

The zref-clever package implementation*

Gustavo Barros[†]

2021-09-29

Contents

1	Initial setup	2
2	Dependencies	3
3	zref setup	3
4	Plumbing	7
4.1	Messages	7
4.2	Data extraction	10
4.3	Reference format	11
4.4	Languages	12
4.5	Dictionaries	14
4.6	Options	21
5	Configuration	38
5.1	\zcsetup	38
5.2	\zcRefTypeSetup	38
5.3	\zcLanguageSetup	40
6	User interface	44
6.1	\zceref	44
6.2	\zcpageref	45
7	Sorting	46
8	Typesetting	53

*This file describes v0.1.0-alpha, released 2021-09-29.

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

9	Compatibility	79
9.1	<code>\footnote</code>	80
9.2	<code>\appendix</code>	80
9.3	<code>appendix</code> package	81
9.4	<code>amsmath</code> package	82
9.5	<code>mathtools</code> package	85
9.6	<code>breqn</code> package	86
9.7	<code>listings</code> package	87
9.8	<code>enumitem</code> package	87
10	Dictionaries	88
10.1	English	88
10.2	German	92
10.3	French	100
10.4	Portuguese	104
10.5	Spanish	108
Index		112

1 Initial setup

Start the DocStrip guards.

```
1 <*package>
```

Identify the internal prefix (L^AT_EX3 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 `expl3` as well (which made to the kernel in the 2020-02-02 release). We also just use UTF-8 for the dictionaries (which became the default input encoding in the 2018-04-01 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.

CHECK Should I just go ahead and bump this to 2021-11-15 considering the appendix case?

```
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} {2021-09-29} {0.1.0-alpha}  
15 {Clever LaTeX cross-references based on zref}
```

2 Dependencies

Required packages. Besides these, `zref-hyperref` may also be required depending on the presence of `hyperref` itself and on the `hyperref` option.

```
16 \RequirePackage { zref-base }  
17 \RequirePackage { zref-user }  
18 \RequirePackage { zref-abspage }  
19 \RequirePackage { l3keys2e }  
20 \RequirePackage { ifdraft }
```

3 zref setup

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

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

The `counter` property, in most cases, will be just the kernel's `\@currentcounter`, set by `\refstepcounter`. However, not everywhere is it assured that `\@currentcounter` gets updated as it should, so we need to have some means to manually tell `zref-clever` what the current counter actually is. This is done with the `currentcounter` option, and stored in `\l__zrefclever_current_counter_tl`, whose default is `\@currentcounter`.

```
21 \zref@newprop { zc@counter } { \l__zrefclever_current_counter_tl }  
22 \zref@addprop \ZREF@mainlist { zc@counter }
```

The reference itself, stored by `zref-base` in the `default` property, is somewhat a disputed real estate. In particular, the use of `\labelformat` (previously from `varioref`, now in the kernel) will include there the reference “prefix” and complicate the job we are trying to do here. Hence, we isolate `\the<counter>` and store it “clean” in `zc@thecnt` for reserved use. Since `\@currentlabel`, which populates the `default` property, is *more reliable* than `\@currentcounter`, `zc@thecnt` is meant to be kept as an *option* (`ref` option), in case there's need to use `zref-clever` together with `\labelformat`. Based on the definition of `\@currentlabel` done inside `\refstepcounter` in ‘texdoc source2e’, section ‘ltxref.dtx’. We just drop the `\p@...` prefix.

```
23 \zref@newprop { zc@thecnt }  
24 {  
25   \cs_if_exist:cTF { c@ \l__zrefclever_current_counter_tl }  
26   { \use:c { the \l__zrefclever_current_counter_tl } }  
27   {  
28     \cs_if_exist:cT { c@ \@currentcounter }  
29     { \use:c { the \@currentcounter } }  
30   }  
31 }  
32 \zref@addprop \ZREF@mainlist { zc@thecnt }
```

Much of the work of `zref-clever` relies on the association between a label’s “counter” and its “type” (see the User manual section on “Reference types”). Superficially examined, one might think this relation could just be stored in a global property list, rather than in the label itself. However, there are cases in which we want to distinguish different types for the same counter, depending on the document context. Hence, we need to store the “type” of the “counter” for each “label”. In setting this, the presumption is that the label’s type has the same name as its counter, unless it is specified otherwise by the `countertype` option, as stored in `\l__zrefclever_counter_type_prop`.

```

33 \zref@newprop { zc@type }
34 {
35   \exp_args:NNe \prop_if_in:NnTF \l__zrefclever_counter_type_prop
36   \l__zrefclever_current_counter_tl
37   {
38     \exp_args:NNe \prop_item:Nn \l__zrefclever_counter_type_prop
39     { \l__zrefclever_current_counter_tl }
40   }
41   { \l__zrefclever_current_counter_tl }
42 }
43 \zref@addprop \ZREF@mainlist { zc@type }

```

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

```

44 \zref@newprop { zc@cntval } [0]
45 {
46   \cs_if_exist:cTF { c@ \l__zrefclever_current_counter_tl }
47   { \int_use:c { c@ \l__zrefclever_current_counter_tl } }
48   {
49     \cs_if_exist:cT { c@ \@currentcounter }
50     { \int_use:c { c@ \@currentcounter } }
51   }
52 }
53 \zref@addprop \ZREF@mainlist { zc@cntval }
54 \zref@newprop* { zc@pgval } [0] { \int_use:c { c@page } }
55 \zref@addprop \ZREF@mainlist { zc@pgval }

```

However, since many counters (may) get reset along the document, we require more than just their numeric values. We need to know the reset chain of a given counter, in order to sort and compress a group of references. Also here, the “printed representation” is not enough, not only because it is easier to work with the numeric values but, given we occasionally group multiple counters within a single type, sorting this group requires to know the actual counter reset chain (the counters’ names and values). Indeed, the set of counters grouped into a single type cannot be arbitrary: all of them must belong to the same reset chain, and must be nested within each other (they cannot even just share the same parent).

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

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

The procedure below examines a set of counters, those included in `\l__zrefclever_counter_resettters_seq`, and for each of them retrieves the set of counters it resets, as stored in `\cl@<counter>`, looking for the counter for which we are trying to set a label (`\l__zrefclever_current_counter_tl`, by default `\@currentcounter`, passed as an argument to the functions). There is one relevant caveat to this procedure: `\l__zrefclever_counter_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, with the option `counterresetters`. Unfortunately, not all counters are created alike, or reset alike. Some counters, even some kernel ones, get reset by other mechanisms (notably, the `enumerate` environment counters do not use the regular counter machinery for resetting on each level, but are nested nevertheless by other means). Therefore, inspecting `\cl@<counter>` cannot possibly fully account for all of the automatic counter resetting which takes place in the document. And there’s also no other “general rule” we could grab on for this, as far as I know. So we provide a way to manually tell `zref-clever` of these cases, by means of the `counterresetby` option, whose information is stored in `\l__zrefclever_counter_resetby_prop`. This manual specification has precedence over the search through `\l__zrefclever_counter_resettters_seq`, and should be handled with care, since there is no possible verification mechanism for this.

`__zrefclever_get_enclosing_counters_value:n`

Recursively generate a *sequence* of “enclosing counters” values, for a given `<counter>` and leave it in the input stream. 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.

```

\__zrefclever_get_enclosing_counters_value:n {<counter>}

56 \cs_new:Npn \__zrefclever_get_enclosing_counters_value:n #1
57 {
58   \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
59   {
60     { \int_use:c { c@ \__zrefclever_counter_reset_by:n {#1} } }

```

```

61     \_zrefclever_get_enclosing_counters_value:e
62     { \_zrefclever_counter_reset_by:n {#1} }
63   }
64 }

```

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

```

65 \cs_generate_variant:Nn \_zrefclever_get_enclosing_counters_value:n { e }

```

(End definition for `_zrefclever_get_enclosing_counters_value:n`.)

`_zrefclever_counter_reset_by:n`

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

```

    \_zrefclever_counter_reset_by:n {<counter>}
66 \cs_new:Npn \_zrefclever_counter_reset_by:n #1
67 {
68   \bool_if:nTF
69   { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} }
70   { \prop_item:Nn \l__zrefclever_counter_resetby_prop {#1} }
71   {
72     \seq_map_tokens:Nn \l__zrefclever_counter_resettors_seq
73     { \_zrefclever_counter_reset_by_aux:nn {#1} }
74   }
75 }
76 \cs_new:Npn \_zrefclever_counter_reset_by_aux:nn #1#2
77 {
78   \cs_if_exist:cT { c@ #2 }
79   {
80     \tl_if_empty:cF { c1@ #2 }
81     {
82       \tl_map_tokens:cn { c1@ #2 }
83       { \_zrefclever_counter_reset_by_auxi:nnn {#2} {#1} }
84     }
85   }
86 }
87 \cs_new:Npn \_zrefclever_counter_reset_by_auxi:nnn #1#2#3
88 {
89   \str_if_eq:nnT {#2} {#3}
90   { \tl_map_break:n { \seq_map_break:n {#1} } }
91 }

```

(End definition for `_zrefclever_counter_reset_by:n`.)

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

```

92 \zref@newprop { zc@enclval }
93 {
94   \_zrefclever_get_enclosing_counters_value:e
95   \l__zrefclever_current_counter_tl
96 }
97 \zref@addprop \ZREF@mainlist { zc@enclval }

```

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

```

98 \tl_new:N \g__zrefclever_page_format_tl
99 \cs_new_protected:Npx \__zrefclever_page_format_aux: { \int_eval:n { 1 } }
100 \AddToHook { shipout / before }
101 {
102   \group_begin:
103   \cs_set_eq:NN \c@page \__zrefclever_page_format_aux:
104   \tl_gset:Nx \g__zrefclever_page_format_tl { \thepage }
105   \group_end:
106 }
107 \zref@newprop* { zc@pgfmt } { \g__zrefclever_page_format_tl }
108 \zref@addprop \ZREF@mainlist { zc@pgfmt }

```

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

4 Plumbing

4.1 Messages

```

109 \msg_new:nnn { zref-clever } { option-not-type-specific }
110 {
111   Option~'#1'~is-not-type-specific~\msg_line_context:~
112   Set-it-in~'\iow_char:N\zcLanguageSetup'~before-first~'type'
113   ~switch-or-as-package-option.
114 }
115 \msg_new:nnn { zref-clever } { option-only-type-specific }
116 {
117   No-type-specified-for-option~'#1'~\msg_line_context:~
118   Set-it-after~'type'~switch-or-in~'\iow_char:N\zcRefTypeSetup'.

