDeclareTranslations

\zcDeclareTranslations Provide \zcDeclareTranslations.

```

769 \NewDocumentCommand \zcDeclareTranslations { m m }
770 {
771   \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
772   \tl_clear:N \l__zrefclever_setup_type_tl
773   \keys_set:nm { zref-clever / translations } {#2}
774 }

```

(End definition for \zcDeclareTranslations.)

```

775 \keys_define:nn { zref-clever / translations }
776 {
777   type .code:n =
778   {
779     \tl_if_empty:nTF {#1}
780     { \tl_clear:N \l__zrefclever_setup_type_tl }
781     {
782       \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
783       { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
784       \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
785     }
786   } ,
787 }

```

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

```

788 \keys_define:nn { zref-clever / translations }
789 {
790   name .value_required:n = true ,
791   name .code:n =
792   {
793     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
794     {
795       \msg_warning:nnn { zref-clever }
796       { option-only-type-specific } {#1}
797     }
798     {
799       \int_case:nnF { \clist_count:n {#1} }
800       {
801         { 2 }
802         {
803           \clist_map_inline:nn
804           {
805             name-sg ,
806             Name-sg ,
807             name-ab-sg ,
808             Name-ab-sg ,
809           }
810           {
811             \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_
812             { zrefclever-type- \l__zrefclever_setup_type_tl - ##1 }
813             { \clist_item:nn {#1} { 1 } } }
814           }
815           \clist_map_inline:nn
816           {
817             name-pl ,
818             Name-pl ,
819             name-ab-pl ,
820             Name-ab-pl ,
821           }
822           {
823             \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_
824             { zrefclever-type- \l__zrefclever_setup_type_tl - ##1 }

```

```

825         { \clist_item:nn {#1} { 2 } }
826     }
827 }
828
829 { 4 }
830 {
831     % Make the first pair the capitalized ones, so as to make
832     % them "feel" the default for the single pair case.
833     \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_tl }
834     { zrefclever-type- \l__zrefclever_setup_type_tl -Name-sg }
835     { \clist_item:nn {#1} { 1 } }
836     \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_tl }
837     { zrefclever-type- \l__zrefclever_setup_type_tl -Name-ab-sg }
838     { \clist_item:nn {#1} { 1 } }
839     \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_tl }
840     { zrefclever-type- \l__zrefclever_setup_type_tl -Name-pl }
841     { \clist_item:nn {#1} { 2 } }
842     \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_tl }
843     { zrefclever-type- \l__zrefclever_setup_type_tl -Name-ab-pl }
844     { \clist_item:nn {#1} { 2 } }
845     \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_tl }
846     { zrefclever-type- \l__zrefclever_setup_type_tl -name-sg }
847     { \clist_item:nn {#1} { 3 } }
848     \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_tl }
849     { zrefclever-type- \l__zrefclever_setup_type_tl -name-ab-sg }
850     { \clist_item:nn {#1} { 3 } }
851     \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_tl }
852     { zrefclever-type- \l__zrefclever_setup_type_tl -name-pl }
853     { \clist_item:nn {#1} { 4 } }
854     \__zrefclever_declare_translation:xxx { \l__zrefclever_setup_language_tl }
855     { zrefclever-type- \l__zrefclever_setup_type_tl -name-ab-pl }
856     { \clist_item:nn {#1} { 4 } }
857 }
858 }
859 {
860     \msg_warning:nxx { zref-clever } { type-name-length }
861     { \clist_count:n {#1} }
862 }
863 }
864 } ,
865 }

```

Not type-specific options.

```

866 \clist_map_inline:nn
867 {
868     tpairsep ,
869     tlistsep ,
870     tlastsep ,
871     notesep ,
872 }
873 {
874     \keys_define:nn { zref-clever / translations }
875     {
876         #1 .value_required:n = true ,
877         #1 .code:n =

```



```

878     {
879       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
880       {
881         \__zrefclever_declare_translation:xxn { \l__zrefclever_setup_language_tl }
882         { zrefclever-default- #1 } {##1}
883       }
884       {
885         \msg_warning:nnn { zref-clever }
886         { option-not-type-specific } {#1}
887       }
888     } ,
889   }
890 }

```

Possibly type-specific options.

```

891 \clist_map_inline:nn
892 {
893   namesep ,
894   pairsep ,
895   listsep ,
896   lastsep ,
897   rangesep ,
898   refpre ,
899   refpos ,
900   refpre-in ,
901   refpos-in ,
902 }
903 {
904   \keys_define:nn { zref-clever / translations }
905   {
906     #1 .value_required:n = true ,
907     #1 .code:n =
908     {
909       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
910       {
911         \__zrefclever_declare_translation:xxn { \l__zrefclever_setup_language_tl }
912         { zrefclever-default- #1 } {##1}
913       }
914       {
915         \__zrefclever_declare_translation:xxn { \l__zrefclever_setup_language_tl }
916         { zrefclever-type- \l__zrefclever_setup_type_tl - #1 } {##1}
917       }
918     } ,
919   }
920 }

```

Necessarily type-specific options.

```

921 \clist_map_inline:nn
922 {
923   name-sg ,
924   name-pl ,
925   Name-sg ,
926   Name-pl ,
927   name-ab-sg ,
928   name-ab-pl ,

```

```

929     Name-ab-sg ,
930     Name-ab-pl ,
931   }
932   {
933     \keys_define:nn { zref-clever / translations }
934     {
935       #1 .value_required:n = true ,
936       #1 .code:n =
937       {
938         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
939         {
940           \msg_warning:nnn { zref-clever }
941             { option-only-type-specific } {#1}
942         }
943         {
944           \__zrefclever_declare_translation:xxn { \l__zrefclever_setup_language_tl }
945             { zrefclever-type- \l__zrefclever_setup_type_tl - #1 } {##1}
946         }
947       } ,
948     }
949   }

```

6 \zcref

```

\zcref      \zcref{*}[\<options>]{\<labels>}
950 \NewDocumentCommand \zcref { s O { } m }
951 { \zref@wrapper@babel \__zrefclever_zcref:nnn {#3} {#1} {#2} }

```

(End definition for \zcref.)

```

\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool
952 \seq_new:N \l__zrefclever_zcref_labels_seq
953 \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>}
954 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
955 {
956   \group_begin:
957   \keys_set:nn { zref-clever } {#3}
958   \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
959   \bool_set:Nn \l__zrefclever_link_star_bool {#2}
960   \bool_lazy_or:nnT
961     { \l__zrefclever_typeset_sort_bool }
962     { \l__zrefclever_typeset_range_bool }
963     { \__zrefclever_sort_labels: }
964   \__zrefclever_typeset_refs:

```

```

965 % Typeset \texttt{note}.
966 \l__zrefclever_noteseq_tl
967 \l__zrefclever_zcref_note_tl
968 \group_end:
969 }

```

(End definition for __zrefclever_zcref:nnnn.)

7 \zcpageref

```

\zcpageref \zcpageref*[\langle options \rangle]{\langle labels \rangle}
970 \NewDocumentCommand \zcpageref { s O { } m }
971 {
972   \IfBooleanTF {#1}
973     { \zcref*[#2, ref = page] {#3} }
974     { \zcref [ #2, ref = page] {#3} }
975 }

```

(End definition for \zcpageref.)

8 Sorting

```

976 \int_new:N \l__zrefclever_sort_prior_a_int
977 \int_new:N \l__zrefclever_sort_prior_b_int

```

\l__zrefclever_label_a_tl \l__zrefclever_label_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.

```

\l__zrefclever_label_type_a_tl 978 \tl_new:N \l__zrefclever_label_a_tl
\l__zrefclever_label_type_b_tl 979 \tl_new:N \l__zrefclever_label_b_tl
\l__zrefclever_label_enclcnt_a_tl 980 \tl_new:N \l__zrefclever_label_type_a_tl
\l__zrefclever_label_enclcnt_b_tl 981 \tl_new:N \l__zrefclever_label_type_b_tl
\l__zrefclever_label_enclval_a_tl 982 \tl_new:N \l__zrefclever_label_enclcnt_a_tl
\l__zrefclever_label_enclval_b_tl 983 \tl_new:N \l__zrefclever_label_enclcnt_b_tl
984 \tl_new:N \l__zrefclever_label_enclval_a_tl
985 \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.

```

986 \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.

```

987 \cs_new_protected:Npn \__zrefclever_sort_labels:
988 {

```

Store label types sequence.

```

989 \seq_clear:N \l__zrefclever_label_types_seq
990 \bool_if:NF \l__zrefclever_page_ref_bool
991 {
992   \seq_map_function:NN
993   \l__zrefclever_zcref_labels_seq \__zrefclever_label_type_put_new_right:n
994 }

```

Sort.

```

995 \seq_sort:Nn \l__zrefclever_zcref_labels_seq
996 {
997   \zref@ifrefundefined {##1}
998   {
999     \zref@ifrefundefined {##2}
1000     {
1001       % Neither label is defined.
1002       \sort_return_same:
1003     }
1004     {
1005       % The second label is defined, but the first isn't, leave the
1006       % undefined first (to be more visible).
1007       \sort_return_same:
1008     }
1009   }
1010   {
1011     \zref@ifrefundefined {##2}
1012     {
1013       % The first label is defined, but the second isn't, bring the
1014       % second forward.
1015       \sort_return_swapped:
1016     }
1017     {
1018       % The interesting case: both labels are defined. The
1019       % reference to the "default" property/counter or to the page
1020       % are quite different from our perspective, they rely on
1021       % different fields and even use different information for
1022       % sorting, so we branch them here to specialized functions.
1023       \bool_if:NTF \l__zrefclever_page_ref_bool
1024       { \__zrefclever_sort_page_aux:nn {##1} {##2} }
1025       { \__zrefclever_sort_default_aux:nn {##1} {##2} }
1026     }
1027   }
1028 }
1029 }

```

(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>}

1030 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
1031 {

```

```

1032 \tl_set:Nx \l__zrefclever_label_type_a_tl
1033 { \zref@extractdefault {#1} {zc@type} { \c_empty_tl } }
1034 \tl_if_empty:NF \l__zrefclever_label_type_a_tl
1035 {
1036   \seq_if_in:NVF \l__zrefclever_label_types_seq \l__zrefclever_label_type_a_tl
1037   {
1038     \seq_put_right:NV
1039     \l__zrefclever_label_types_seq \l__zrefclever_label_type_a_tl
1040   }
1041 }
1042 }

```

(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.