```

```

119 }
120 \msg_new:nnn { zref-clever } { key-requires-value }
121 { The~'#1'~key~'#2'~requires~a~value~\msg_line_context:.. }
122 \msg_new:nnn { zref-clever } { language-declared }
123 { Language~'#1'~is~already~declared~\msg_line_context:..Nothing~to~do. }
124 \msg_new:nnn { zref-clever } { unknown-language-alias }
125 {
126   Language~'#1'~is~unknown~\msg_line_context:..Can't~alias~to~it.~
127   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
128   '\iow_char:N\zcDeclareLanguageAlias'.
129 }
130 \msg_new:nnn { zref-clever } { unknown-language-setup }
131 {
132   Language~'#1'~is~unknown~\msg_line_context:..Can't~set~it~up.~
133   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
134   '\iow_char:N\zcDeclareLanguageAlias'.
135 }
136 \msg_new:nnn { zref-clever } { unknown-language-opt }
137 {
138   Language~'#1'~is~unknown~\msg_line_context:..Using~default.~
139   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
140   '\iow_char:N\zcDeclareLanguageAlias'.
141 }
142 \msg_new:nnn { zref-clever } { unknown-language-decl }
143 {
144   Can't~set~declension~'#1'~for~unknown~language~'#2'~\msg_line_context:..
145   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
146   '\iow_char:N\zcDeclareLanguageAlias'.
147 }
148 \msg_new:nnn { zref-clever } { language-no-decl-ref }
149 {
150   Language~'#1'~has~no~declared~declension~cases~\msg_line_context:..
151   Nothing~to~do~with~option~'d=#2'.
152 }
153 \msg_new:nnn { zref-clever } { language-no-gender }
154 {
155   Language~'#1'~has~no~declared~gender~\msg_line_context:..
156   Nothing~to~do~with~option~'#2=#3'.
157 }
158 \msg_new:nnn { zref-clever } { language-no-decl-setup }
159 {
160   Language~'#1'~has~no~declared~declension~cases~\msg_line_context:..
161   Nothing~to~do~with~option~'case=#2'.
162 }
163 \msg_new:nnn { zref-clever } { unknown-decl-case }
164 {
165   Declension~case~'#1'~unknown~for~language~'#2'~\msg_line_context:..
166   Using~default~declension~case.
167 }
168 \msg_new:nnn { zref-clever } { nudge-multitype }
169 {
170   Reference~with~multiple~types~\msg_line_context:..
171   You~may~wish~to~separate~them~or~review~language~around~it.
172 }

```



```

173 \msg_new:nnn { zref-clever } { nudge-comptosing }
174 {
175   Multiple~labels~have~been~compressed~into~singular~type~name~
176   for~type~'#1'~\msg_line_context:.
177 }
178 \msg_new:nnn { zref-clever } { nudge-plural-when-sg }
179 {
180   Option~'sg'~signals~that~a~singular~type~name~was~expected~
181   \msg_line_context:.~But~type~'#1'~has~plural~type~name.
182 }
183 \msg_new:nnn { zref-clever } { gender-not-declared }
184 { Language~'#1'~has~no~'#2'~gender~declared~\msg_line_context:. }
185 \msg_new:nnn { zref-clever } { nudge-gender-mismatch }
186 {
187   Gender~mismatch~for~type~'#1'~\msg_line_context:.~
188   You've~specified~'g=#2'~but~type~name~is~'#3'~for~language~'#4'.
189 }
190 \msg_new:nnn { zref-clever } { nudge-gender-not-declared-for-type }
191 {
192   You've~specified~'g=#1'~\msg_line_context:.~
193   But~gender~for~type~'#2'~is~not~declared~for~language~'#3'.
194 }
195 \msg_new:nnn { zref-clever } { nudgeif-unknown-value }
196 { Unknown~value~'#1'~for~'nudgeif'~option~\msg_line_context:. }
197 \msg_new:nnn { zref-clever } { option-document-only }
198 { Option~'#1'~is~only~available~after~\iow_char:N\begin\{document\}. }
199 \msg_new:nnn { zref-clever } { dict-loaded }
200 { Loaded~'#1'~dictionary. }
201 \msg_new:nnn { zref-clever } { dict-not-available }
202 { Dictionary~for~'#1'~not~available~\msg_line_context:. }
203 \msg_new:nnn { zref-clever } { unknown-language-load }
204 {
205   Language~'#1'~is~unknown~\msg_line_context:.~Unable~to~load~dictionary.~
206   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
207   '\iow_char:N\zcDeclareLanguageAlias'.
208 }
209 \msg_new:nnn { zref-clever } { missing-zref-titleref }
210 {
211   Option~'ref=title'~requested~\msg_line_context:.~
212   But~package~'zref-titleref'~is~not~loaded,~falling~back~to~default~'ref'.
213 }
214 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
215 {
216   Option~'hyperref'~only~available~in~the~preamble~\msg_line_context:.~
217   Use~the~starred~version~of~'\iow_char:N\zcRef'~instead.
218 }
219 \msg_new:nnn { zref-clever } { missing-hyperref }
220 { Missing~'hyperref'~package.~Setting~'hyperref=false'. }
221 \msg_new:nnn { zref-clever } { titleref-preamble-only }
222 {
223   Option~'titleref'~only~available~in~the~preamble~\msg_line_context:.~
224   Did~you~mean~'ref=title'?
225 }
226 \msg_new:nnn { zref-clever } { missing-zref-check }

```

```

227 {
228   Option~'check'~requested~\msg_line_context:..~
229   But~package~'zref-clever'~is~not~loaded,~can't~run~the~checks.
230 }
231 \msg_new:nnn { zref-clever } { missing-type }
232 { Reference-type-undefined-for-label~'#1'~\msg_line_context:. }
233 \msg_new:nnn { zref-clever } { missing-name }
234 { Reference-format-option~'#1'~undefined-for-type~'#2'~\msg_line_context:. }
235 \msg_new:nnn { zref-clever } { missing-string }
236 {
237   We~couldn't~find~a~value~for~reference-option~'#1'~\msg_line_context:..~
238   But~we~should~have:~throw~a~rock~at~the~maintainer.
239 }
240 \msg_new:nnn { zref-clever } { single-element-range }
241 { Range-for-type~'#1'~resulted~in~single-element~\msg_line_context:. }
242 \msg_new:nnn { zref-clever } { compat-package }
243 { Loaded-support-for~'#1'~package. }
244 \msg_new:nnn { zref-clever } { compat-class }
245 { Loaded-support-for~'#1'~documentclass. }

```

4.2 Data extraction

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

```

\__zrefclever_def_extract:Nnnn {(tl val)}
  {(label)} {(prop)} {(default)}
246 \cs_new_protected:Npn \__zrefclever_def_extract:Nnnn #1#2#3#4
247 {
248   \exp_args:NNNo \exp_args:NNo \tl_set:Nn #1
249   { \zref@extractdefault {#2} {#3} {#4} }
250 }
251 \cs_generate_variant:Nn \__zrefclever_def_extract:Nnnn { NVnn }

```

(End definition for `_zrefclever_def_extract:Nnnn`.)

`_zrefclever_extract_unexp:nnn` Extract property $\langle prop \rangle$ from $\langle label \rangle$. Ensure that, in the context of an x expansion, `\zref@extractdefault` is expanded exactly twice, but no further to retrieve the proper value. Thus, this is meant to be use in an x expansion context, not in other situations. In case the property is not found, leave $\langle default \rangle$ in the stream.

```

\__zrefclever_extract_unexp:nnn{(label)}{(prop)}{(default)}
252 \cs_new:Npn \__zrefclever_extract_unexp:nnn #1#2#3
253 {
254   \exp_args:NNNo \exp_args:No
255   \exp_not:n { \zref@extractdefault {#1} {#2} {#3} }
256 }
257 \cs_generate_variant:Nn \__zrefclever_extract_unexp:nnn { Vnn , nvnn , Vvn }

```

(End definition for `_zrefclever_extract_unexp:nnn`.)

`_zrefclever_extract:nnn` An internal version for `\zref@extractdefault`.

```

    \__zrefclever_extract:nnn{\label}\{<prop>\}\{<default>\}

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

(End definition for \__zrefclever_extract:nnn.)

```

4.3 Reference format

For a general discussion on the precedence rules for reference format options, see Section “Reference format” in the User manual. Internally, these precedence rules are handled / enforced in `__zrefclever_get_ref_string:nN`, `__zrefclever_get_ref_font:nN`, and `__zrefclever_type_name_setup:` which are the basic functions to retrieve proper values for reference format settings. The “fallback” settings are stored in `\g__zrefclever_fallback_dict_prop`.

```

\l__zrefclever_setup_type_tl Store “current” type, language, and declension cases in different places for option and
    \l__zrefclever_dict_language_tl translation handling, notably in \__zrefclever_provide_dictionary:n, \zcRefTypeSetup,
    \l__zrefclever_dict_decl_case_tl and \zcLanguageSetup. But also for translations retrieval, in \__zrefclever_get_
    \l__zrefclever_dict_declension_seq type_transl:nnnN and \__zrefclever_get_default_transl:nnN.
    \l__zrefclever_dict_gender_seq

260 \tl_new:N \l__zrefclever_setup_type_tl
261 \tl_new:N \l__zrefclever_dict_language_tl
262 \tl_new:N \l__zrefclever_dict_decl_case_tl
263 \seq_new:N \l__zrefclever_dict_declension_seq
264 \seq_new:N \l__zrefclever_dict_gender_seq

(End definition for \l__zrefclever_setup_type_tl and others.)