```

1043 \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.

```

1044 \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>}

1045 \cs_new_protected:Npn \__zrefclever_sort_default_aux:nn #1#2
1046 {
1047   \tl_set:Nx \l__zrefclever_label_type_a_tl
1048   { \zref@extractdefault {#1} {zc@type} { \c_empty_tl } }
1049   \tl_set:Nx \l__zrefclever_label_type_b_tl
1050   { \zref@extractdefault {#2} {zc@type} { \c_empty_tl } }
1051
1052   \bool_if:nTF
1053   {
1054     % The second label has a type, but the first doesn't, leave the
1055     % undefined first (to be more visible).
1056     \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1057     ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1058   }
1059   { \sort_return_same: }
1060   {
1061     \bool_if:nTF
1062     {
1063       % The first label has a type, but the second doesn't, bring the
1064       % second forward.
1065       ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1066       \tl_if_empty_p:N \l__zrefclever_label_type_b_tl

```

```

1067 }
1068 { \sort_return_swapped: }
1069 {
1070   \bool_if:nTF
1071   {
1072     % Both labels have a type\dots{}
1073     ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1074     ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1075   }
1076   {
1077     \tl_if_eq:NNTF \l__zrefclever_label_type_a_tl \l__zrefclever_label_type_b_tl
1078     {
1079       % \dots{} and it's the same type.
1080       \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1081         { \zref@extractdefault {#1} {zc@enclcnt} { \c_empty_tl } }
1082       \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1083         { \tl_reverse_items:V \l__zrefclever_label_enclcnt_a_tl }
1084       \tl_set:Nx \l__zrefclever_label_enclcnt_c_tl
1085         { \zref@extractdefault {#2} {zc@enclcnt} { \c_empty_tl } }
1086       \tl_set:Nx \l__zrefclever_label_enclcnt_d_tl
1087         { \tl_reverse_items:V \l__zrefclever_label_enclcnt_b_tl }
1088       \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1089         { \zref@extractdefault {#1} {zc@enclval} { \c_empty_tl } }
1090       \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1091         { \tl_reverse_items:V \l__zrefclever_label_enclval_a_tl }
1092       \tl_set:Nx \l__zrefclever_label_enclval_c_tl
1093         { \zref@extractdefault {#2} {zc@enclval} { \c_empty_tl } }
1094       \tl_set:Nx \l__zrefclever_label_enclval_d_tl
1095         { \tl_reverse_items:V \l__zrefclever_label_enclval_b_tl }
1096
1097       \bool_set_false:N \l__zrefclever_sort_decided_bool
1098       % CHECK should I replace the tmp variables here?
1099       \tl_clear:N \l_tmpa_tl
1100       \tl_clear:N \l_tmpb_tl
1101       \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1102       {
1103         \tl_set:Nx \l_tmpa_tl
1104           { \tl_head:N \l__zrefclever_label_enclcnt_a_tl }
1105         \tl_set:Nx \l_tmpb_tl
1106           { \tl_head:N \l__zrefclever_label_enclcnt_b_tl }
1107
1108         \bool_if:nTF
1109         {
1110           % Both are empty, meaning: neither labels have any
1111           % (further) ‘‘enclosing counters’’ (left).
1112           \tl_if_empty_p:V \l_tmpa_tl &&
1113           \tl_if_empty_p:V \l_tmpb_tl
1114         }
1115         {
1116           \exp_args:Nxx \tl_if_eq:nnTF
1117             { \zref@extractdefault {#1} {zc@counter} { } }
1118             { \zref@extractdefault {#2} {zc@counter} { } }
1119             {
1120               \bool_set_true:N \l__zrefclever_sort_decided_bool

```

```

1121 \int_compare:nNnTF
1122 { \zref@extractdefault {#1} { zc@cntval } {-1} }
1123 >
1124 { \zref@extractdefault {#2} { zc@cntval } {-1} }
1125 { \sort_return_swapped: }
1126 { \sort_return_same: }
1127 }
1128 {
1129 \msg_warning:nnnn { zref-clever }
1130 { counters-not-nested } {#1} {#2}
1131 \bool_set_true:N \l__zrefclever_sort_decided_bool
1132 \sort_return_same:
1133 }
1134 }
1135 {
1136 \bool_if:nTF
1137 {
1138 % 'a' is empty (and 'b' is not), meaning: 'b'
1139 % is (possibly) nested in 'a'.
1140 \tl_if_empty_p:V \l_tmpa_tl
1141 }
1142 {
1143 \tl_set:Nx \l_tmpa_tl
1144 { {\zref@extractdefault {#1} { zc@counter } { }} }
1145 \exp_args:NNx \tl_if_in:NnTF
1146 \l__zrefclever_label_enclcnt_b_tl { \l_tmpa_tl }
1147 {
1148 \bool_set_true:N \l__zrefclever_sort_decided_bool
1149 \sort_return_same:
1150 }
1151 {
1152 \msg_warning:nnnn { zref-clever }
1153 { counters-not-nested } {#1} {#2}
1154 \bool_set_true:N \l__zrefclever_sort_decided_bool
1155 \sort_return_same:
1156 }
1157 }
1158 {
1159 \bool_if:nTF
1160 {
1161 % 'b' is empty (and 'a' is not), meaning:
1162 % 'a' is (possibly) nested in 'b'.
1163 \tl_if_empty_p:V \l_tmpb_tl
1164 }
1165 {
1166 \tl_set:Nx \l_tmpb_tl
1167 { {\zref@extractdefault {#2} { zc@counter } { }} }
1168 \exp_args:NNx \tl_if_in:NnTF
1169 \l__zrefclever_label_enclcnt_a_tl { \l_tmpb_tl }
1170 {
1171 \bool_set_true:N \l__zrefclever_sort_decided_bool
1172 \sort_return_swapped:
1173 }
1174 {

```

```

1175         \msg_warning:nnnn { zref-clever }
1176         { counters-not-nested } {#1} {#2}
1177         \bool_set_true:N \l__zrefclever_sort_decided_bool
1178         \sort_return_same:
1179     }
1180 }
1181 {
1182     % Neither is empty, meaning: we can
1183     % (possibly) compare the values of the
1184     % current enclosing counter in the loop,
1185     % if they are equal, we are still in the
1186     % loop, if they are not, a sorting
1187     % decision can be made directly.
1188     \tl_if_eq:NNTF \l_tmpa_tl \l_tmpb_tl
1189     {
1190         \int_compare:nNnTF
1191         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1192         =
1193         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1194         {
1195             \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1196             { \tl_tail:N \l__zrefclever_label_enclcnt_a_tl }
1197             \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1198             { \tl_tail:N \l__zrefclever_label_enclcnt_b_tl }
1199             \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1200             { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1201             \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1202             { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1203         }
1204         {
1205             \bool_set_true:N \l__zrefclever_sort_decided_bool
1206             \int_compare:nNnTF
1207             { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1208             >
1209             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1210             { \sort_return_swapped: }
1211             { \sort_return_same: }
1212         }
1213     }
1214     {
1215         \msg_warning:nnnn { zref-clever }
1216         { counters-not-nested } {#1} {#2}
1217         \bool_set_true:N \l__zrefclever_sort_decided_bool
1218         \sort_return_same:
1219     }
1220 }
1221 }
1222 }
1223 }
1224 }
1225 {
1226     % \dots{} and they are different types.
1227     \int_zero:N \l__zrefclever_sort_prior_a_int
1228     \int_zero:N \l__zrefclever_sort_prior_b_int

```



```

1229 % \cs{l__zrefclever_typesort_seq} was stored in reverse sequence,
1230 % and we compute the sort priorities in the negative
1231 % range, so that we can implicitly rely on '0' being the
1232 % 'last value'.
1233 \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1234 {
1235   \tl_if_eq:nnTF {##2} {{othertypes}}
1236   {
1237     \int_compare:nNt { \l__zrefclever_sort_prior_a_int } = { 0 }
1238     { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1239     \int_compare:nNt { \l__zrefclever_sort_prior_b_int } = { 0 }
1240     { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1241   }
1242   {
1243     \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1244     { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1245     {
1246       \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
1247       { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1248     }
1249   }
1250 }
1251 \bool_if:nTF
1252 {
1253   \int_compare_p:nNn
1254   { \l__zrefclever_sort_prior_a_int } <
1255   { \l__zrefclever_sort_prior_b_int }
1256 }
1257 { \sort_return_same: }
1258 {
1259   \bool_if:nTF
1260   {
1261     \int_compare_p:nNn
1262     { \l__zrefclever_sort_prior_a_int } >
1263     { \l__zrefclever_sort_prior_b_int }
1264   }
1265   { \sort_return_swapped: }
1266   {
1267     % Sort priorities are equal for different types:
1268     % the type that occurs first in \meta{labels}, as
1269     % given by the user, is kept (or brought) forward.
1270     \seq_map_inline:Nn \l__zrefclever_label_types_seq
1271     {
1272       \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1273       { \seq_map_break:n { \sort_return_same: } }
1274       {
1275         \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
1276         { \seq_map_break:n { \sort_return_swapped: } }
1277       }
1278     }
1279   }
1280 }
1281 }
1282 }

```

```

1283     {
1284         % Neither of the labels has a type. We can't do much of
1285         % meaningful here, but if it's the same counter, compare it.
1286         \exp_args:Nxx \tl_if_eq:nnTF
1287         { \zref@extractdefault {#1} { zc@counter } { } }
1288         { \zref@extractdefault {#2} { zc@counter } { } }
1289         {
1290             \int_compare:nNnTF
1291             { \zref@extractdefault {#1} { zc@cntval } {-1} }
1292             >
1293             { \zref@extractdefault {#2} { zc@cntval } {-1} }
1294             { \sort_return_swapped: }
1295             { \sort_return_same: }
1296         }
1297         { \sort_return_same: }
1298     }
1299 }
1300 }
1301 }

```

(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>}
1302 \cs_new_protected:Npn \__zrefclever_sort_page_aux:nn #1#2
1303 {
1304     \int_compare:nNnTF
1305     { \zref@extractdefault {#1} { zc@abspg } {-1} }
1306     >
1307     { \zref@extractdefault {#2} { zc@abspg } {-1} }
1308     { \sort_return_swapped: }
1309     { \sort_return_same: }
1310 }

```

(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.

```
1311 \bool_new:N \l_zrefclever_typeset_last_bool
1312 \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.

```
1313 \tl_new:N \l_zrefclever_typeset_queue_prev_tl
1314 \tl_new:N \l_zrefclever_typeset_queue_curr_tl
1315 \tl_new:N \l_zrefclever_type_first_label_tl
1316 \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.

```
1317 \int_new:N \l_zrefclever_label_count_int
1318 \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.

```
1319 \int_new:N \l_zrefclever_range_count_int
1320 \int_new:N \l_zrefclever_range_same_count_int
1321 \tl_new:N \l_zrefclever_range_beg_label_tl
1322 \bool_new:N \l_zrefclever_next_maybe_range_bool
1323 \bool_new:N \l_zrefclever_next_is_same_bool
1324 \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.

```

1325 \tl_new:N \l__zrefclever_namefont_tl
1326 \tl_new:N \l__zrefclever_reffont_out_tl
1327 \tl_new:N \l__zrefclever_reffont_in_tl
1328
1329 \tl_new:N \l__zrefclever_namesep_tl
1330 \tl_new:N \l__zrefclever_rangeseq_tl
1331 \tl_new:N \l__zrefclever_pairsep_tl
1332 \tl_new:N \l__zrefclever_listsep_tl
1333 \tl_new:N \l__zrefclever_lastsep_tl
1334 % ‘t’ for ‘type’
1335 \tl_new:N \l__zrefclever_tpairsep_tl
1336 \tl_new:N \l__zrefclever_tlistsep_tl
1337 \tl_new:N \l__zrefclever_tlastsep_tl
1338 \tl_new:N \l__zrefclever_noteseq_tl
1339 \tl_new:N \l__zrefclever_refpre_out_tl
1340 \tl_new:N \l__zrefclever_refpos_out_tl
1341 \tl_new:N \l__zrefclever_refpre_in_tl
1342 \tl_new:N \l__zrefclever_refpos_in_tl

```

(End definition for .)

`\l__zrefclever_type_name_tl` Auxiliary variables for `__zrefclever_get_ref_first:` and `__zrefclever_type_name_setup:`.