```

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

```

\c__zrefclever_ref_options_type_names_seq
\c__zrefclever_ref_options_genders_seq
\c__zrefclever_ref_options_font_seq
\c__zrefclever_ref_options_typesetup_seq
\c__zrefclever_ref_options_reference_seq

265 \seq_const_from_clist:Nn
266 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
267 {
268   tpairsep ,
269   tlistsep ,
270   tlastsep ,
271   notesep ,
272 }
273 \seq_const_from_clist:Nn
274 \c__zrefclever_ref_options_possibly_type_specific_seq
275 {
276   namesep ,
277   pairsep ,
278   listsep ,
279   lastsep ,
280   rangesep ,
281   refpre ,
282   refpos ,
283   refpre-in ,
284   refpos-in ,
285 }

```

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

```

286 \seq_const_from_clist:Nn
287   \c__zrefclever_ref_options_type_names_seq
288   {
289     Name-sg ,
290     name-sg ,
291     Name-pl ,
292     name-pl ,
293     Name-sg-ab ,
294     name-sg-ab ,
295     Name-pl-ab ,
296     name-pl-ab ,
297   }
298 \seq_const_from_clist:Nn
299   \c__zrefclever_ref_options_genders_seq
300   { f , m , n }

```

`\c__zrefclever_ref_options_font_seq` are technically “possibly type-specific”, but are not “language-specific”, so we separate them.

```

301 \seq_const_from_clist:Nn
302   \c__zrefclever_ref_options_font_seq
303   {
304     namefont ,
305     reffont ,
306     reffont-in ,
307   }

```

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

```

308 \seq_new:N \c__zrefclever_ref_options_typesetup_seq
309 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
310   \c__zrefclever_ref_options_possibly_type_specific_seq
311   \c__zrefclever_ref_options_type_names_seq
312 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
313   \c__zrefclever_ref_options_typesetup_seq
314   \c__zrefclever_ref_options_font_seq
315 \seq_new:N \c__zrefclever_ref_options_reference_seq
316 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
317   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
318   \c__zrefclever_ref_options_possibly_type_specific_seq
319 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
320   \c__zrefclever_ref_options_reference_seq
321   \c__zrefclever_ref_options_font_seq

```

(End definition for `\c__zrefclever_ref_options_necessarily_not_type_specific_seq` and others.)

4.4 Languages

`\g__zrefclever_languages_prop` Stores the names of known languages and the mapping from “language name” to “dictionary name”. Whether of not a language or alias is known to `zref-clever` is decided by its presence in this property list. A “base language” (loose concept here, meaning just “the name we gave for the dictionary in that particular language”) is just like any other one,

the only difference is that the “language name” happens to be the same as the “dictionary name”, in other words, it is an “alias to itself”.

```
322 \prop_new:N \g__zrefclever_languages_prop
```

(End definition for \g__zrefclever_languages_prop.)

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

```

\zcDeclareLanguage [ $\langle options \rangle$ ] { $\langle language \rangle$ }

323 \NewDocumentCommand \zcDeclareLanguage { 0 { } m }
324 {
325   \group_begin:
326   \tl_if_empty:nF {#2}
327   {
328     \prop_if_in:NnTF \g__zrefclever_languages_prop {#2}
329     { \msg_warning:nnn { zref-clever } { language-declared } {#2} }
330     {
331       \prop_gput:Nnn \g__zrefclever_languages_prop {#2} {#2}
332       \prop_new:c { g__zrefclever_dict_ #2 _prop }
333       \tl_set:Nn \l__zrefclever_dict_language_tl {#2}
334       \keys_set:nn { zref-clever / declarelang } {#1}
335     }
336   }
337   \group_end:
338 }
339 \@onlypreamble \zcDeclareLanguage

340 \keys_define:nn { zref-clever / declarelang }
341 {
342   declension .code:n =
343   {
344     \prop_gput:cnn
345     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
346     { declension } {#1}
347   } ,
348   declension .value_required:n = true ,
349   gender .code:n =
350   {
351     \prop_gput:cnn
352     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
353     { gender } {#1}
354   } ,
355   gender .value_required:n = true ,

```

```

356     allcaps .code:n =
357     {
358         \prop_gput:cnn
359         { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
360         { allcaps } { true }
361     } ,
362     allcaps .value_forbidden:n = true ,
363 }

```

(End definition for `\zcDeclareLanguage`.)

`\zcDeclareLanguageAlias` Declare $\langle language\ alias \rangle$ to be an alias of $\langle aliased\ language \rangle$. $\langle aliased\ language \rangle$ must be already known to `zref-clever`, as stored in `\g__zrefclever_languages_prop`. `\zcDeclareLanguageAlias` is preamble only.

```

\zcDeclareLanguageAlias {\langle language\ alias \rangle} {\langle aliased\ language \rangle}

364 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
365 {
366     \tl_if_empty:nF {#1}
367     {
368         \prop_if_in:NnTF \g__zrefclever_languages_prop {#2}
369         {
370             \exp_args:NnNx
371             \prop_gput:Nnn \g__zrefclever_languages_prop {#1}
372             { \prop_item:Nn \g__zrefclever_languages_prop {#2} }
373         }
374         { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
375     }
376 }
377 \@onlypreamble \zcDeclareLanguageAlias

```

(End definition for `\zcDeclareLanguageAlias`.)

4.5 Dictionaries

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

The built-in dictionaries and their related infrastructure are designed to perform “on the fly” loading of dictionaries, “lazily” as needed. Much like `babel` does for languages not declared in the preamble, but used in the document. This offers some convenience, of course, and that’s one reason to do it. But it also has the purpose of parsimony, of “loading the least possible”. My expectation is that for most use cases, users will require a single language of the functionality of `zref-clever` – the main language of the document –, even in multilingual documents. Hence, even the set of `babel` or `polyglossia` “loaded languages”, which would be the most tenable set if loading were restricted to the preamble, is bound to be an overshoot in typical cases. Therefore, we load at `\begin{document}` one single language (see [lang option](#)), as specified by the user in the preamble with the `lang` option or, failing any specification, the main language of the document, which is the default. Anything else is lazily loaded, on the fly, along the document.

This design decision has also implications to the *form* the dictionary files assumed. As far as my somewhat impressionistic sampling goes, dictionary or localization files of the

most common packages in this area of functionality, are usually a set of commands which perform the relevant definitions and assignments in the preamble or at `\begin{document}`. This includes `translator`, `translations`, but also `babel`'s `.ldf` files, and `biblatex`'s `.lbx` files. I'm not really well acquainted with this machinery, but as far as I grasp, they all rely on some variation of `\ProvidesFile` and `\input`. And they can be safely `\input` without generating spurious content, because they rely on being loaded before the document has actually started. As far as I can tell, `babel`'s "on the fly" functionality is not based on the `.ldf` files, but on the `.ini` files, and on `\babelprovide`. And the `.ini` files are not in this form, but actually resemble "configuration files" of sorts, which means they are read and processed somehow else than with just `\input`. So we do the more or less the same here. It seems a reasonable way to ensure we can load dictionaries on the fly robustly mid-document, without getting paranoid with the last bit of white-space in them, and without introducing any undue content on the stream when we cannot afford to do it. Hence, `zref-clever`'s built-in dictionary files are a set of *key-value options* which are read from the file, and fed to `\keys_set:nn{zref-clever/dictionary}` by `__zrefclever_provide_dictionary:n`. And they use the same syntax and options as `\zcLanguageSetup` does. The dictionary file itself is read with `\ExplSyntaxOn` with the usual implications for white-space and catcodes.

`__zrefclever_provide_dictionary:n` is only meant to load the built-in dictionaries. For languages declared by the user, or for any settings to a known language made with `\zcLanguageSetup`, values are populated directly to a variable `\g__zrefclever_dict_{language}_prop`, created as needed. Hence, there is no need to "load" anything in this case: definitions and assignments made by the user are performed immediately.

Provide

`\g__zrefclever_loaded_dictionaries_seq` Used to keep track of whether a dictionary has already been loaded or not.

```
378 \seq_new:N \g__zrefclever_loaded_dictionaries_seq
```

(End definition for `\g__zrefclever_loaded_dictionaries_seq`.)

`\l__zrefclever_load_dict_verbose_bool` Controls whether `__zrefclever_provide_dictionary:n` fails silently or verbosely in case of unknown languages or dictionaries not found.

```
379 \bool_new:N \l__zrefclever_load_dict_verbose_bool
```

(End definition for `\l__zrefclever_load_dict_verbose_bool`.)

`__zrefclever_provide_dictionary:n` Load dictionary for known `\langle language \rangle` if it is available and if it has not already been loaded.

```

\__zrefclever_provide_dictionary:n {\langle language \rangle}

380 \cs_new_protected:Npn \__zrefclever_provide_dictionary:n #1
381 {
382   \group_begin:
383   \@bsphack
384   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
385   \l__zrefclever_dict_language_tl
386   {
387     \seq_if_in:NVF
388     \g__zrefclever_loaded_dictionaries_seq
389     \l__zrefclever_dict_language_tl
390     {

```

```

391 \exp_args:Nx \file_get:nnNTF
392 { zref-clever- \l__zrefclever_dict_language_tl .dict }
393 { \ExplSyntaxOn }
394 \l_tmpa_tl
395 {
396   \tl_clear:N \l__zrefclever_setup_type_tl
397   \exp_args:NNx \seq_set_from_clist:Nn
398     \l__zrefclever_dict_declension_seq
399     {
400       \prop_item:cn
401       {
402         g__zrefclever_dict_
403         \l__zrefclever_dict_language_tl _prop
404       }
405       { declension }
406     }
407   \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
408     { \tl_clear:N \l__zrefclever_dict_decl_case_tl }
409     {
410       \seq_get_left:NN \l__zrefclever_dict_declension_seq
411       \l__zrefclever_dict_decl_case_tl
412     }
413   \exp_args:NNx \seq_set_from_clist:Nn
414     \l__zrefclever_dict_gender_seq
415     {
416       \prop_item:cn
417       {
418         g__zrefclever_dict_
419         \l__zrefclever_dict_language_tl _prop
420       }
421       { gender }
422     }
423   \keys_set:nV { zref-clever / dictionary } \l_tmpa_tl
424   \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
425     \l__zrefclever_dict_language_tl
426   \msg_note:nxx { zref-clever } { dict-loaded }
427     { \l__zrefclever_dict_language_tl }
428 }
429 {
430   \bool_if:NT \l__zrefclever_load_dict_verbose_bool
431   {
432     \msg_warning:nxx { zref-clever } { dict-not-available }
433     { \l__zrefclever_dict_language_tl }
434   }

```

Even if we don't have the actual dictionary, we register it as “loaded”. At this point, it is a known language, properly declared. There is no point in trying to load it multiple times, because users cannot really provide the dictionary files (well, technically they could, but we are working so they don't need to, and have better ways to do what they want). And if the users had provided some translations themselves, by means of `\zcLanguageSetup`, everything would be in place, and they could use the `lang` option multiple times, and the `dict-not-available` warning would never go away.

```

435   \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
436     \l__zrefclever_dict_language_tl

```



```

437         }
438     }
439 }
440 {
441     \bool_if:NT \l__zrefclever_load_dict_verbose_bool
442     { \msg_warning:nnn { zref-clever } { unknown-language-load } {#1} }
443 }
444 \@esphack
445 \group_end:
446 }
447 \cs_generate_variant:Nn \__zrefclever_provide_dictionary:n { x }

```

(End definition for __zrefclever_provide_dictionary:n.)