```

\l__zrefclever_name_in_link_bool
\l__zrefclever_name_format_tl
1343 \tl_new:N \l__zrefclever_type_name_tl
1344 \bool_new:N \l__zrefclever_name_in_link_bool
1345 \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`.

```

1346 \cs_new_protected:Npn \__zrefclever_typeset_refs:
1347 {
1348   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1349   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1350   \tl_clear:N \l__zrefclever_type_first_label_tl
1351   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1352   \tl_clear:N \l__zrefclever_range_beg_label_tl
1353   \int_zero:N \l__zrefclever_label_count_int
1354   \int_zero:N \l__zrefclever_type_count_int
1355   \int_zero:N \l__zrefclever_range_count_int
1356   \int_zero:N \l__zrefclever_range_same_count_int
1357
1358   % Get not-type-specific separators and refpre/pos options.
1359   \__zrefclever_get_option_with_transl:nN {tpairsep} \l__zrefclever_tpairsep_tl
1360   \__zrefclever_get_option_with_transl:nN {tlistsep} \l__zrefclever_tlistsep_tl
1361   \__zrefclever_get_option_with_transl:nN {tlastsep} \l__zrefclever_tlastsep_tl
1362   \__zrefclever_get_option_with_transl:nN {noteseq} \l__zrefclever_noteseq_tl
1363
1364   % Set the font option for this zcref call.

```

```

1365 \l__zrefclever_ref_typeset_font_tl
1366
1367 % Loop over the label list in sequence.
1368 \bool_set_false:N \l__zrefclever_typeset_last_bool
1369 \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1370 {
1371   \seq_pop_left:NN \l__zrefclever_zcref_labels_seq \l__zrefclever_label_a_tl
1372   \seq_if_empty:NTF \l__zrefclever_zcref_labels_seq
1373   {
1374     \tl_clear:N \l__zrefclever_label_b_tl
1375     \bool_set_true:N \l__zrefclever_typeset_last_bool
1376   }
1377   { \seq_get_left:NN \l__zrefclever_zcref_labels_seq \l__zrefclever_label_b_tl }
1378
1379 \bool_if:NTF \l__zrefclever_page_ref_bool
1380 {
1381   \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1382   \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1383 }
1384 {
1385   \tl_set:Nx \l__zrefclever_label_type_a_tl
1386   {
1387     \zref@extractdefault
1388     { \l__zrefclever_label_a_tl } { zc@type } { \c_empty_tl }
1389   }
1390   \tl_set:Nx \l__zrefclever_label_type_b_tl
1391   {
1392     \zref@extractdefault
1393     { \l__zrefclever_label_b_tl } { zc@type } { \c_empty_tl }
1394   }
1395 }
1396
1397 % First, we establish whether the ‘current label’ (i.e. ‘a’) is the
1398 % last one of its type. This can happen because the ‘next label’
1399 % (i.e. ‘b’) is of a different type (or different definition status),
1400 % or because we are at the end of the list.
1401 \bool_if:NTF \l__zrefclever_typeset_last_bool
1402 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1403 {
1404   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1405   {
1406     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1407     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1408     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1409   }
1410   {
1411     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1412     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1413     {
1414       % Neither is undefined, we must check the types.
1415       \bool_if:nTF
1416       % Both empty: same ‘type’.
1417       {
1418         \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&

```

```

1419         \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1420     }
1421     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1422     {
1423         \bool_if:nTF
1424             % Neither empty: compare types.
1425             {
1426                 ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1427                 ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1428             }
1429             {
1430                 \tl_if_eq:NNTF
1431                     \l__zrefclever_label_type_a_tl \l__zrefclever_label_type_b_tl
1432                     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1433                     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1434             }
1435             % One empty, the other not: different ‘types’.
1436             { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1437         }
1438     }
1439 }
1440 }
1441
1442 % Handle warnings in case of reference or type undefined.
1443 \zref@refused { \l__zrefclever_label_a_tl }
1444 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1445     {}
1446     {
1447         \tl_if_empty:NT \l__zrefclever_label_type_a_tl
1448         {
1449             \msg_warning:nxx { zref-clever } { missing-type }
1450             { \l__zrefclever_label_a_tl }
1451         }
1452     }
1453
1454 % Get type-specific separators, refpre/pos and font options, once per
1455 % type.
1456 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
1457     {
1458         \__zrefclever_get_option_plain:nN {namefont}          \l__zrefclever_namefont_tl
1459         \__zrefclever_get_option_plain:nN {reffont}          \l__zrefclever_reffont_out_tl
1460         \__zrefclever_get_option_plain:nN {reffont-in}       \l__zrefclever_reffont_in_tl
1461         \__zrefclever_get_option_with_transl:nN {namesep}     \l__zrefclever_namesep_tl
1462         \__zrefclever_get_option_with_transl:nN {rangesep}    \l__zrefclever_rangesep_tl
1463         \__zrefclever_get_option_with_transl:nN {pairsep}     \l__zrefclever_pairsep_tl
1464         \__zrefclever_get_option_with_transl:nN {listsep}     \l__zrefclever_listsep_tl
1465         \__zrefclever_get_option_with_transl:nN {lastsep}     \l__zrefclever_lastsep_tl
1466         \__zrefclever_get_option_with_transl:nN {refpre}      \l__zrefclever_refpre_out_tl
1467         \__zrefclever_get_option_with_transl:nN {refpos}      \l__zrefclever_refpos_out_tl
1468         \__zrefclever_get_option_with_transl:nN {refpre-in}   \l__zrefclever_refpre_in_tl
1469         \__zrefclever_get_option_with_transl:nN {refpos-in}   \l__zrefclever_refpos_in_tl
1470     }
1471
1472 % Here we send this to a couple of auxiliary functions for no other

```

```

1473      % reason than to keep this long function a little less unreadable.
1474      \bool_if:NTF \l__zrefclever_last_of_type_bool
1475      {
1476        % There exists no next label of the same type as the current.
1477        \__zrefclever_typeset_refs_aux_last_of_type:
1478      }
1479      {
1480        % There exists a next label of the same type as the current.
1481        \__zrefclever_typeset_refs_aux_not_last_of_type:
1482      }
1483    }
1484  }

```

(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.

```

1485  \cs_new_protected:Npn \__zrefclever_typeset_refs_aux_last_of_type:
1486  {
1487    % Process the current label to the current queue.
1488    \int_case:nnF { \l__zrefclever_label_count_int }
1489    {
1490      % It is the last label of its type, but also the first one, and that's
1491      % what matters here: just store it.
1492      { 0 }
1493      {
1494        \tl_set:NV \l__zrefclever_type_first_label_tl \l__zrefclever_label_a_tl
1495        \tl_set:NV \l__zrefclever_type_first_label_type_tl \l__zrefclever_label_type_a_tl
1496      }
1497
1498      % The last is the second: we have a pair (if not repeated).
1499      { 1 }
1500      {
1501        \int_compare:nNnF { \l__zrefclever_range_same_count_int } = {1}
1502        {
1503          \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1504          {
1505            \exp_not:V \l__zrefclever_pairsep_tl
1506            \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1507          }
1508        }
1509      }
1510    }
1511    % If neither the first, nor the second: we have the last label
1512    % on the current type list (if not repeated).
1513    {
1514      \int_case:nnF { \l__zrefclever_range_count_int }
1515      {
1516        % There was no range going on.
1517        {0}
1518        {
1519          \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1520          {
1521            \exp_not:V \l__zrefclever_lastsep_tl
1522            \__zrefclever_get_ref:V \l__zrefclever_label_a_tl

```

```

1523     }
1524 }
1525 % Last in the range is also the second in it.
1526 {1}
1527 {
1528   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1529   {
1530     % We know 'range_beg_label' is not empty, since this is the
1531     % second element in the range, but the third or more in the
1532     % type list.
1533     \exp_not:V \l__zrefclever_listsep_tl
1534     \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1535     \int_compare:nNnF { \l__zrefclever_range_same_count_int } = {1}
1536     {
1537       \exp_not:V \l__zrefclever_lastsep_tl
1538       \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1539     }
1540   }
1541 }
1542 }
1543 % Last in the range is third or more in it.
1544 {
1545   \int_case:nnF
1546   { \l__zrefclever_range_count_int - \l__zrefclever_range_same_count_int }
1547   {
1548     % Repetition, not a range.
1549     {0}
1550     {
1551       % If 'range_beg_label' is empty, it means it was also the
1552       % first of the type, and hence was already handled.
1553       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1554       {
1555         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1556         {
1557           \exp_not:V \l__zrefclever_lastsep_tl
1558           \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1559         }
1560       }
1561     }
1562     % A 'range', but with no skipped value, treat as list.
1563     {1}
1564     {
1565       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1566       {
1567         % Ditto.
1568         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1569         {
1570           \exp_not:V \l__zrefclever_listsep_tl
1571           \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1572         }
1573         \exp_not:V \l__zrefclever_lastsep_tl
1574         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1575       }
1576     }
1577   }

```



```

1577     }
1578     {
1579         % An actual range.
1580         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1581         {
1582             % Ditto.
1583             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1584             {
1585                 \exp_not:V \l__zrefclever_lastsep_tl
1586                 \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1587             }
1588             \exp_not:V \l__zrefclever_rangesep_tl
1589             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1590         }
1591     }
1592 }
1593 }
1594
1595 % Handle ‘‘range’’ option. The idea is simple: if the queue is not empty,
1596 % we replace it with the end of the range (or pair). We can still
1597 % retrieve the end of the range from \cs{l__zrefclever_label_a_tl} since we know to
1598 % be processing the last label of its type at this point.
1599 \bool_if:NT \l__zrefclever_typeset_range_bool
1600 {
1601     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
1602     {
1603         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1604         { }
1605         {
1606             \msg_warning:nxx { zref-clever } { single-element-range }
1607             { \l__zrefclever_type_first_label_type_tl }
1608         }
1609     }
1610     {
1611         \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1612         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1613         { }
1614         {
1615             \__zrefclever_labels_in_sequence:nn
1616             { \l__zrefclever_type_first_label_tl } { \l__zrefclever_label_a_tl }
1617         }
1618         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1619         {
1620             \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1621             { \exp_not:V \l__zrefclever_pairsep_tl }
1622             { \exp_not:V \l__zrefclever_rangesep_tl }
1623             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1624         }
1625     }
1626 }
1627
1628 % Now that the type is finished, we can add the name and the first ref to
1629 % the queue. Or, if ‘‘typset’’ option is not ‘‘both’’, handle it here
1630 % too.