__zrefclever_provide_dictionary_verbose:n Does the same as __zrefclever_provide_dictionary:n, but warns if the loading of the dictionary has failed.

```

\__zrefclever_provide_dictionary_verbose:n {<language>}

448 \cs_new_protected:Npn \__zrefclever_provide_dictionary_verbose:n #1
449 {
450     \group_begin:
451     \bool_set_true:N \l__zrefclever_load_dict_verbose_bool
452     \__zrefclever_provide_dictionary:n {#1}
453     \group_end:
454 }
455 \cs_generate_variant:Nn \__zrefclever_provide_dictionary_verbose:n { x }

```

(End definition for __zrefclever_provide_dictionary_verbose:n.)

__zrefclever_provide_dict_type_transl:nn A couple of auxiliary functions for the of zref-clever/dictionary keys set in __zrefclever_provide_dictionary:n. They respectively “provide” (i.e. set if it value does not exist, do nothing if it already does) “type-specific” and “default” translations. Both receive <key> and <translation> as arguments, but __zrefclever_provide_dict_type_transl:nn relies on the current value of \l__zrefclever_setup_type_tl, as set by the type key.

```

\__zrefclever_provide_dict_type_transl:nn {<key>} {<translation>}
\__zrefclever_provide_dict_default_transl:nn {<key>} {<translation>}

456 \cs_new_protected:Npn \__zrefclever_provide_dict_type_transl:nn #1#2
457 {
458     \exp_args:Nnx \prop_gput_if_new:cnn
459     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
460     { type- \l__zrefclever_setup_type_tl - #1 } {#2}
461 }
462 \cs_new_protected:Npn \__zrefclever_provide_dict_default_transl:nn #1#2
463 {
464     \prop_gput_if_new:cnn
465     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
466     { default- #1 } {#2}
467 }

```

(End definition for __zrefclever_provide_dict_type_transl:nn and __zrefclever_provide_dict_default_transl:nn.)

The set of keys for `zref-clever/dictionary`, which is used to process the dictionary files in `_zrefclever_provide_dictionary:n`. The no-op cases for each category have their messages sent to “info”. These messages should not occur, as long as the dictionaries are well formed, but they’re placed there nevertheless, and can be leveraged in regression tests.

```

468 \keys_define:nn { zref-clever / dictionary }
469 {
470   type .code:n =
471   {
472     \tl_if_empty:NTF {#1}
473     { \tl_clear:N \l__zrefclever_setup_type_tl }
474     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
475   } ,
476   case .code:n =
477   {
478     \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
479     {
480       \msg_info:nxxx { zref-clever } { language-no-decl-setup }
481       { \l__zrefclever_dict_language_tl } {#1}
482     }
483     {
484       \seq_if_in:NnTF \l__zrefclever_dict_declension_seq {#1}
485       { \tl_set:Nn \l__zrefclever_dict_decl_case_tl {#1} }
486       {
487         \msg_info:nxxx { zref-clever } { unknown-decl-case }
488         {#1} { \l__zrefclever_dict_language_tl }
489         \seq_get_left:NN \l__zrefclever_dict_declension_seq
490         \l__zrefclever_dict_decl_case_tl
491       }
492     }
493   } ,
494   case .value_required:n = true ,
495   gender .code:n =
496   {
497     \seq_if_empty:NTF \l__zrefclever_dict_gender_seq
498     {
499       \msg_info:nxxxx { zref-clever } { language-no-gender }
500       { \l__zrefclever_dict_language_tl } { gender } {#1}
501     }
502     {
503       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
504       {
505         \msg_info:nnn { zref-clever }
506         { option-only-type-specific } { gender }
507       }
508       {
509         \seq_if_in:NnTF \l__zrefclever_dict_gender_seq {#1}
510         { \_zrefclever_provide_dict_type_transl:nn { gender } {#1} }
511         {
512           \msg_info:nxxx { zref-clever } { gender-not-declared }
513           { \l__zrefclever_dict_language_tl } {#1}
514         }
515       }
516     }
517   }

```

```

516     }
517   } ,
518   gender .value_required:n = true ,
519 }
520 \seq_map_inline:Nn
521 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
522 {
523   \keys_define:nn { zref-clever / dictionary }
524   {
525     #1 .value_required:n = true ,
526     #1 .code:n =
527     {
528       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
529       { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
530       {
531         \msg_info:nnn { zref-clever }
532         { option-not-type-specific } {#1}
533       }
534     } ,
535   }
536 }
537 \seq_map_inline:Nn
538 \c__zrefclever_ref_options_possibly_type_specific_seq
539 {
540   \keys_define:nn { zref-clever / dictionary }
541   {
542     #1 .value_required:n = true ,
543     #1 .code:n =
544     {
545       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
546       { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
547       { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
548     } ,
549   }
550 }
551 \seq_map_inline:Nn
552 \c__zrefclever_ref_options_type_names_seq
553 {
554   \keys_define:nn { zref-clever / dictionary }
555   {
556     #1 .value_required:n = true ,
557     #1 .code:n =
558     {
559       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
560       {
561         \msg_info:nnn { zref-clever }
562         { option-only-type-specific } {#1}
563       }
564       {
565         \tl_if_empty:NTF \l__zrefclever_dict_decl_case_tl
566         { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
567         {
568           \__zrefclever_provide_dict_type_transl:nn
569           { \l__zrefclever_dict_decl_case_tl - #1 } {##1}

```

```

570         }
571     } ,
572 } ,
573 }
574 }

```

Fallback

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

TODO Add regression test to ensure all fallback “translations” are indeed present.

```

575 \prop_new:N \g__zrefclever_fallback_dict_prop
576 \prop_gset_from_keyval:Nn \g__zrefclever_fallback_dict_prop
577 {
578     tpairsep = {,~} ,
579     tlistsep = {,~} ,
580     tlastsep = {,~} ,
581     notesep = {~} ,
582     namesep = {\nobreakspace} ,
583     pairsep = {,~} ,
584     listsep = {,~} ,
585     lastsep = {,~} ,
586     rangesep = {\textendash} ,
587     refpre = {} ,
588     refpos = {} ,
589     refpre-in = {} ,
590     refpos-in = {} ,
591 }

```

Get translations

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

```

    \_zrefclever_get_type_transl:nnnNF {\langle language \rangle} {\langle type \rangle} {\langle key \rangle}
    \langle tl variable \rangle {\langle false code \rangle}

592 \prg_new_protected_conditional:Npnn
593 \_zrefclever_get_type_transl:nnnNF #1#2#3#4 { F }
594 {
595     \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
596     \l__zrefclever_dict_language_tl
597     {

```

```

598     \prop_get:cnNTF
599     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
600     { type- #2 - #3 } #4
601     { \prg_return_true: }
602     { \prg_return_false: }
603   }
604   { \prg_return_false: }
605 }
606 \prg_generate_conditional_variant:Nnn
607   \__zrefclever_get_type_transl:nnnN { xxxN , xxnN } { F }

```

(End definition for __zrefclever_get_type_transl:nnnNF.)

_zrefclever_get_default_transl:nnNF Get default translation of $\langle key \rangle$ for $\langle language \rangle$, and store it in $\langle tl variable \rangle$ if found. If not found, leave the $\langle false code \rangle$ on the stream, in which case the value of $\langle tl variable \rangle$ should not be relied upon.

```

        \__zrefclever_get_default_transl:nnNF {<language>} {<key>}
        {<tl variable>} {<false code>}}

608 \prg_new_protected_conditional:Npnn
609   \__zrefclever_get_default_transl:nnN #1#2#3 { F }
610   {
611     \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
612     \l__zrefclever_dict_language_tl
613     {
614       \prop_get:cnNTF
615       { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
616       { default- #2 } #3
617       { \prg_return_true: }
618       { \prg_return_false: }
619     }
620     { \prg_return_false: }
621   }
622 \prg_generate_conditional_variant:Nnn
623   \__zrefclever_get_default_transl:nnN { xnN } { F }

```

(End definition for __zrefclever_get_default_transl:nnNF.)

_zrefclever_get_fallback_transl:nnNF Get fallback translation of $\langle key \rangle$, and store it in $\langle tl variable \rangle$ if found. If not found, leave the $\langle false code \rangle$ on the stream, in which case the value of $\langle tl variable \rangle$ should not be relied upon.

```

        \__zrefclever_get_fallback_transl:nnNF {<key>}
        {<tl variable>} {<false code>}}

624 % {<key>}<tl var to set>
625 \prg_new_protected_conditional:Npnn
626   \__zrefclever_get_fallback_transl:nnN #1#2 { F }
627   {
628     \prop_get:NnNTF \g__zrefclever_fallback_dict_prop
629     { #1 } #2
630     { \prg_return_true: }
631     { \prg_return_false: }
632   }

```

(End definition for __zrefclever_get_fallback_transl:nnNF.)

4.6 Options

Auxiliary

`_zrefclever_prop_put_non_empty:Nnn` If $\langle value \rangle$ is empty, remove $\langle key \rangle$ from $\langle property list \rangle$. Otherwise, add $\langle key \rangle = \langle value \rangle$ to $\langle property list \rangle$.

```

\__zrefclever_prop_put_non_empty:Nnn <property list> {<key>} {<value>}

633 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
634 {
635   \tl_if_empty:nTF {#3}
636     { \prop_remove:Nn #1 {#2} }
637     { \prop_put:Nnn #1 {#2} {#3} }
638 }
```

(End definition for `_zrefclever_prop_put_non_empty:Nnn`.)

ref option

`\l__zrefclever_ref_property_tl` stores the property to which the reference is being made. Currently, we restrict `ref=` to these three (or four) alternatives – `default`, `zc@thecnt`, `page`, and `title` if `zref-titleref` is loaded –, 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 current 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.

```

639 \tl_new:N \l__zrefclever_ref_property_tl
640 \keys_define:nn { zref-clever / reference }
641 {
642   ref .choice: ,
643   ref / default .code:n =
644     { \tl_set:Nn \l__zrefclever_ref_property_tl { default } } ,
645   ref / zc@thecnt .code:n =
646     { \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt } } ,
647   ref / page .code:n =
648     { \tl_set:Nn \l__zrefclever_ref_property_tl { page } } ,
649   ref / title .code:n =
650     {
651       \AddToHook { begindocument }
652       {
653         \@ifpackageloaded { zref-titleref }
654           { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
655           {
656             \msg_warning:nn { zref-clever } { missing-zref-titleref }
657             \tl_set:Nn \l__zrefclever_ref_property_tl { default }
658           }
659       }
660     } ,
661   ref .initial:n = default ,
```

```

662     ref .default:n = default ,
663     page .meta:n = { ref = page },
664     page .value_forbidden:n = true ,
665 }
666 \AddToHook { begindocument }
667 {
668     \@ifpackageloaded { zref-titleref }
669     {
670         \keys_define:nn { zref-clever / reference }
671         {
672             ref / title .code:n =
673             { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
674         }
675     }
676     {
677         \keys_define:nn { zref-clever / reference }
678         {
679             ref / title .code:n =
680             {
681                 \msg_warning:nn { zref-clever } { missing-zref-titleref }
682                 \tl_set:Nn \l__zrefclever_ref_property_tl { default }
683             }
684         }
685     }
686 }

```

typeset option

```

687 \bool_new:N \l__zrefclever_typeset_ref_bool
688 \bool_new:N \l__zrefclever_typeset_name_bool
689 \keys_define:nn { zref-clever / reference }
690 {
691     typeset .choice: ,
692     typeset / both .code:n =
693     {
694         \bool_set_true:N \l__zrefclever_typeset_ref_bool
695         \bool_set_true:N \l__zrefclever_typeset_name_bool
696     } ,
697     typeset / ref .code:n =
698     {
699         \bool_set_true:N \l__zrefclever_typeset_ref_bool
700         \bool_set_false:N \l__zrefclever_typeset_name_bool
701     } ,
702     typeset / name .code:n =
703     {
704         \bool_set_false:N \l__zrefclever_typeset_ref_bool
705         \bool_set_true:N \l__zrefclever_typeset_name_bool
706     } ,
707     typeset .initial:n = both ,
708     typeset .value_required:n = true ,
709
710     noname .meta:n = { typeset = ref },
711     noname .value_forbidden:n = true ,
712 }

```

sort option

```
713 \bool_new:N \l__zrefclever_typeset_sort_bool
714 \keys_define:nn { zref-clever / reference }
715 {
716   sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
717   sort .initial:n = true ,
718   sort .default:n = true ,
719   nosort .meta:n = { sort = false },
720   nosort .value_forbidden:n = true ,
721 }
```

typesort option

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

```
722 \seq_new:N \l__zrefclever_typesort_seq
723 \keys_define:nn { zref-clever / reference }
724 {
725   typesort .code:n =
726   {
727     \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
728     \seq_reverse:N \l__zrefclever_typesort_seq
729   } ,
730   typesort .initial:n =
731   { part , chapter , section , paragraph },
732   typesort .value_required:n = true ,
733   notypesort .code:n =
734   { \seq_clear:N \l__zrefclever_typesort_seq } ,
735   notypesort .value_forbidden:n = true ,
736 }
```

comp option

```
737 \bool_new:N \l__zrefclever_typeset_compress_bool
738 \keys_define:nn { zref-clever / reference }
739 {
740   comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
741   comp .initial:n = true ,
742   comp .default:n = true ,
743   nocomp .meta:n = { comp = false },
744   nocomp .value_forbidden:n = true ,
745 }
```

range option

```
746 \bool_new:N \l__zrefclever_typeset_range_bool
747 \keys_define:nn { zref-clever / reference }
748 {
749   range .bool_set:N = \l__zrefclever_typeset_range_bool ,
750   range .initial:n = false ,
751   range .default:n = true ,
752 }
```


cap and capfirst options

```
753 \bool_new:N \l__zrefclever_capitalize_bool
754 \bool_new:N \l__zrefclever_capitalize_first_bool
755 \keys_define:nn { zref-clever / reference }
756 {
757   cap .bool_set:N = \l__zrefclever_capitalize_bool ,
758   cap .initial:n = false ,
759   cap .default:n = true ,
760   nocap .meta:n = { cap = false },
761   nocap .value_forbidden:n = true ,
762
763   capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,
764   capfirst .initial:n = false ,
765   capfirst .default:n = true ,
766 }
```

abbrev and noabbrevfirst options

```
767 \bool_new:N \l__zrefclever_abbrev_bool
768 \bool_new:N \l__zrefclever_noabbrev_first_bool
769 \keys_define:nn { zref-clever / reference }
770 {
771   abbrev .bool_set:N = \l__zrefclever_abbrev_bool ,
772   abbrev .initial:n = false ,
773   abbrev .default:n = true ,
774   noabbrev .meta:n = { abbrev = false },
775   noabbrev .value_forbidden:n = true ,
776
777   noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
778   noabbrevfirst .initial:n = false ,
779   noabbrevfirst .default:n = true ,
780 }
```

S option

```
781 \keys_define:nn { zref-clever / reference }
782 {
783   S .meta:n =
784     { capfirst = true , noabbrevfirst = true },
785   S .value_forbidden:n = true ,
786 }
```

hyperref option

```
787 \bool_new:N \l__zrefclever_use_hyperref_bool
788 \bool_new:N \l__zrefclever_warn_hyperref_bool
789 \keys_define:nn { zref-clever / reference }
790 {
791   hyperref .choice: ,
792   hyperref / auto .code:n =
793     {
794       \bool_set_true:N \l__zrefclever_use_hyperref_bool
795       \bool_set_false:N \l__zrefclever_warn_hyperref_bool
796     } ,
797   hyperref / true .code:n =
798     {
```

```

799         \bool_set_true:N \l__zrefclever_use_hyperref_bool
800         \bool_set_true:N \l__zrefclever_warn_hyperref_bool
801     } ,
802     hyperref / false .code:n =
803     {
804         \bool_set_false:N \l__zrefclever_use_hyperref_bool
805         \bool_set_false:N \l__zrefclever_warn_hyperref_bool
806     } ,
807     hyperref .initial:n = auto ,
808     hyperref .default:n = auto
809 }
810 \AddToHook { begindocument }
811 {
812     \@ifpackageloaded { hyperref }
813     {
814         \bool_if:NT \l__zrefclever_use_hyperref_bool
815         { \RequirePackage { zref-clever } }
816     }
817     {
818         \bool_if:NT \l__zrefclever_warn_hyperref_bool
819         { \msg_warning:nn { zref-clever } { missing-hyperref } }
820         \bool_set_false:N \l__zrefclever_use_hyperref_bool
821     }
822     \keys_define:nn { zref-clever / reference }
823     {
824         hyperref .code:n =
825         { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }
826     }
827 }

```

nameinlink option

```

828 \str_new:N \l__zrefclever_nameinlink_str
829 \keys_define:nn { zref-clever / reference }
830 {
831     nameinlink .choice: ,
832     nameinlink / true .code:n =
833     { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
834     nameinlink / false .code:n =
835     { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
836     nameinlink / single .code:n =
837     { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
838     nameinlink / tsingle .code:n =
839     { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
840     nameinlink .initial:n = tsingle ,
841     nameinlink .default:n = true ,
842 }

```

lang option

`\l__zrefclever_current_language_tl` is an internal alias for babel's `\language` or polyglossia's `\mainbabelname` and, if none of them is loaded, we set it to `english`. `\l__zrefclever_main_language_tl` is an internal alias for babel's `\bbl@main@language` or for polyglossia's `\mainbabelname`, as the case may be. Note that for polyglossia we

get babel’s language names, so that we only need to handle those internally. `\l__zrefclever_ref_language_tl` is the internal variable which stores the language in which the reference is to be made.

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

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

```

843 \tl_new:N \l__zrefclever_ref_language_tl
844 \tl_new:N \l__zrefclever_main_language_tl
845 \tl_new:N \l__zrefclever_current_language_tl
846 \AddToHook { begindocument }
847 {
848   \@ifpackageloaded { babel }
849   {
850     \tl_set:Nn \l__zrefclever_current_language_tl { \language }
851     \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
852   }
853   {
854     \@ifpackageloaded { polyglossia }
855     {
856       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
857       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
858     }
859     {
860       \tl_set:Nn \l__zrefclever_current_language_tl { english }
861       \tl_set:Nn \l__zrefclever_main_language_tl { english }
862     }
863   }

```

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

```

864   \tl_set:Nn \l__zrefclever_ref_language_tl
865   { \l__zrefclever_main_language_tl }
866 }

```

```

867 \keys_define:nn { zref-clever / reference }
868 {
869   lang .code:n =
870   {
871     \AddToHook { begindocument }
872     {
873       \str_case:nnF {#1}
874       {
875         { main }
876         {
877           \tl_set:Nn \l__zrefclever_ref_language_tl
878             { \l__zrefclever_main_language_tl }
879           \__zrefclever_provide_dictionary_verbose:x
880             { \l__zrefclever_ref_language_tl }
881         }
882
883         { current }
884         {
885           \tl_set:Nn \l__zrefclever_ref_language_tl
886             { \l__zrefclever_current_language_tl }
887           \__zrefclever_provide_dictionary_verbose:x
888             { \l__zrefclever_ref_language_tl }
889         }
890       }
891     }
892     \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
893     {
894       \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
895     }
896     {
897       \msg_warning:nnn { zref-clever }
898         { unknown-language-opt } {#1}
899       \tl_set:Nn \l__zrefclever_ref_language_tl
900         { \l__zrefclever_main_language_tl }
901     }
902     \__zrefclever_provide_dictionary_verbose:x
903       { \l__zrefclever_ref_language_tl }
904   }
905 } ,
906 lang .value_required:n = true ,
907 }
908
909 \AddToHook { begindocument / before }
910 {
911   \AddToHook { begindocument }
912   {

```

If any `lang` option has been given by the user, the corresponding language is already loaded, otherwise, ensure the default one (`main`) gets loaded early, but not verbosely.

```

913   \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }

```

Redefinition of the `lang` key option for the document body. Also, drop the verbose dictionary loading in the document body, as it can become intrusive depending on the

use case, and does not provide much “juice” anyway: in `\zcref` missing names warnings will already ensue.

```

914     \keys_define:nn { zref-clever / reference }
915     {
916         lang .code:n =
917         {
918             \str_case:nnF {#1}
919             {
920                 { main }
921                 {
922                     \tl_set:Nn \l__zrefclever_ref_language_tl
923                     { \l__zrefclever_main_language_tl }
924                     \__zrefclever_provide_dictionary:x
925                     { \l__zrefclever_ref_language_tl }
926                 }
927
928                 { current }
929                 {
930                     \tl_set:Nn \l__zrefclever_ref_language_tl
931                     { \l__zrefclever_current_language_tl }
932                     \__zrefclever_provide_dictionary:x
933                     { \l__zrefclever_ref_language_tl }
934                 }
935             }
936         {
937             \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
938             {
939                 \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
940             }
941             {
942                 \msg_warning:nnn { zref-clever }
943                 { unknown-language-opt } {#1}
944                 \tl_set:Nn \l__zrefclever_ref_language_tl
945                 { \l__zrefclever_main_language_tl }
946             }
947             \__zrefclever_provide_dictionary:x
948             { \l__zrefclever_ref_language_tl }
949         }
950     } ,
951     lang .value_required:n = true ,
952 }
953 }
954 }
```

d option

Thanks @samcarter and Alan Munn for useful comments about declension on the TeX.SX chat. Also, Florent Rougon’s efforts in this area, with the `xcref` package (<https://github.com/frougon/xcref>), have been an insightful source to frame the problem in general terms.

```

955 \tl_new:N \l__zrefclever_ref_decl_case_tl
956 \keys_define:nn { zref-clever / reference }
957 {
```

```

958     d .code:n =
959       { \msg_warning:nnn { zref-clever } { option-document-only } { d } } ,
960     }
961   \AddToHook { begindocument }
962   {
963     \keys_define:nn { zref-clever / reference }
964     {