```

```

1631 \_zrefclever_type_name_setup:
1632 \bool_if:nTF
1633 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
1634 {
1635   \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1636   { \_zrefclever_get_ref_first: }
1637 }
1638 {
1639   \bool_if:nTF
1640   { \l__zrefclever_typeset_ref_bool }
1641   {
1642     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1643     { \_zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
1644   }
1645   {
1646     \bool_if:nTF
1647     { \l__zrefclever_typeset_name_bool }
1648     {
1649       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1650       {
1651         \bool_if:NTF \l__zrefclever_name_in_link_bool
1652         {
1653           \exp_not:N \group_begin:
1654           \exp_not:V \l__zrefclever_namefont_tl
1655           % It's two '@s', but escaped for DocStrip.
1656           \exp_not:N \hyper@@link
1657           {
1658             \zref@ifrefcontainsprop
1659             { \l__zrefclever_type_first_label_tl } { urluse }
1660             {
1661               \zref@extractdefault
1662               { \l__zrefclever_type_first_label_tl }
1663               { urluse } {}
1664             }
1665             {
1666               \zref@extractdefault
1667               { \l__zrefclever_type_first_label_tl }
1668               { url } {}
1669             }
1670           }
1671           {
1672             \zref@extractdefault
1673             { \l__zrefclever_type_first_label_tl } { anchor } {}
1674           }
1675           { \exp_not:V \l__zrefclever_type_name_tl }
1676           \exp_not:N \group_end:
1677         }
1678         {
1679           \exp_not:N \group_begin:
1680           \exp_not:V \l__zrefclever_namefont_tl
1681           \exp_not:V \l__zrefclever_type_name_tl
1682           \exp_not:N \group_end:
1683         }
1684       }
1685     }
1686   }

```

```

1685     }
1686     {
1687         % This case would correspond to "typeset=none" but should not
1688         % happen, given the options are set up to typeset at least one
1689         % of "ref" or "name", but a sensible fallback, equal to the
1690         % behavior of 'both'.
1691         \tl_put_left:Nx
1692         \l__zrefclever_typeset_queue_curr_tl { \__zrefclever_get_ref_first: }
1693     }
1694 }
1695 }
1696
1697 % Typeset the previous type, if there is one.
1698 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
1699 {
1700     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
1701     { \l__zrefclever_tlistsep_tl }
1702     \l__zrefclever_typeset_queue_prev_tl
1703 }
1704
1705 % Wrap up loop, or prepare for next iteration.
1706 \bool_if:NTF \l__zrefclever_typeset_last_bool
1707 {
1708     % We are finishing, typeset the current queue.
1709     \int_case:nnF { \l__zrefclever_type_count_int }
1710     {
1711         % Single type.
1712         { 0 }
1713         { \l__zrefclever_typeset_queue_curr_tl }
1714         % Pair of types.
1715         { 1 }
1716         {
1717             \l__zrefclever_tpairsep_tl
1718             \l__zrefclever_typeset_queue_curr_tl
1719         }
1720     }
1721     {
1722         % Last in list of types.
1723         \l__zrefclever_tlastsep_tl
1724         \l__zrefclever_typeset_queue_curr_tl
1725     }
1726 }
1727 {
1728     % There are further labels, set variables for next iteration.
1729     \tl_set_eq:NN
1730     \l__zrefclever_typeset_queue_prev_tl \l__zrefclever_typeset_queue_curr_tl
1731     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1732     \tl_clear:N \l__zrefclever_type_first_label_tl
1733     \tl_clear:N \l__zrefclever_type_first_label_type_tl
1734     \tl_clear:N \l__zrefclever_range_beg_label_tl
1735     \int_zero:N \l__zrefclever_label_count_int
1736     \int_incr:N \l__zrefclever_type_count_int
1737     \int_zero:N \l__zrefclever_range_count_int
1738     \int_zero:N \l__zrefclever_range_same_count_int

```

```

1739     }
1740 }

```

(End definition for `_zrefclever_typeset_refs_aux_last_of_type:`)

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

```

1741 \cs_new_protected:Npn \_zrefclever_typeset_refs_aux_not_last_of_type:
1742 {
1743   % Signal if next label may form a range with the current one (of
1744   % course, only considered if compression is enabled in the first
1745   % place).
1746   \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1747   \bool_set_false:N \l__zrefclever_next_is_same_bool
1748   \bool_lazy_and:nnT
1749     { \l__zrefclever_typeset_compress_bool }
1750     % Currently no-op, but kept as ‘handle’ to inhibit compression of
1751     % individual labels.
1752     { ! \l__zrefclever_range_inhibit_next_bool }
1753   {
1754     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1755     { }
1756     {
1757       \__zrefclever_labels_in_sequence:nn
1758         { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
1759     }
1760   }
1761
1762   % Process the current label to the current queue.
1763   \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
1764   {
1765     % Current label is the first of its type (also not the last, but it
1766     % doesn’t matter here): just store the label.
1767     \tl_set:NV \l__zrefclever_type_first_label_tl \l__zrefclever_label_a_tl
1768     \tl_set:NV \l__zrefclever_type_first_label_type_tl \l__zrefclever_label_type_a_tl
1769
1770     % If the next label may be part of a range, we set ‘range_beg_label’
1771     % to ‘empty’ (we deal with it as the ‘first’, and must do it
1772     % there, to handle hyperlinking), but also step the range counters.
1773     \bool_if:NT \l__zrefclever_next_maybe_range_bool
1774     {
1775       \tl_clear:N \l__zrefclever_range_beg_label_tl
1776       \int_incr:N \l__zrefclever_range_count_int
1777       \bool_if:NT \l__zrefclever_next_is_same_bool
1778       { \int_incr:N \l__zrefclever_range_same_count_int }
1779     }
1780   }
1781   {
1782     % Current label is neither the first (nor the last) of its
1783     % type.
1784     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1785     {
1786       % Starting, or continuing a range.
1787       \int_compare:nNnTF
1788         { \l__zrefclever_range_count_int } = { 0 }

```

```

1789 {
1790 % There was no range going, we are starting one.
1791 \tl_set:NV \l__zrefclever_range_beg_label_tl \l__zrefclever_label_a_tl
1792 \int_incr:N \l__zrefclever_range_count_int
1793 \bool_if:NT \l__zrefclever_next_is_same_bool
1794 { \int_incr:N \l__zrefclever_range_same_count_int }
1795 }
1796 {
1797 % Second or more in the range, but not the last.
1798 \int_incr:N \l__zrefclever_range_count_int
1799 \bool_if:NT \l__zrefclever_next_is_same_bool
1800 { \int_incr:N \l__zrefclever_range_same_count_int }
1801 }
1802 }
1803 {
1804 % Next element is not in sequence, meaning: there was no range, or
1805 % we are closing one.
1806 \int_case:nnF { \l__zrefclever_range_count_int }
1807 {
1808 % There was no range going on.
1809 {0}
1810 {
1811 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1812 {
1813 \exp_not:V \l__zrefclever_listsep_tl
1814 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1815 }
1816 }
1817 % Last is second in the range: if 'range_same_count' is also
1818 % '1', it's a repetition (drop it), otherwise, it's a 'pair
1819 % within a list'', treat as list.
1820 {1}
1821 {
1822 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1823 {
1824 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1825 {
1826 \exp_not:V \l__zrefclever_listsep_tl
1827 \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1828 }
1829 \int_compare:nNnF { \l__zrefclever_range_same_count_int } = {1}
1830 {
1831 \exp_not:V \l__zrefclever_listsep_tl
1832 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1833 }
1834 }
1835 }
1836 }
1837 {
1838 % Last is third or more in the range: if 'range_count' and
1839 % 'range_same_count' are the same, its a repetition (drop it),
1840 % if they differ by '1', its a list, if they differ by more,
1841 % it is a real range.
1842 \int_case:nnF

```

```

1843 { \l__zrefclever_range_count_int - \l__zrefclever_range_same_count_int }
1844 {
1845   {0}
1846   {
1847     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1848     {
1849       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1850       {
1851         \exp_not:V \l__zrefclever_listsep_tl
1852         \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1853       }
1854     }
1855   }
1856   {1}
1857   {
1858     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1859     {
1860       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1861       {
1862         \exp_not:V \l__zrefclever_listsep_tl
1863         \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1864       }
1865       \exp_not:V \l__zrefclever_listsep_tl
1866       \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1867     }
1868   }
1869 }
1870 {
1871   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1872   {
1873     \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1874     {
1875       \exp_not:V \l__zrefclever_listsep_tl
1876       \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1877     }
1878     \exp_not:V \l__zrefclever_rangesep_tl
1879     \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1880   }
1881 }
1882 }
1883 % Reset counters.
1884 \int_zero:N \l__zrefclever_range_count_int
1885 \int_zero:N \l__zrefclever_range_same_count_int
1886 }
1887 }
1888 % Step label counter for next iteration.
1889 \int_incr:N \l__zrefclever_label_count_int
1890 }

```

(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.

```

1891 \cs_new:Npn \_zrefclever_get_ref:n #1
1892 {
1893   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
1894   {
1895     \bool_if:nTF
1896       { \l__zrefclever_use_hyperref_bool && ! \l__zrefclever_link_star_bool }
1897       {
1898         \exp_not:N \group_begin:
1899         \exp_not:V \l__zrefclever_reffont_out_tl
1900         \exp_not:V \l__zrefclever_refpre_out_tl
1901         \exp_not:N \group_begin:
1902         \exp_not:V \l__zrefclever_reffont_in_tl
1903         % It's two '@s', but escaped for DocStrip.
1904         \exp_not:N \hyper@@link
1905         {
1906           \zref@ifrefcontainsprop {#1} { urluse }
1907           { \zref@extractdefault {#1} { urluse } {} }
1908           { \zref@extractdefault {#1} { url } {} }
1909         }
1910         { \zref@extractdefault {#1} { anchor } {} }
1911         {
1912           \exp_not:V \l__zrefclever_refpre_in_tl
1913           \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } {}
1914           \exp_not:V \l__zrefclever_refpos_in_tl
1915         }
1916         \exp_not:N \group_end:
1917         \exp_not:V \l__zrefclever_refpos_out_tl
1918         \exp_not:N \group_end:
1919       }
1920       {
1921         \exp_not:N \group_begin:
1922         \exp_not:V \l__zrefclever_reffont_out_tl
1923         \exp_not:V \l__zrefclever_refpre_out_tl
1924         \exp_not:N \group_begin:
1925         \exp_not:V \l__zrefclever_reffont_in_tl
1926         \exp_not:V \l__zrefclever_refpre_in_tl
1927         \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } {}
1928         \exp_not:V \l__zrefclever_refpos_in_tl
1929         \exp_not:N \group_end:
1930         \exp_not:V \l__zrefclever_refpos_out_tl
1931         \exp_not:N \group_end:
1932       }
1933     }
1934     { \exp_not:N \zref@default }
1935   }
1936 \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.