```

We just store the value at this point, since what are valid values for this variable depends on `\l__zrefclever_ref_language_tl`, which may also be set as an option. Hence, validation for this must be done after `\keys_set:nn`.

```

965       d .tl_set:N = \l__zrefclever_ref_decl_case_tl ,
966       d .value_required:n = true ,
967     }
968   }

```

`__zrefclever_process_language_options:` Auxiliary function for `__zrefclever_zcref:nnn`, responsible for processing options from `\zcDeclareLanguage`. It validates the declension case (d) option for the reference language. It is expected to be called right (or soon) after `\keys_set:nn` in `__zrefclever_zcref:nnn`, where current values for `\l__zrefclever_ref_language_tl` and `\l__zrefclever_ref_decl_case_tl` are in place. If the user value for the latter does not match the declension cases declared for the former, the function sets an appropriate value for `\l__zrefclever_ref_decl_case_tl`, either using the default case, or clearing the variable, depending on the language setup. And also issues a warning about it. This function also ensures `\l__zrefclever_capitalize_bool` is set to `true` when the language was declared with `allcaps` option.

```

969   \cs_new_protected:Npn \__zrefclever_process_language_options:
970   {
971     \exp_args:NNx \prop_get:NnNTF \g__zrefclever_languages_prop
972     { \l__zrefclever_ref_language_tl }
973     \l__zrefclever_dict_language_tl
974     {
975       % ‘declension’ option.
976       \exp_args:NNx \seq_set_from_clist:Nn
977       \l__zrefclever_dict_declension_seq
978       {
979         \prop_item:cn
980         {
981           g__zrefclever_dict_
982           \l__zrefclever_dict_language_tl _prop
983         }
984         { declension }
985       }
986       \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
987       {
988         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
989         {
990           \msg_warning:nnxx { zref-clever }
991           { language-no-decl-ref }
992           { \l__zrefclever_ref_language_tl }
993           { \l__zrefclever_ref_decl_case_tl }
994           \tl_clear:N \l__zrefclever_ref_decl_case_tl
995         }

```

```

996     }
997     {
998         \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
999         {
1000             \seq_get_left:NN \l__zrefclever_dict_declension_seq
1001             \l__zrefclever_ref_decl_case_tl
1002         }
1003         {
1004             \seq_if_in:NVF \l__zrefclever_dict_declension_seq
1005             \l__zrefclever_ref_decl_case_tl
1006             {
1007                 \msg_warning:nxxx { zref-clever }
1008                 { unknown-decl-case }
1009                 { \l__zrefclever_ref_decl_case_tl }
1010                 { \l__zrefclever_ref_language_tl }
1011                 \seq_get_left:NN \l__zrefclever_dict_declension_seq
1012                 \l__zrefclever_ref_decl_case_tl
1013             }
1014         }
1015     }
1016     % 'gender' option.
1017     \exp_args:NNx \seq_set_from_clist:Nn
1018     \l__zrefclever_dict_gender_seq
1019     {
1020         \prop_item:cn
1021         {
1022             g__zrefclever_dict_
1023             \l__zrefclever_dict_language_tl _prop
1024         }
1025         { gender }
1026     }
1027     \seq_if_empty:NTF \l__zrefclever_dict_gender_seq
1028     {
1029         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
1030         {
1031             \msg_warning:nxxxx { zref-clever }
1032             { language-no-gender }
1033             { \l__zrefclever_ref_language_tl }
1034             { g }
1035             { \l__zrefclever_ref_gender_tl }
1036             \tl_clear:N \l__zrefclever_ref_gender_tl
1037         }
1038     }
1039     {
1040         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
1041         {
1042             \seq_if_in:NVF \l__zrefclever_dict_gender_seq
1043             \l__zrefclever_ref_gender_tl
1044             {
1045                 \msg_warning:nxxx { zref-clever }
1046                 { gender-not-declared }
1047                 { \l__zrefclever_ref_language_tl }
1048                 { \l__zrefclever_ref_gender_tl }
1049                 \tl_clear:N \l__zrefclever_ref_gender_tl

```

```

1050         }
1051     }
1052 }
1053
1054 % 'allcaps' option.
1055 \str_if_eq:eeT
1056 {
1057     \prop_item:cn
1058     {
1059         g__zrefclever_dict_
1060         \l__zrefclever_dict_language_tl _prop
1061     }
1062     { allcaps }
1063 }
1064 { true }
1065 { \bool_set_true:N \l__zrefclever_capitalize_bool }
1066 }
1067 {
1068     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
1069     {
1070         \msg_warning:nxxx { zref-clever } { unknown-language-decl }
1071         { \l__zrefclever_ref_decl_case_tl }
1072         { \l__zrefclever_ref_language_tl }
1073         \tl_clear:N \l__zrefclever_ref_decl_case_tl
1074     }
1075 }
1076 }

```

(End definition for `__zrefclever_process_language_options:.`)

nudge & Co. options

```

1077 \bool_new:N \l__zrefclever_nudge_enabled_bool
1078 \bool_new:N \l__zrefclever_nudge_multitype_bool
1079 \bool_new:N \l__zrefclever_nudge_comptosing_bool
1080 \bool_new:N \l__zrefclever_nudge_singular_bool
1081 \bool_new:N \l__zrefclever_nudge_gender_bool
1082 \tl_new:N \l__zrefclever_ref_gender_tl
1083 \keys_define:nn { zref-clever / reference }
1084 {
1085     nudge .choice: ,
1086     nudge / true .code:n =
1087     { \bool_set_true:N \l__zrefclever_nudge_enabled_bool } ,
1088     nudge / false .code:n =
1089     { \bool_set_false:N \l__zrefclever_nudge_enabled_bool } ,
1090     nudge / obeydraft .code:n =
1091     {
1092         \ifdraft
1093         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
1094         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
1095     } ,
1096     nudge / obeyfinal .code:n =
1097     {
1098         \ifoptionfinal
1099         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }

```



```

1100         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
1101     } ,
1102     nudge .initial:n = false ,
1103     nudge .default:n = true ,
1104     nonnudge .meta:n = { nudge = false } ,
1105     nonnudge .value_forbidden:n = true ,
1106     nudgeif .code:n =
1107     {
1108         \bool_set_false:N \l__zrefclever_nudge_multitype_bool
1109         \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
1110         \bool_set_false:N \l__zrefclever_nudge_gender_bool
1111         \clist_map_inline:nn {#1}
1112         {
1113             \str_case:nnF {##1}
1114             {
1115                 { multitype }
1116                 { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
1117                 { comptosing }
1118                 { \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
1119                 { gender }
1120                 { \bool_set_true:N \l__zrefclever_nudge_gender_bool }
1121                 { all }
1122                 {
1123                     \bool_set_true:N \l__zrefclever_nudge_multitype_bool
1124                     \bool_set_true:N \l__zrefclever_nudge_comptosing_bool
1125                     \bool_set_true:N \l__zrefclever_nudge_gender_bool
1126                 }
1127             }
1128         {
1129             \msg_warning:nnn { zref-clever }
1130             { nudgeif-unknown-value } {##1}
1131         }
1132     }
1133 } ,
1134 nudgeif .value_required:n = true ,
1135 nudgeif .initial:n = all ,
1136 sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
1137 sg .initial:n = false ,
1138 sg .default:n = true ,
1139 g .code:n =
1140 { \msg_warning:nnn { zref-clever } { option-document-only } { g } } ,
1141 }
1142 \AddToHook { begindocument }
1143 {
1144     \keys_define:nn { zref-clever / reference }
1145     {

```

Same thing as for d option.

```

1146         g .tl_set:N = \l__zrefclever_ref_gender_tl ,
1147         g .value_required:n = true ,
1148     }
1149 }

```

font option

`font` *can't be used as a package option*, since the options get expanded by L^AT_EX before being passed to the package (see <https://tex.stackexchange.com/a/489570>). It can't be set in `\zcref` and, for global settings, with `\zcsetup`. Note that, technically, the “raw” options are already available as `\@raw@opt@<package>.sty` (see <https://tex.stackexchange.com/a/618439>, thanks David Carlisle).

```
1150 \tl_new:N \l__zrefclever_ref_typeset_font_tl
1151 \keys_define:nn { zref-clever / reference }
1152 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }
```

titleref option

```
1153 \keys_define:nn { zref-clever / reference }
1154 {
1155     titleref .code:n = { \RequirePackage { zref-titleref } } ,
1156     titleref .value_forbidden:n = true ,
1157 }
1158 \AddToHook { begindocument }
1159 {
1160     \keys_define:nn { zref-clever / reference }
1161     {
1162         titleref .code:n =
1163         { \msg_warning:nn { zref-clever } { titleref-preamble-only } }
1164     }
1165 }
```

note option

```
1166 \tl_new:N \l__zrefclever_zcref_note_tl
1167 \keys_define:nn { zref-clever / reference }
1168 {
1169     note .tl_set:N = \l__zrefclever_zcref_note_tl ,
1170     note .value_required:n = true ,
1171 }
```

check option

Integration with `zref-check`.

```
1172 \bool_new:N \l__zrefclever_zrefcheck_available_bool
1173 \bool_new:N \l__zrefclever_zcref_with_check_bool
1174 \keys_define:nn { zref-clever / reference }
1175 {
1176     check .code:n = { \RequirePackage { zref-check } } ,
1177     check .value_forbidden:n = true ,
1178 }
1179 \AddToHook { begindocument }
1180 {
1181     \@ifpackageloaded { zref-check }
1182     {
1183         \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
1184         \keys_define:nn { zref-clever / reference }
1185         {
1186             check .code:n =
1187             {
```

```

1188         \bool_set_true:N \l__zrefclever_zcref_with_check_bool
1189         \keys_set:nn { zref-check / zcheck } {#1}
1190     } ,
1191     check .value_required:n = true ,
1192 }
1193 }
1194 {
1195     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
1196     \keys_define:nn { zref-clever / reference }
1197     {
1198         check .value_forbidden:n = false ,
1199         check .code:n =
1200         { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
1201     }
1202 }
1203 }