```

1937 \cs_new_protected:Npn \_zrefclever_type_name_setup:
1938 {
1939   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1940   { \tl_clear:N \l__zrefclever_type_name_tl }
1941   {
1942     \tl_if_empty:nTF \l__zrefclever_type_first_label_type_tl
1943     { \tl_clear:N \l__zrefclever_type_name_tl }
1944     {

```

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

```

1945   \bool_lazy_or:nnTF
1946   { \l__zrefclever_capitalize_bool }
1947   {
1948     \l__zrefclever_capitalize_first_bool &&
1949     \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
1950   }
1951   { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
1952   { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
1953   \bool_lazy_and:nnTF
1954   { \l__zrefclever_abbrev_bool }
1955   {
1956     ! \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 } ||
1957     ! \l__zrefclever_noabbrev_first_bool
1958   }
1959   { \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab } }
1960   % If the queue is empty, we have a singular, otherwise, plural.
1961   \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
1962   { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
1963   { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
1964
1965   \prop_get:cVNF
1966   { \l__zrefclever_type \l__zrefclever_type_first_label_type_tl _options_prop }
1967   \l__zrefclever_name_format_tl
1968   \l__zrefclever_type_name_tl
1969   {
1970     \_zrefclever_if_translation:xxTF
1971     { \l__zrefclever_ref_language_tl }
1972     {
1973       zrefclever-type- \l__zrefclever_type_first_label_type_tl -
1974       \l__zrefclever_name_format_tl
1975     }
1976     {
1977       \_zrefclever_get_translation_for:nxx { \l__zrefclever_type_name_tl }
1978       { \l__zrefclever_ref_language_tl }
1979       {
1980         zrefclever-type- \l__zrefclever_type_first_label_type_tl -
1981         \l__zrefclever_name_format_tl
1982       }
1983     }
1984   }

```



```

1985         \tl_clear:N \l__zrefclever_type_name_tl
1986         \msg_warning:nxx { zref-clever } { missing-name }
1987         { \l__zrefclever_type_first_label_type_tl }
1988     }
1989 }
1990 }
1991 }

```

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

```

1992 \bool_lazy_any:nTF
1993 {
1994     { ! \l__zrefclever_use_hyperref_bool }
1995     { \l__zrefclever_link_star_bool }
1996     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
1997     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
1998 }
1999 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2000 {
2001     \bool_lazy_any:nTF
2002     {
2003         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2004         {
2005             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2006             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2007         }
2008         {
2009             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2010             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2011             \l__zrefclever_typeset_last_bool &&
2012             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2013         }
2014     }
2015     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2016     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2017 }
2018 }

```

(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.

```

2019 \cs_new:Npn \_zrefclever_get_ref_first:
2020 {
2021     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2022     { \exp_not:N \zref@default }
2023     {
2024         \bool_if:NTF \l__zrefclever_name_in_link_bool
2025         {
2026             \zref@ifrefcontainsprop
2027             { \l__zrefclever_type_first_label_tl } { \l__zrefclever_ref_property_tl }
2028             {
2029                 % It's two '@s', but escaped for DocStrip.
2030                 \exp_not:N \hyper@@link
2031                 {

```

```

2032 \zref@ifrefcontainsprop
2033 { \l__zrefclever_type_first_label_tl } { urluse }
2034 {
2035   \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2036   { urluse } {}
2037 }
2038 {
2039   \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2040   { url } {}
2041 }
2042 }
2043 {
2044   \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2045   { anchor } {}
2046 }
2047 {
2048   \exp_not:N \group_begin:
2049   \exp_not:V \l__zrefclever_namefont_tl
2050   \exp_not:V \l__zrefclever_type_name_tl
2051   \exp_not:N \group_end:
2052   \exp_not:V \l__zrefclever_namesep_tl
2053   \exp_not:N \group_begin:
2054   \exp_not:V \l__zrefclever_reffont_out_tl
2055   \exp_not:V \l__zrefclever_refpre_out_tl
2056   \exp_not:N \group_begin:
2057   \exp_not:V \l__zrefclever_reffont_in_tl
2058   \exp_not:V \l__zrefclever_refpre_in_tl
2059   \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2060   { \l__zrefclever_ref_property_tl } {}
2061   \exp_not:V \l__zrefclever_refpos_in_tl
2062   \exp_not:N \group_end:
2063   % hyperlink makes it's own group, we'd like to close the
2064   % 'refpre-out' group after 'refpos-out', but... we close
2065   % it here, and give the trailing 'refpos-out' its own
2066   % group. This will result that formatting given to
2067   % 'refpre-out' will not reach 'refpos-out', but I see no
2068   % alternative, and this has to be handled specially.
2069   \exp_not:N \group_end:
2070 }
2071 \exp_not:N \group_begin:
2072 % Ditto: special treatment.
2073 \exp_not:V \l__zrefclever_reffont_out_tl
2074 \exp_not:V \l__zrefclever_refpos_out_tl
2075 \exp_not:N \group_end:
2076 }
2077 {
2078   \exp_not:N \group_begin:
2079   \exp_not:V \l__zrefclever_namefont_tl
2080   \exp_not:V \l__zrefclever_type_name_tl
2081   \exp_not:N \group_end:
2082   \exp_not:V \l__zrefclever_namesep_tl
2083   \exp_not:N \zref@default
2084 }
2085 }

```

```

2086 {
2087   \tl_if_empty:NTF \l__zrefclever_type_name_tl
2088   {
2089     \exp_not:N \zref@default
2090     \exp_not:V \l__zrefclever_namesep_tl
2091   }
2092   {
2093     \exp_not:N \group_begin:
2094     \exp_not:V \l__zrefclever_namefont_tl
2095     \exp_not:V \l__zrefclever_type_name_tl
2096     \exp_not:N \group_end:
2097     \exp_not:V \l__zrefclever_namesep_tl
2098   }
2099   \zref@ifrefcontainsprop
2100   { \l__zrefclever_type_first_label_tl } { \l__zrefclever_ref_property_tl }
2101   {
2102     \bool_if:nTF
2103     {
2104       \l__zrefclever_use_hyperref_bool &&
2105       ! \l__zrefclever_link_star_bool
2106     }
2107     {
2108       \exp_not:N \group_begin:
2109       \exp_not:V \l__zrefclever_reffont_out_tl
2110       \exp_not:V \l__zrefclever_refpre_out_tl
2111       \exp_not:N \group_begin:
2112       \exp_not:V \l__zrefclever_reffont_in_tl
2113       % It's two '@s', but escaped for DocStrip.
2114       \exp_not:N \hyper@@link
2115       {
2116         \zref@ifrefcontainsprop
2117         { \l__zrefclever_type_first_label_tl } { urluse }
2118         {
2119           \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2120           { urluse } {}
2121         }
2122         {
2123           \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2124           { url } {}
2125         }
2126       }
2127     }
2128     {
2129       \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2130       { anchor } {}
2131     }
2132     {
2133       \exp_not:V \l__zrefclever_refpre_in_tl
2134       \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2135       { \l__zrefclever_ref_property_tl } {}
2136       \exp_not:V \l__zrefclever_refpos_in_tl
2137     }
2138     \exp_not:N \group_end:
2139     \exp_not:V \l__zrefclever_refpos_out_tl
2140     \exp_not:N \group_end:

```

```

2140     }
2141     {
2142         \exp_not:N \group_begin:
2143         \exp_not:V \l__zrefclever_reffont_out_tl
2144         \exp_not:V \l__zrefclever_refpre_out_tl
2145         \exp_not:N \group_begin:
2146         \exp_not:V \l__zrefclever_reffont_in_tl
2147         \exp_not:V \l__zrefclever_refpre_in_tl
2148         \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2149         { \l__zrefclever_ref_property_tl } {}
2150         \exp_not:V \l__zrefclever_refpos_in_tl
2151         \exp_not:N \group_end:
2152         \exp_not:V \l__zrefclever_refpos_out_tl
2153         \exp_not:N \group_end:
2154     }
2155 }
2156 { \exp_not:N \zref@default }
2157 }
2158 }
2159 }

```

(End definition for __zrefclever_get_ref_first:.)

__zrefclever_get_option_with_transl:nN

```

2160 % \Arg{option} \Arg{var to store result}
2161 \cs_new_protected:Npn \__zrefclever_get_option_with_transl:nN #1#2
2162 {
2163     % First attempt options stored in \cs{l__zrefclever_ref_options_prop}.
2164     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2165     {
2166         % If not found, try the type specific options.
2167         \bool_lazy_all:nTF
2168         {
2169             { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2170             {
2171                 \prop_if_exist_p:c
2172                 { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop }
2173             }
2174             {
2175                 \prop_if_in_p:cn
2176                 { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1}
2177             }
2178         }
2179         {
2180             \prop_get:cnN
2181             { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1} #2
2182         }
2183         {
2184             % If not found, try the type specific translations.
2185             \__zrefclever_if_translation:xxTF
2186             { \l__zrefclever_ref_language_tl }
2187             { zrefclever-type- \l__zrefclever_label_type_a_tl - #1 }
2188             {
2189                 \__zrefclever_get_translation_for:nxx {#2}

```

```

2190         { \l__zrefclever_ref_language_tl }
2191         { zrefclever-type- \l__zrefclever_label_type_a_tl - #1 }
2192     }
2193     {
2194         % If not found, try general translations. We are not
2195         % controlling for their existence, but we must make sure all
2196         % options being retrieved with
2197         % \cs{__zrefclever_get_option_with_transl:nN} have their values set for
2198         % ‘English’ and ‘fallback’.
2199         \__zrefclever_get_translation_for:nxx {#2}
2200         { \l__zrefclever_ref_language_tl }
2201         { zrefclever-default- #1 }
2202     }
2203 }
2204 }
2205 }

```

(End definition for __zrefclever_get_option_with_transl:nN.)

__zrefclever_get_option_plain:nN

```

2206 \cs_new_protected:Npn \__zrefclever_get_option_plain:nN #1#2
2207 {
2208     % First attempt options stored in \cs{l__zrefclever_ref_options_prop}.
2209     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2210     {
2211         % If not found, try the type specific options.
2212         \bool_lazy_and:nnTF
2213         { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2214         {
2215             \prop_if_exist_p:c
2216             { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop }
2217         }
2218         {
2219             \prop_get:cnNF
2220             { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1} #2
2221             { \tl_clear:N #2 }
2222         }
2223         { \tl_clear:N #2 }
2224     }
2225 }

```

(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”.

```

2226 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
2227 {
2228     \bool_if:NTF \l__zrefclever_page_ref_bool
2229     {
2230         \exp_args:Nxx \tl_if_eq:nnT
2231         { \zref@extractdefault {#1} { zc@pgfmt } { } }
2232         { \zref@extractdefault {#2} { zc@pgfmt } { } }
2233         {

```

```

2234 \int_compare:nNnTF
2235 { \zref@extractdefault {#1} { zc@pgval } {-2} + 1 }
2236 =
2237 { \zref@extractdefault {#2} { zc@pgval } {-1} }
2238 { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2239 {
2240 \int_compare:nNnT
2241 { \zref@extractdefault {#1} { zc@pgval } {-1} }
2242 =
2243 { \zref@extractdefault {#2} { zc@pgval } {-1} }
2244 {
2245 \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2246 \bool_set_true:N \l__zrefclever_next_is_same_bool
2247 }
2248 }
2249 }
2250 }
2251 {
2252 \exp_args:Nxx \tl_if_eq:nnT
2253 { \zref@extractdefault {#1} { zc@counter } { } }
2254 { \zref@extractdefault {#2} { zc@counter } { } }
2255 {
2256 \exp_args:Nxx \tl_if_eq:nnT
2257 { \zref@extractdefault {#1} { zc@enclval } { } }
2258 { \zref@extractdefault {#2} { zc@enclval } { } }
2259 {
2260 \int_compare:nNnTF
2261 { \zref@extractdefault {#1} { zc@cntval } {-2} + 1 }
2262 =
2263 { \zref@extractdefault {#2} { zc@cntval } {-1} }
2264 { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2265 {
2266 \int_compare:nNnT
2267 { \zref@extractdefault {#1} { zc@cntval } {-1} }
2268 =
2269 { \zref@extractdefault {#2} { zc@cntval } {-1} }
2270 {
2271 \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2272 \bool_set_true:N \l__zrefclever_next_is_same_bool
2273 }
2274 }
2275 }
2276 }
2277 }
2278 }

```

(End definition for `__zrefclever_labels_in_sequence:nn`.)

10 Fallback 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’.