```

countertype option

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

```

1204 \prop_new:N \l__zrefclever_counter_type_prop
1205 \keys_define:nn { zref-clever / label }
1206 {
1207     countertype .code:n =
1208     {
1209         \keyval_parse:nnn
1210         {
1211             \msg_warning:nnnn { zref-clever }
1212             { key-requires-value } { countertype }
1213         }
1214         {
1215             \__zrefclever_prop_put_non_empty:Nnn
1216             \l__zrefclever_counter_type_prop
1217         }
1218         {#1}
1219     } ,
1220     countertype .value_required:n = true ,
1221     countertype .initial:n =
1222     {
1223         subsection      = section ,
1224         subsubsection    = section ,
1225         subparagraph     = paragraph ,
1226         enumi            = item ,
1227         enumii           = item ,
1228         enumiii          = item ,
1229         enumiv           = item ,
1230         mpfootnote       = footnote ,
1231     } ,
1232 }

```

counterresetters option

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

```
1233 \seq_new:N \l__zrefclever_counter_resetters_seq
1234 \keys_define:nn { zref-clever / label }
1235 {
1236   counterresetters .code:n =
1237   {
1238     \clist_map_inline:nn {#1}
1239     {
1240       \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
1241       {
1242         \seq_put_right:Nn
1243         \l__zrefclever_counter_resetters_seq {##1}
1244       }
1245     }
1246   } ,
1247   counterresetters .initial:n =
1248   {
1249     part ,
1250     chapter ,
1251     section ,
1252     subsection ,
1253     subsubsection ,
1254     paragraph ,
1255     subparagraph ,
1256   },
1257   counterresetters .value_required:n = true ,
1258 }
```

counterresetby option

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

```
1259 \prop_new:N \l__zrefclever_counter_resetby_prop
1260 \keys_define:nn { zref-clever / label }
1261 {
1262   counterresetby .code:n =
1263   {
1264     \keyval_parse:nnn
1265     {
1266       \msg_warning:nnn { zref-clever }
1267       { key-requires-value } { counterresetby }
1268     }
1269   }
```

```

1269         {
1270             \_zrefclever_prop_put_non_empty:Nnn
1271             \l_zrefclever_counter_resetby_prop
1272         }
1273         {#1}
1274     } ,
1275     counterresetby .value_required:n = true ,
1276     counterresetby .initial:n =
1277     {

```

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.

```

1278         enumii = enumi ,
1279         enumiii = enumii ,
1280         enumiv = enumiii ,
1281     } ,
1282 }

```

currentcounter option

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

```

1283 \tl_new:N \l_zrefclever_current_counter_tl
1284 \keys_define:nn { zref-clever / label }
1285 {
1286     currentcounter .tl_set:N = \l_zrefclever_current_counter_tl ,
1287     currentcounter .value_required:n = true ,
1288     currentcounter .initial:n = \@currentcounter ,
1289 }

```

Reference options

This is a set of options related to reference typesetting which receive equal treatment and, hence, are handled in batch. Since we are dealing with options to be passed to `\zceref` or to `\zcsetup` or at load time, only “not necessarily type-specific” options are pertinent here. However, they *may* either be type-specific or language-specific, and thus must be stored in a property list, `\l_zrefclever_ref_options_prop`, in order to be retrieved from the option *name* by `_zrefclever_get_ref_string:nN` and `_zrefclever_get_ref_font:nN` according to context and precedence rules.

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

```

1290 \prop_new:N \l_zrefclever_ref_options_prop
1291 \seq_map_inline:Nn
1292     \c_zrefclever_ref_options_reference_seq
1293     {
1294         \keys_define:nn { zref-clever / reference }
1295         {

```

```

1296      #1 .default:V = \c_novalue_tl ,
1297      #1 .code:n =
1298      {
1299          \tl_if_novalue:nTF {##1}
1300          { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
1301          { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
1302      } ,
1303  }
1304 }

```

Package options

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

```

1305 \keys_define:nn { }
1306 {
1307     zref-clever / zcsetup .inherit:n =
1308     {
1309         zref-clever / label ,
1310         zref-clever / reference ,
1311     }
1312 }

```

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

```

1313 \ProcessKeysOptions { zref-clever / zcsetup }

```

5 Configuration

5.1 \zcsetup

`\zcsetup` Provide `\zcsetup`.

```

\zcsetup{\options}

```

```

1314 \NewDocumentCommand \zcsetup { m }
1315 { \__zrefclever_zcsetup:n {#1} }

```

(End definition for \zcsetup.)

`__zrefclever_zcsetup:n` A version of `\zcsetup` for internal use with variant.

```

\__zrefclever_zcsetup:n{\options}

```

```

1316 \cs_new_protected:Npn \__zrefclever_zcsetup:n #1
1317 { \keys_set:nn { zref-clever / zcsetup } {#1} }
1318 \cs_generate_variant:Nn \__zrefclever_zcsetup:n { x }

```

(End definition for __zrefclever_zcsetup:n.)

5.2 \zcRefTypeSetup

`\zcRefTypeSetup` is the main user interface for “type-specific” reference formatting. Settings done by this command have a higher precedence than any translation, hence they override any language-specific setting, either done at `\zcLanguageSetup` or by the package’s dictionaries. On the other hand, they have a lower precedence than non type-specific general options. The *⟨options⟩* should be given in the usual `key=val` format. The *⟨type⟩* does not need to pre-exist, the property list variable to store the properties for the type gets created if need be.

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

1319 \NewDocumentCommand \zcRefTypeSetup { m m }
1320 {
1321   \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
1322   { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
1323   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
1324   \keys_set:nn { zref-clever / typesetup } {#2}
1325 }
```

(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 been made in `\l__zrefclever_type_⟨type⟩_options_prop` or in `\l__zrefclever_ref_options_prop` it stays there forever, and can only be overridden by a new value at the same precedence level or a higher one. But it would be nice if an user can “unset” an option at either of those scopes to go back to the lower precedence level of the translations at any given point. So both in `\zcRefTypeSetup` and in setting reference options (see Section 4.6), we leverage the distinction of an “empty valued key” (`key=` or `key={}`) from a “key with no value” (`key`). This distinction is captured internally by the lower-level key parsing, but must be made explicit at `\keys_set:nn` by means of the `.default:V` property of the key in `\keys_define:nn`. For the technique and some discussion about it, see <https://tex.stackexchange.com/q/614690> (thanks Jonathan P. Spratte, aka ‘Skillmon’, and Phelype Oleinik) and <https://github.com/latex3/latex3/pull/988>.

```
1326 \seq_map_inline:Nn
1327   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
1328   {
1329     \keys_define:nn { zref-clever / typesetup }
1330     {
1331       #1 .code:n =
1332       {
1333         \msg_warning:nnn { zref-clever }
1334         { option-not-type-specific } {#1}
1335       } ,
1336     }
1337   }

1338 \seq_map_inline:Nn
1339   \c__zrefclever_ref_options_typesetup_seq
1340   {
1341     \keys_define:nn { zref-clever / typesetup }
1342     {
1343       #1 .default:V = \c_novaluel_tl ,
```

```

1344     #1 .code:n =
1345     {
1346         \tl_if_novalue:nTF {##1}
1347         {
1348             \prop_remove:cn
1349             {
1350                 l__zrefclever_type_
1351                 \l__zrefclever_setup_type_tl _options_prop
1352             }
1353             {#1}
1354         }
1355         {
1356             \prop_put:cnn
1357             {
1358                 l__zrefclever_type_
1359                 \l__zrefclever_setup_type_tl _options_prop
1360             }
1361             {#1} {##1}
1362         }
1363     } ,
1364 }
1365 }

```

5.3 \zcLanguageSetup

\zcLanguageSetup is the main user interface for “language-specific” reference formatting, be it “type-specific” or not. The difference between the two cases is captured by the `type` key, which works as a sort of a “switch”. Inside the `<options>` argument of \zcLanguageSetup, any options made before the first `type` key declare “default” (non type-specific) translations. When the `type` key is given with a value, the options following it will set “type-specific” translations for that type. The current type can be switched off by an empty `type` key. \zcLanguageSetup is preamble only.

```

\zcLanguageSetup      \zcLanguageSetup{<language>}{<options>}
1366 \NewDocumentCommand \zcLanguageSetup { m m }
1367 {
1368     \group_begin:
1369     \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
1370     \l__zrefclever_dict_language_tl
1371     {
1372         \tl_clear:N \l__zrefclever_setup_type_tl
1373         \exp_args:NNx \seq_set_from_clist:Nn
1374         \l__zrefclever_dict_declension_seq
1375         {
1376             \prop_item:cn
1377             {
1378                 g__zrefclever_dict_
1379                 \l__zrefclever_dict_language_tl _prop
1380             }
1381             { declension }
1382         }
1383         \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
1384         { \tl_clear:N \l__zrefclever_dict_decl_case_tl }

```



```

1385         {
1386             \seq_get_left:NN \l__zrefclever_dict_declension_seq
1387             \l__zrefclever_dict_decl_case_tl
1388         }
1389     \exp_args:NNx \seq_set_from_clist:Nn
1390     \l__zrefclever_dict_gender_seq
1391     {
1392         \prop_item:cn
1393         {
1394             g__zrefclever_dict_
1395             \l__zrefclever_dict_language_tl _prop
1396         }
1397         { gender }
1398     }
1399     \keys_set:nn { zref-clever / langsetup } {#2}
1400 }
1401 { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
1402 \group_end:
1403 }
1404 \@onlypreamble \zcLanguageSetup

```

(End definition for \zcLanguageSetup.)

_zrefclever_declare_type_transl:nnnn
_zrefclever_declare_default_transl:nnn

A couple of auxiliary functions for the of zref-clever/translation keys set in \zcLanguageSetup. They respectively declare (unconditionally set) “type-specific” and “default” translations.