```

2279 \__zrefclever_add_default_translation:nnn { fallback } { namesep } {\nobreakspace}
2280 \__zrefclever_add_default_translation:nnn { fallback } { pairsep } {,~}
2281 \__zrefclever_add_default_translation:nnn { fallback } { listsep } {,~}
2282 \__zrefclever_add_default_translation:nnn { fallback } { lastsep } {,~}
2283 \__zrefclever_add_default_translation:nnn { fallback } { tpairsep } {,~}
2284 \__zrefclever_add_default_translation:nnn { fallback } { tlistsep } {,~}
2285 \__zrefclever_add_default_translation:nnn { fallback } { tlastsep } {,~}
2286 \__zrefclever_add_default_translation:nnn { fallback } { notesep } {~}
2287 \__zrefclever_add_default_translation:nnn { fallback } { rangesep } {\textendash}
2288 \__zrefclever_add_default_translation:nnn { fallback } { refpre } {}
2289 \__zrefclever_add_default_translation:nnn { fallback } { refpos } {}
2290 \__zrefclever_add_default_translation:nnn { fallback } { refpre-in } {}
2291 \__zrefclever_add_default_translation:nnn { fallback } { refpos-in } {}
2292 \endpackage

```

11 Localization

```

2293 \iflang-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.

```

2294 \ProvideDictionaryFor{English}{zref-clever}
2295
2296 \zcdicDefaultTransl{namesep}{\nobreakspace}
2297 \zcdicDefaultTransl{pairsep}{~and\nobreakspace}
2298 \zcdicDefaultTransl{listsep}{,~}
2299 \zcdicDefaultTransl{lastsep}{~and\nobreakspace}
2300 \zcdicDefaultTransl{tpairsep}{~and\nobreakspace}
2301 \zcdicDefaultTransl{tlistsep}{,~}
2302 \zcdicDefaultTransl{tlastsep}{,~and\nobreakspace}
2303 \zcdicDefaultTransl{notesep}{~}
2304 \zcdicDefaultTransl{rangesep}{~to\nobreakspace}
2305 \zcdicDefaultTransl{refpre}{}
2306 \zcdicDefaultTransl{refpos}{}
2307 \zcdicDefaultTransl{refpre-in}{}
2308 \zcdicDefaultTransl{refpos-in}{}
2309
2310 \zcdicTypeTransl{part}{name-sg}{part}
2311 \zcdicTypeTransl{part}{name-pl}{parts}
2312 \zcdicTypeTransl{part}{Name-sg}{Part}
2313 \zcdicTypeTransl{part}{Name-pl}{Parts}
2314 \zcdicTypeTransl{part}{name-ab-sg}{part}
2315 \zcdicTypeTransl{part}{name-ab-pl}{parts}
2316 \zcdicTypeTransl{part}{Name-ab-sg}{Part}
2317 \zcdicTypeTransl{part}{Name-ab-pl}{Parts}
2318
2319 \zcdicTypeTransl{chapter}{name-sg}{chapter}
2320 \zcdicTypeTransl{chapter}{name-pl}{chapters}
2321 \zcdicTypeTransl{chapter}{Name-sg}{Chapter}

```

2322 \zcDicTypeTransl{chapter}{Name-pl}{Chapters}
2323 \zcDicTypeTransl{chapter}{name-ab-sg}{chapter}
2324 \zcDicTypeTransl{chapter}{name-ab-pl}{chapters}
2325 \zcDicTypeTransl{chapter}{Name-ab-sg}{Chapter}
2326 \zcDicTypeTransl{chapter}{Name-ab-pl}{Chapters}
2327
2328 \zcDicTypeTransl{section}{name-sg}{section}
2329 \zcDicTypeTransl{section}{name-pl}{sections}
2330 \zcDicTypeTransl{section}{Name-sg}{Section}
2331 \zcDicTypeTransl{section}{Name-pl}{Sections}
2332 \zcDicTypeTransl{section}{name-ab-sg}{section}
2333 \zcDicTypeTransl{section}{name-ab-pl}{sections}
2334 \zcDicTypeTransl{section}{Name-ab-sg}{Section}
2335 \zcDicTypeTransl{section}{Name-ab-pl}{Sections}
2336
2337 \zcDicTypeTransl{paragraph}{name-sg}{paragraph}
2338 \zcDicTypeTransl{paragraph}{name-pl}{paragraphs}
2339 \zcDicTypeTransl{paragraph}{Name-sg}{Paragraph}
2340 \zcDicTypeTransl{paragraph}{Name-pl}{Paragraphs}
2341 \zcDicTypeTransl{paragraph}{name-ab-sg}{par.}
2342 \zcDicTypeTransl{paragraph}{name-ab-pl}{par.}
2343 \zcDicTypeTransl{paragraph}{Name-ab-sg}{Par.}
2344 \zcDicTypeTransl{paragraph}{Name-ab-pl}{Par.}
2345
2346 \zcDicTypeTransl{page}{name-sg}{page}
2347 \zcDicTypeTransl{page}{name-pl}{pages}
2348 \zcDicTypeTransl{page}{Name-sg}{Page}
2349 \zcDicTypeTransl{page}{Name-pl}{Pages}
2350 \zcDicTypeTransl{page}{name-ab-sg}{p.}
2351 \zcDicTypeTransl{page}{name-ab-pl}{pp.}
2352 \zcDicTypeTransl{page}{Name-ab-sg}{Page}
2353 \zcDicTypeTransl{page}{Name-ab-pl}{Pages}
2354
2355 \zcDicTypeTransl{figure}{name-sg}{figure}
2356 \zcDicTypeTransl{figure}{name-pl}{figures}
2357 \zcDicTypeTransl{figure}{Name-sg}{Figure}
2358 \zcDicTypeTransl{figure}{Name-pl}{Figures}
2359 \zcDicTypeTransl{figure}{name-ab-sg}{fig.}
2360 \zcDicTypeTransl{figure}{name-ab-pl}{figs.}
2361 \zcDicTypeTransl{figure}{Name-ab-sg}{Fig.}
2362 \zcDicTypeTransl{figure}{Name-ab-pl}{Figs.}
2363
2364 \zcDicTypeTransl{table}{name-sg}{table}
2365 \zcDicTypeTransl{table}{name-pl}{tables}
2366 \zcDicTypeTransl{table}{Name-sg}{Table}
2367 \zcDicTypeTransl{table}{Name-pl}{Tables}
2368 \zcDicTypeTransl{table}{name-ab-sg}{table}
2369 \zcDicTypeTransl{table}{name-ab-pl}{tables}
2370 \zcDicTypeTransl{table}{Name-ab-sg}{Table}
2371 \zcDicTypeTransl{table}{Name-ab-pl}{Tables}
2372
2373 \zcDicTypeTransl{equation}{name-sg}{equation}
2374 \zcDicTypeTransl{equation}{name-pl}{equations}
2375 \zcDicTypeTransl{equation}{Name-sg}{Equation}


```

2376 \zcDicTypeTransl{equation}{Name-pl}{Equations}
2377 \zcDicTypeTransl{equation}{name-ab-sg}{eq.}
2378 \zcDicTypeTransl{equation}{name-ab-pl}{eqs.}
2379 \zcDicTypeTransl{equation}{Name-ab-sg}{Eq.}
2380 \zcDicTypeTransl{equation}{Name-ab-pl}{Eqs.}
2381 \zcDicTypeTransl{equation}{refpre-in}{()}
2382 \zcDicTypeTransl{equation}{refpos-in}{()}
2383
2384 \zcDicTypeTransl{item}{name-sg}{item}
2385 \zcDicTypeTransl{item}{name-pl}{items}
2386 \zcDicTypeTransl{item}{Name-sg}{Item}
2387 \zcDicTypeTransl{item}{Name-pl}{Items}
2388 \zcDicTypeTransl{item}{name-ab-sg}{item}
2389 \zcDicTypeTransl{item}{name-ab-pl}{items}
2390 \zcDicTypeTransl{item}{Name-ab-sg}{Item}
2391 \zcDicTypeTransl{item}{Name-ab-pl}{Items}
2392 </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.

A		389, 404, 1097, 1368, 1407, 1421,
\AddToHook	31, 88, 257, 277, 394, 465, 489, 516	1432, 1611, 1746, 1747, 1999, 2016
\addtranslation	153, 157	\bool_set_true:N
\appendix	6, 7	253, 309, 310, 314, 320,
\appendixname	7	378, 383, 384, 1120, 1131, 1148,
\Arg	2160	1154, 1171, 1177, 1205, 1217, 1375,
\AtEndOfPackage	16, 487	1402, 1408, 1412, 1433, 1436, 2015,
		2238, 2245, 2246, 2264, 2271, 2272
B		\bool_until_do:Nn
\bool_case_true:	2	1101, 1369
\bool_if:NTF	398, 402, 990, 1023, 1379, 1401,	
	1474, 1599, 1620, 1651, 1706, 1773,	
	1777, 1784, 1793, 1799, 2024, 2228	
\bool_if:nTF	56, 1052, 1061, 1070,	
	1108, 1136, 1159, 1251, 1259, 1415,	
	1423, 1632, 1639, 1646, 1895, 2102	
\bool_lazy_all:nTF	2167	
\bool_lazy_and:nnTF	1748, 1953, 2212	
\bool_lazy_any:nTF	1992, 2001	
\bool_lazy_or:nnTF	960, 1945	
\bool_new:N	241, 302, 303, 328, 355, 364, 371,	
	372, 427, 428, 445, 446, 953, 1043,	
	1311, 1312, 1322, 1323, 1324, 1344	
\bool_set:Nn	959	
\bool_set_false:N	248, 262,	
	267, 286, 297, 315, 319, 379, 388,	
C		
clist commands:		
\clist_count:n	641, 703, 799, 861	
\clist_item:nn	655, 667, 677, 680, 683,	
	686, 689, 692, 695, 698, 813, 825,	
	835, 838, 841, 844, 847, 850, 853, 856	
\clist_map_inline:Nn	526, 544	
\clist_map_inline:nn	204, 570, 623, 645,	
	657, 708, 724, 803, 815, 866, 891, 921	
\cs	519, 1229, 1597, 2163, 2197, 2208	
cs commands:		
\cs:w	47	
\cs_end:	47	
\cs_generate_variant:Nn	52, 53, 146, 149, 1044, 1936	
\cs_if_exist:NTF	36, 45, 66	
\cs_new:Npn	34, 43, 54, 64, 75, 1891, 2019	

\cs_new_protected:Npn	144, 147, 152, 156, 165, 954, 987, 1030, 1045, 1302, 1346, 1485, 1741, 1937, 2161, 2206, 2226
\cs_new_protected:Npx	87
\cs_set_eq:NN	91
D	
\declaretranslation	148
\dots	533, 1072, 1079, 1226
E	
\endinput	12
exp commands:	
\exp_args:NNe	22
\exp_args:NNx	92, 1145, 1168
\exp_args:Nx	535, 546
\exp_args:Nxx	1116, 1286, 2230, 2252, 2256
\exp_not:N	105, 111, 1653, 1656, 1676, 1679, 1682, 1898, 1901, 1904, 1916, 1918, 1921, 1924, 1929, 1931, 1934, 2022, 2030, 2048, 2051, 2053, 2056, 2062, 2069, 2071, 2075, 2078, 2081, 2083, 2089, 2093, 2096, 2108, 2111, 2114, 2137, 2139, 2142, 2145, 2151, 2153, 2156
\exp_not:n	1505, 1521, 1533, 1537, 1557, 1570, 1573, 1585, 1588, 1621, 1622, 1654, 1675, 1680, 1681, 1813, 1826, 1831, 1851, 1862, 1865, 1875, 1878, 1899, 1900, 1902, 1912, 1914, 1917, 1922, 1923, 1925, 1926, 1928, 1930, 2049, 2050, 2052, 2054, 2055, 2057, 2058, 2061, 2073, 2074, 2079, 2080, 2082, 2090, 2094, 2095, 2097, 2109, 2110, 2112, 2132, 2135, 2138, 2143, 2144, 2146, 2147, 2150, 2152
F	
file commands:	
\file_if_exist:nTF	535, 546
\fmtversion	3
G	
group commands:	
\group_begin:	90, 956, 1653, 1679, 1898, 1901, 1921, 1924, 2048, 2053, 2056, 2071, 2078, 2093, 2108, 2111, 2142, 2145
\group_end:	93, 968, 1676, 1682, 1916, 1918, 1929, 1931, 2051, 2062, 2069, 2075, 2081, 2096, 2137, 2139, 2151, 2153
H	
\hyperlink	47
I	
\IfBooleanTF	972
\IfFormatAtLeastTF	3, 4
\IfTranslation	139
int commands:	
\int_case:nnTF	641, 799, 1488, 1514, 1545, 1709, 1806, 1842
\int_compare:nNnTF	1121, 1190, 1206, 1237, 1239, 1290, 1304, 1456, 1501, 1535, 1698, 1700, 1763, 1787, 1829, 2234, 2240, 2260, 2266
\int_compare_p:nNn	1253, 1261, 1949, 1956, 2012
\int_eval:n	87
\int_gincr:N	31
\int_incr:N	1736, 1776, 1778, 1792, 1794, 1798, 1800, 1889
\int_new:N	30, 976, 977, 1317, 1318, 1319, 1320
\int_set:Nn	1238, 1240, 1244, 1247
\int_use:N	26, 28, 32, 47
\int_zero:N	1227, 1228, 1353, 1354, 1355, 1356, 1735, 1737, 1738, 1884, 1885
iow commands:	
\iow_newline:	124, 128
K	
keys commands:	
\keys_define:nn	19, 171, 200, 223, 242, 281, 291, 304, 329, 338, 356, 365, 373, 406, 413, 429, 447, 461, 491, 561, 567, 592, 616, 716, 750, 775, 788, 874, 904, 933
\keys_set:nn	19, 606, 614, 773, 957
keyval commands:	
\keyval_parse:nnn	175, 227
L	
\label	7
\labelformat	2, 3
\LoadDictionaryFor	537, 548, 554
M	
\MessageBreak	10
\meta	1268
msg commands:	
\msg_line_context:	99, 104, 110, 119, 130, 132, 134, 136
\msg_new:nnn	97, 102, 108, 113, 115, 117, 122, 127, 129, 131, 133, 135
\msg_warning:nn	265, 295, 403, 409

<code>\msg_warning:nnn</code>	176, 228, 702, 720, 795, 860, 885, 940, 1449, 1606, 1986
<code>\msg_warning:nnnn</code>	1129, 1152, 1175, 1215
N	
<code>\NewDocumentCommand</code>	159, 162, 605, 609, 769, 950, 970
<code>\NewHook</code>	460
<code>\newlabel</code>	7
<code>\newtheorem</code>	6, 7
<code>\nobreakspace</code>	2279, 2296, 2297, 2299, 2300, 2302, 2304
<code>\noexpand</code>	125
P	
<code>\PackageError</code>	7
<code>\pagenumbering</code>	5
<code>\paragraph</code>	6
<code>\pkg</code>	518, 528, 531
prg commands:	
<code>\prg_generate_conditional_-variant:Nnn</code>	143
<code>\prg_new_conditional:Npnn</code>	137
<code>\prg_return_false:</code>	141
<code>\prg_return_true:</code>	140
<code>\ProcessKeysOptions</code>	16, 604
prop commands:	
<code>\prop_get:NnN</code>	2180
<code>\prop_get:NnNTF</code>	1965, 2164, 2209, 2219
<code>\prop_if_exist:NTF</code>	611, 782
<code>\prop_if_exist_p:N</code>	2171, 2215
<code>\prop_if_in_p:Nn</code>	57, 2175
<code>\prop_item:Nn</code>	22, 58
<code>\prop_new:N</code>	164, 222, 569, 612, 783
<code>\prop_put:Nnn</code>	169, 599, 653, 665, 675, 678, 681, 684, 687, 690, 693, 696, 762
<code>\prop_remove:Nn</code>	168, 598, 635, 757
<code>\providecommand</code>	3
<code>\ProvideDictionaryFor</code>	2294
<code>\ProvideDictTranslation</code>	160, 163
<code>\ProvidesExplPackage</code>	14
R	
<code>\refstepcounter</code>	3, 7
<code>\RequirePackage</code>	16, 17, 18, 399, 603
S	
<code>\SaveTranslationFor</code>	145
<code>\section</code>	6
seq commands:	
<code>\seq_clear:N</code>	352, 989
<code>\seq_get_left:NN</code>	1377
<code>\seq_if_empty:NTF</code>	1372
<code>\seq_if_in:NnTF</code>	206, 1036
<code>\seq_map_break:n</code>	78, 1273, 1276
<code>\seq_map_function:NN</code>	992
<code>\seq_map_indexed_inline:Nn</code>	1233
<code>\seq_map_inline:Nn</code>	1270
<code>\seq_map_tokens:Nn</code>	60
<code>\seq_new:N</code>	199, 337, 952, 986
<code>\seq_pop_left:NN</code>	1371
<code>\seq_put_right:Nn</code>	207, 1038
<code>\seq_reverse:N</code>	346
<code>\seq_set_from_clist:Nn</code>	342, 958
<code>\seq_sort:Nn</code>	995
<code>\setcounter</code>	5
sort commands:	
<code>\sort_return_same:</code>	29, 34, 1002, 1007, 1059, 1126, 1132, 1149, 1155, 1178, 1211, 1218, 1257, 1273, 1295, 1297, 1309
<code>\sort_return_swapped:</code>	29, 34, 1015, 1068, 1125, 1172, 1210, 1265, 1276, 1294, 1308
str commands:	
<code>\str_case:nnTF</code>	467, 495
<code>\str_if_eq:nnTF</code>	77
<code>\str_if_eq_p:nn</code>	1997, 2003, 2005, 2009
<code>\str_new:N</code>	412
<code>\str_set:Nn</code>	417, 419, 421, 423
<code>\subparagraph</code>	6
<code>\subsection</code>	6
<code>\subsubsection</code>	6
T	
T _E X and L ^A T _E X 2 _ε commands:	
<code>\@Alph</code>	7
<code>\@chapapp</code>	7
<code>\@currentcounter</code>	19, 22, 24, 26, 81, 83
<code>\@currentlabel</code>	3
<code>\@ifl@t@r</code>	3
<code>\@ifpackageloaded</code>	259, 279, 396, 523, 541
<code>\@trnslt@current@language</code>	520
<code>\@trnslt@language</code>	536, 547
<code>\bbl@loaded</code>	526
<code>\bbl@main@language</code>	525
<code>\c@</code>	3
<code>\c@page</code>	5, 91
<code>\cl@</code>	3
<code>\hyper@@link</code>	1656, 1904, 2030, 2114
<code>\p@...</code>	3
<code>\xpg@loaded</code>	544
<code>\xpg@main@language</code>	543
<code>\zref@addprop</code>	20, 23, 25, 27, 29, 33, 84, 85, 96
<code>\zref@default</code>	47, 1934, 2022, 2083, 2089, 2156

\zcheck	125	__zrefclever_get_ref_first:	...
\zclabel	7		36, 47, 1636, 1692, 2019
\zcpageref	27, 970	__zrefclever_get_translation_-	
\zceref	26, 27, 34, 36, 950, 973, 974	for:nnn	144, 146, 1977, 2189, 2199
\zcRefTypeSetup	19, 111, 609	__zrefclever_if_translation:nn	
\zcsetup	19, 605		137, 143
\zlabel	7	__zrefclever_if_translation:nnTF	
zrefclever internal commands:			1970, 2185
\l_zrefclever_abbrev_bool	445, 449, 1954	\l_zrefclever_label_a_tl	978, 1371, 1388, 1404, 1443,
\g_zrefclever_abbrev_int	30, 31, 32		1444, 1450, 1494, 1506, 1522, 1538,
__zrefclever_add_default_-			1574, 1589, 1616, 1623, 1754, 1758,
translation:nnn	152, 2279,		1767, 1791, 1814, 1832, 1866, 1879
	2280, 2281, 2282, 2283, 2284, 2285,	\l_zrefclever_label_b_tl	978, 1374, 1377, 1393, 1406, 1411, 1758
	2286, 2287, 2288, 2289, 2290, 2291	\l_zrefclever_label_count_int	35, 1317,
__zrefclever_add_type_translation:nnnn	156		1353, 1456, 1488, 1735, 1763, 1889
\l_zrefclever_capitalize_bool	427, 431, 1946	\l_zrefclever_label_enclcnt_a_-	
\l_zrefclever_capitalize_first_-		tl	978, 1080,
bool	428, 437, 1948		1082, 1083, 1104, 1169, 1195, 1196
__zrefclever_counter_reset_by:n	4, 36, 38, 40, 45, 47, 49, 54	\l_zrefclever_label_enclcnt_b_-	
__zrefclever_counter_reset_by_-		tl	978, 1084,
aux:nn	54		1086, 1087, 1106, 1146, 1197, 1198
__zrefclever_counter_reset_by_-		\l_zrefclever_label_enclval_a_-	
auxi:nnn	54	tl	978, 1088,
\l_zrefclever_counter_resetby_-			1090, 1091, 1191, 1199, 1200, 1207
prop	57, 58, 222, 229	\l_zrefclever_label_enclval_b_-	
\l_zrefclever_counter_resettters_-		tl	978, 1092,
seq	3, 60, 199, 206, 207		1094, 1095, 1193, 1201, 1202, 1209
\l_zrefclever_counter_type_prop	7, 22, 164, 177	\l_zrefclever_label_type_a_tl	978, 1032, 1034, 1036, 1039,
\l_zrefclever_current_language_-			1047, 1056, 1065, 1073, 1077, 1243,
tl	478, 506, 520		1272, 1381, 1385, 1418, 1426, 1431,
__zrefclever_declare_translation:nnn	147,		1447, 1495, 1768, 2169, 2172, 2176,
	149, 811, 823, 833, 836, 839, 842,		2181, 2187, 2191, 2213, 2216, 2220
	845, 848, 851, 854, 881, 911, 915, 944	\l_zrefclever_label_type_b_tl	978,
__zrefclever_get_enclosing_-			1049, 1057, 1066, 1074, 1077, 1246,
counters:n	4, 34, 39, 81		1275, 1382, 1390, 1419, 1427, 1431
__zrefclever_get_enclosing_-		__zrefclever_label_type_put_-	
counters_value:n	4, 34, 48, 83	new_right:n	28, 993, 1030
__zrefclever_get_option_-		\l_zrefclever_label_types_seq	28, 986, 989, 1036, 1039, 1270
plain:nN	1458, 1459, 1460, 2206	__zrefclever_labels_in_sequence:nn	1615, 1757, 2226
__zrefclever_get_option_with_-			
transl:nN	54, 55, 1359, 1360,	\l_zrefclever_last_of_type_bool	35, 1311, 1402, 1407, 1408,
	1361, 1362, 1461, 1462, 1463, 1464,		1412, 1421, 1432, 1433, 1436, 1474
	1465, 1466, 1467, 1468, 1469, 2160	\l_zrefclever_lastsep_tl	1333,
__zrefclever_get_ref:n	1506, 1522,		1465, 1521, 1537, 1557, 1573, 1585
	1534, 1538, 1558, 1571, 1574, 1586,	\l_zrefclever_link_star_bool	
	1589, 1623, 1643, 1814, 1827, 1832,		952, 959, 1896, 1995, 2105
	1852, 1863, 1866, 1876, 1879, 1891		

\l_zrefclever_listsep_tl	1332 , 1464 , 1533 , 1570 , 1813 ,	1319 , 1356 , 1501 , 1535 , 1546 , 1738 ,
	1826 , 1831 , 1851 , 1862 , 1865 , 1875	1778 , 1794 , 1800 , 1829 , 1843 , 1885
\l_zrefclever_main_language_tl 472 , 500 , 525 , 543 , 552 , 553	\l_zrefclever_rangesep_tl
\l_zrefclever_name_format_tl 1343 , 1951 , 1952 , 1330 , 1462 , 1588 , 1622 , 1878
	1959 , 1962 , 1963 , 1967 , 1974 , 1981	\l_zrefclever_ref_language_tl . .
\l_zrefclever_name_in_link_bool	. . 1343 , 1651 , 1999 , 2015 , 2016 , 2024 459 , 472 , 478 , 481 , 500 ,
\l_zrefclever_namefont_tl	1325 ,	506 , 509 , 1971 , 1978 , 2186 , 2190 , 2200
	1458 , 1654 , 1680 , 2049 , 2079 , 2094	\l_zrefclever_ref_options_prop .
\l_zrefclever_nameinlink_str 412 , 417 , 19 , 569 , 598 , 599 , 2164 , 2209
	419 , 421 , 423 , 1997 , 2003 , 2005 , 2009	\l_zrefclever_ref_property_tl . .
\l_zrefclever_nameinlink_tl . . 412	 12 , 240 , 247 ,
\l_zrefclever_namesep_tl 1329 , 1461 , 2052 , 2082 , 2090 , 2097	252 , 261 , 266 , 285 , 296 , 1893 , 1913 ,
\l_zrefclever_next_is_same_bool 35 , 53 , 1319 ,	1927 , 2027 , 2060 , 2100 , 2134 , 2149
	1747 , 1777 , 1793 , 1799 , 2246 , 2272	\l_zrefclever_ref_typeset_font_-
\l_zrefclever_next_maybe_range_-	. . 35 , 53 , 1319 , 1611 , 1620 , 1746 ,	tl 566 , 568 , 1365
bool	1773 , 1784 , 2238 , 2245 , 2264 , 2271	\l_zrefclever_reffont_in_tl 1327 ,
\l_zrefclever_noabbrev_first_- 446 , 455 , 1957	1460 , 1902 , 1925 , 2057 , 2112 , 2146
bool 966 , 1338 , 1362	\l_zrefclever_reffont_out_tl . . .
\l_zrefclever_notesept_tl 87 , 91 1326 , 1459 ,
_zrefclever_page_format_aux: 5 , 86 , 92 , 95	1899 , 1922 , 2054 , 2073 , 2109 , 2143
\g_zrefclever_page_format_tl 86	\l_zrefclever_refpos_in_tl 1342 ,
_zrefclever_page_numbering: 241 , 248 , 253 , 262 ,	1469 , 1914 , 1928 , 2061 , 2135 , 2150
\l_zrefclever_page_ref_bool . . .	267 , 286 , 297 , 990 , 1023 , 1379 , 2228	\l_zrefclever_refpos_out_tl 1340 ,
\l_zrefclever_pairsep_tl 1331 , 1463 , 1505 , 1621	1467 , 1917 , 1930 , 2074 , 2138 , 2152
_zrefclever_prop_put_non_- 165 , 177 , 229	\l_zrefclever_refpre_in_tl 1341 ,
empty:Nnn 35 , 1319 , 1352 ,	1468 , 1912 , 1926 , 2058 , 2132 , 2147
\l_zrefclever_range_beg_label_-	1534 , 1553 , 1558 , 1568 , 1571 , 1583 ,	\l_zrefclever_refpre_out_tl 1339 ,
tl	1586 , 1734 , 1775 , 1791 , 1824 , 1827 ,	1466 , 1900 , 1923 , 2055 , 2110 , 2144
	1849 , 1852 , 1860 , 1863 , 1873 , 1876	\l_zrefclever_setup_language_tl
\l_zrefclever_range_count_int 35 , 607 ,
	1319 , 1355 , 1514 , 1546 , 1737 , 1776 ,	771 , 811 , 823 , 833 , 836 , 839 , 842 ,
	1788 , 1792 , 1798 , 1806 , 1843 , 1884	845 , 848 , 851 , 854 , 881 , 911 , 915 , 944
\l_zrefclever_range_inhibit_- 34 , 35 , 1319 , 1752	\l_zrefclever_setup_type_tl . . .
next_bool 35 , 607 , 613 , 636 , 654 , 666 ,
\l_zrefclever_range_same_count_- 1319 , 1355 , 1514 , 1546 , 1737 , 1776 ,	676 , 679 , 682 , 685 , 688 , 691 , 694 ,
int	1788 , 1792 , 1798 , 1806 , 1843 , 1884	697 , 758 , 763 , 772 , 780 , 784 , 793 ,
 35 ,	812 , 824 , 834 , 837 , 840 , 843 , 846 ,
 1319 , 1355 , 1514 , 1546 , 1737 , 1776 ,	849 , 852 , 855 , 879 , 909 , 916 , 938 , 945
	1788 , 1792 , 1798 , 1806 , 1843 , 1884	\l_zrefclever_sort_decided_bool
 35 , 1043 , 1097 , 1101 , 1120 , 1131 ,
 1319 , 1355 , 1514 , 1546 , 1737 , 1776 ,	1148 , 1154 , 1171 , 1177 , 1205 , 1217
	1788 , 1792 , 1798 , 1806 , 1843 , 1884	_zrefclever_sort_default_-
 35 ,	aux:nn 27 , 29 , 1025 , 1045
 1319 , 1355 , 1514 , 1546 , 1737 , 1776 ,	_zrefclever_sort_labels:
	1788 , 1792 , 1798 , 1806 , 1843 , 1884 28 , 29 , 34 , 963 , 987
 35 ,	_zrefclever_sort_page_aux:nn . .
 1319 , 1355 , 1514 , 1546 , 1737 , 1776 , 34 , 1024 , 1302
	1788 , 1792 , 1798 , 1806 , 1843 , 1884	\l_zrefclever_sort_prior_a_int .
 35 , 976 , 1227 , 1237 , 1238 , 1244 , 1254 , 1262
 1319 , 1355 , 1514 , 1546 , 1737 , 1776 ,	\l_zrefclever_sort_prior_b_int .
	1788 , 1792 , 1798 , 1806 , 1843 , 1884 977 , 1228 , 1239 , 1240 , 1247 , 1255 , 1263
 35 ,	\l_zrefclever_tlastsep_tl
 1319 , 1355 , 1514 , 1546 , 1737 , 1776 , 1337 , 1361 , 1723
	1788 , 1792 , 1798 , 1806 , 1843 , 1884	

<code>\l_zrefclever_tlistsep_tl</code>	<code>\l_zrefclever_typeset_queue_-</code>
. 1336 , 1360 , 1701	<code>curr_tl</code> 1313 , 1349 , 1503 ,
<code>\l_zrefclever_tpairsep_tl</code>	1519 , 1528 , 1555 , 1565 , 1580 , 1601 ,
. 1335 , 1359 , 1717	1618 , 1635 , 1642 , 1649 , 1692 , 1713 ,
<code>\l_zrefclever_type_<type>_-</code>	1718 , 1724 , 1730 , 1731 , 1811 , 1822 ,
<code>options_prop</code> 19	1847 , 1858 , 1871 , 1961 , 2006 , 2010
<code>\l_zrefclever_type_count_int</code>	<code>\l_zrefclever_typeset_queue_-</code>
. 35 , 1317 , 1354 , 1698 ,	<code>prev_tl</code> 1313 , 1348 , 1702 , 1730
1700 , 1709 , 1736 , 1949 , 1956 , 2012	<code>\l_zrefclever_typeset_range_-</code>
<code>\l_zrefclever_type_first_label_-</code>	<code>bool</code> 364 , 367 , 962 , 1599
<code>tl</code> 1313 , 1350 , 1494 , 1603 ,	<code>\l_zrefclever_typeset_ref_bool</code> .
1612 , 1616 , 1643 , 1659 , 1662 , 1667 , 302 , 309 , 314 , 319 , 1633 , 1640
1673 , 1732 , 1767 , 1939 , 2021 , 2027 ,	<code>_zrefclever_typeset_refs:</code>
2033 , 2035 , 2039 , 2044 , 2059 , 2100 , 35 , 36 , 47-49 , 964 , 1346
2117 , 2119 , 2123 , 2128 , 2133 , 2148	<code>_zrefclever_typeset_refs_aux_-</code>
<code>\l_zrefclever_type_first_label_-</code>	<code>last_of_type:</code> 1477 , 1485
<code>type_tl</code>	<code>_zrefclever_typeset_refs_aux_-</code>
. 1313 , 1351 , 1495 , 1607 , 1733 ,	<code>not_last_of_type:</code> 1481 , 1741
1768 , 1942 , 1966 , 1973 , 1980 , 1987	<code>\l_zrefclever_typeset_sort_bool</code>
<code>_zrefclever_type_name_setup:</code> 328 , 331 , 961
. 36 , 1631 , 1937	<code>\l_zrefclever_typesort_seq</code>
<code>\l_zrefclever_type_name_tl</code> 337 , 342 , 346 , 352 , 1233
. 48 , 1343 ,	<code>\l_zrefclever_use_hyperref_bool</code>
1675 , 1681 , 1940 , 1943 , 1968 , 1977 , 371 , 398 , 404 , 1896 , 1994 , 2104
1985 , 1996 , 2050 , 2080 , 2087 , 2095	<code>\l_zrefclever_warn_hyperref_-</code>
<code>\l_zrefclever_typeset_compress_-</code>	<code>bool</code> 371 , 402
<code>bool</code> 355 , 358 , 1749	<code>_zrefclever_zcref:nnn</code> 951 , 954
<code>\l_zrefclever_typeset_last_bool</code>	<code>_zrefclever_zcref:nnnn</code> . 26 , 27 , 954
. 35 , 1311 ,	<code>\l_zrefclever_zcref_labels_seq</code> .
1368 , 1369 , 1375 , 1401 , 1706 , 2011	27 , 952 , 958 , 993 , 995 , 1371 , 1372 , 1377
<code>\l_zrefclever_typeset_name_bool</code>	<code>\l_zrefclever_zcref_note_tl</code>
. 303 , 310 , 315 , 320 , 1633 , 1647 560 , 563 , 967