```

\__zrefclever_declare_type_transl:nnnn {<language>} {<type>}
{<key>} {<translation>}
\__zrefclever_declare_default_transl:nnn {<language>}
{<key>} {<translation>}

1405 \cs_new_protected:Npn \__zrefclever_declare_type_transl:nnnn #1#2#3#4
1406 {
1407     \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
1408     { type- #2 - #3 } {#4}
1409 }
1410 \cs_generate_variant:Nn \__zrefclever_declare_type_transl:nnnn { VVnn , VVxn }
1411 \cs_new_protected:Npn \__zrefclever_declare_default_transl:nnn #1#2#3
1412 {
1413     \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
1414     { default- #2 } {#3}
1415 }
1416 \cs_generate_variant:Nn \__zrefclever_declare_default_transl:nnn { Vnn }

```

(End definition for __zrefclever_declare_type_transl:nnnn and __zrefclever_declare_default_transl:nnn.)

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

```

1417 \keys_define:nn { zref-clever / langsetup }
1418 {
1419     type .code:n =
1420     {
1421         \tl_if_empty:nTF {#1}
1422         { \tl_clear:N \l__zrefclever_setup_type_tl }

```

```

1423         { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
1424     } ,
1425     case .code:n =
1426     {
1427         \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
1428         {
1429             \msg_warning:nxxx { zref-clever } { language-no-decl-setup }
1430             { \l__zrefclever_dict_language_tl } {#1}
1431         }
1432         {
1433             \seq_if_in:NnTF \l__zrefclever_dict_declension_seq {#1}
1434             { \tl_set:Nn \l__zrefclever_dict_decl_case_tl {#1} }
1435             {
1436                 \msg_warning:nxxx { zref-clever } { unknown-decl-case }
1437                 {#1} { \l__zrefclever_dict_language_tl }
1438                 \seq_get_left:NN \l__zrefclever_dict_declension_seq
1439                 \l__zrefclever_dict_decl_case_tl
1440             }
1441         }
1442     } ,
1443     case .value_required:n = true ,
1444     gender .code:n =
1445     {
1446         \seq_if_empty:NTF \l__zrefclever_dict_gender_seq
1447         {
1448             \msg_warning:nxxxx { zref-clever } { language-no-gender }
1449             { \l__zrefclever_dict_language_tl } { gender } {#1}
1450         }
1451         {
1452             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1453             {
1454                 \msg_warning:nnn { zref-clever }
1455                 { option-only-type-specific } { gender }
1456             }
1457             {
1458                 \seq_if_in:NnTF \l__zrefclever_dict_gender_seq {#1}
1459                 {
1460                     \__zrefclever_declare_type_transl:VWnn
1461                     \l__zrefclever_dict_language_tl
1462                     \l__zrefclever_setup_type_tl
1463                     { gender } {#1}
1464                 }
1465                 {
1466                     \msg_warning:nxxx { zref-clever } { gender-not-declared }
1467                     { \l__zrefclever_dict_language_tl } {#1}
1468                 }
1469             }
1470         }
1471     } ,
1472     gender .value_required:n = true ,
1473 }
1474 \seq_map_inline:Nn
1475 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
1476 {

```

```

1477 \keys_define:nn { zref-clever / langsetup }
1478 {
1479   #1 .value_required:n = true ,
1480   #1 .code:n =
1481   {
1482     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1483     {
1484       \__zrefclever_declare_default_transl:Vnn
1485       \l__zrefclever_dict_language_tl
1486       {#1} {##1}
1487     }
1488     {
1489       \msg_warning:nnn { zref-clever }
1490       { option-not-type-specific } {#1}
1491     }
1492   } ,
1493 }
1494 }
1495 \seq_map_inline:Nn
1496 \c__zrefclever_ref_options_possibly_type_specific_seq
1497 {
1498   \keys_define:nn { zref-clever / langsetup }
1499   {
1500     #1 .value_required:n = true ,
1501     #1 .code:n =
1502     {
1503       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1504       {
1505         \__zrefclever_declare_default_transl:Vnn
1506         \l__zrefclever_dict_language_tl
1507         {#1} {##1}
1508       }
1509       {
1510         \__zrefclever_declare_type_transl:VVnn
1511         \l__zrefclever_dict_language_tl
1512         \l__zrefclever_setup_type_tl
1513         {#1} {##1}
1514       }
1515     } ,
1516   }
1517 }
1518 \seq_map_inline:Nn
1519 \c__zrefclever_ref_options_type_names_seq
1520 {
1521   \keys_define:nn { zref-clever / langsetup }
1522   {
1523     #1 .value_required:n = true ,
1524     #1 .code:n =
1525     {
1526       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1527       {
1528         \msg_warning:nnn { zref-clever }
1529         { option-only-type-specific } {#1}
1530       }
1531     }
1532   }

```

```

1531         {
1532             \tl_if_empty:NTF \l__zrefclever_dict_decl_case_tl
1533             {
1534                 \__zrefclever_declare_type_transl:VWnn
1535                 \l__zrefclever_dict_language_tl
1536                 \l__zrefclever_setup_type_tl
1537                 {#1} {##1}
1538             }
1539             {
1540                 \__zrefclever_declare_type_transl:VWxn
1541                 \l__zrefclever_dict_language_tl
1542                 \l__zrefclever_setup_type_tl
1543                 { \l__zrefclever_dict_decl_case_tl - #1 } {##1}
1544             }
1545         }
1546     } ,
1547 }
1548 }

```

6 User interface

6.1 \zcref

`\zcref` The main user command of the package.

```
\zcref<*>[<options>]{<labels>}
```

```

1549 \NewDocumentCommand \zcref { s O { } m }
1550 { \zref@wrapper@babel \__zrefclever_zcref:nnn {#3} {#1} {#2} }

```

(End definition for `\zcref`.)

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

```
\__zrefclever_zcref:nnnn {<labels>} {<*>} {<options>}
```

```

1551 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
1552 {
1553     \group_begin:

```

Set options.

```
1554     \keys_set:nn { zref-clever / reference } {#3}
```

Store arguments values.

```

1555     \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
1556     \bool_set:Nn \l__zrefclever_link_star_bool {#2}

```

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

```
1557     \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }
```

Process `\zcDeclareLanguage` options.

```
1558 \__zrefclever_process_language_options:
```

Integration with `zref-check`.

```
1559 \bool_lazy_and:nnT
1560 { \l__zrefclever_zrefcheck_available_bool }
1561 { \l__zrefclever_zcref_with_check_bool }
1562 { \zrefcheck_zcref_beg_label: }
```

Sort the labels.

```
1563 \bool_lazy_or:nnT
1564 { \l__zrefclever_typeset_sort_bool }
1565 { \l__zrefclever_typeset_range_bool }
1566 { \__zrefclever_sort_labels: }
```

Typeset the references. Also, set the reference font, and group it, so that it does not leak to the note.

```
1567 \group_begin:
1568 \l__zrefclever_ref_typeset_font_tl
1569 \__zrefclever_typeset_refs:
1570 \group_end:
```

Typeset note.

```
1571 \tl_if_empty:NF \l__zrefclever_zcref_note_tl
1572 {
1573   \__zrefclever_get_ref_string:nN { notesep } \l_tmpa_tl
1574   \l_tmpa_tl
1575   \l__zrefclever_zcref_note_tl
1576 }
```

Integration with `zref-check`.

```
1577 \bool_lazy_and:nnT
1578 { \l__zrefclever_zrefcheck_available_bool }
1579 { \l__zrefclever_zcref_with_check_bool }
1580 {
1581   \zrefcheck_zcref_end_label_maybe:
1582   \zrefcheck_zcref_run_checks_on_labels:n
1583   { \l__zrefclever_zcref_labels_seq }
1584 }
```

Integration with `mathtools`.

```
1585 \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
1586 {
1587   \__zrefclever_mathtools_showonlyrefs:n
1588   { \l__zrefclever_zcref_labels_seq }
1589 }
1590 \group_end:
1591 }
```

(End definition for `__zrefclever_zcref:nnnn`.)

```
\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool
```

```
1592 \seq_new:N \l__zrefclever_zcref_labels_seq
1593 \bool_new:N \l__zrefclever_link_star_bool
```

(End definition for `\l__zrefclever_zcref_labels_seq` and `\l__zrefclever_link_star_bool`.)

6.2 \zcpageref

\zcpageref A \pageref equivalent of \zcref.

```

\zcpageref<*>[<options>]{<labels>}

1594 \NewDocumentCommand \zcpageref { s O { } m }
1595 {
1596   \IfBooleanTF {#1}
1597     { \zcref*[#2, ref = page] {#3} }
1598     { \zcref [ #2, ref = page] {#3} }
1599 }

```

(End definition for \zcpageref.)

7 Sorting

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

<pre> \l_zrefclever_label_type_a_tl \l_zrefclever_label_type_b_tl \l_zrefclever_label_enclval_a_tl \l_zrefclever_label_enclval_b_tl \l_zrefclever_label_extdoc_a_tl \l_zrefclever_label_extdoc_b_tl </pre>	<p>Auxiliary variables, for use in sorting, and some also in typesetting. Used to store reference information – label properties – of the “current” (a) and “next” (b) labels.</p> <pre> 1600 \tl_new:N \l__zrefclever_label_type_a_tl 1601 \tl_new:N \l__zrefclever_label_type_b_tl 1602 \tl_new:N \l__zrefclever_label_enclval_a_tl 1603 \tl_new:N \l__zrefclever_label_enclval_b_tl 1604 \tl_new:N \l__zrefclever_label_extdoc_a_tl 1605 \tl_new:N \l__zrefclever_label_extdoc_b_tl </pre>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

(End definition for \l_zrefclever_label_type_a_tl and others.)

<pre> \l_zrefclever_sort_decided_bool </pre>	<p>Auxiliary variable for \l__zrefclever_sort_default_same_type:nn, signals if the sorting between two labels has been decided or not.</p> <pre> 1606 \bool_new:N \l__zrefclever_sort_decided_bool </pre>
----------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

(End definition for \l_zrefclever_sort_decided_bool.)

<pre> \l_zrefclever_sort_prior_a_int \l_zrefclever_sort_prior_b_int </pre>	<p>Auxiliary variables for \l__zrefclever_sort_default_different_types:nn. Store the sort priority of the “current” and “next” labels.</p> <pre> 1607 \int_new:N \l__zrefclever_sort_prior_a_int 1608 \int_new:N \l__zrefclever_sort_prior_b_int </pre>
----------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

(End definition for \l_zrefclever_sort_prior_a_int and \l_zrefclever_sort_prior_b_int.)

`\l_zrefclever_label_types_seq` Stores the order in which reference types appear in the label list supplied by the user in `\zcref`. This variable is populated by `__zrefclever_label_type_put_new_right:n` at the start of `__zrefclever_sort_labels:`. This order is required as a “last resort” sort criterion between the reference types, for use in `__zrefclever_sort_default_different_types:nn`.

```
1609 \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.

```
1610 \cs_new_protected:Npn \__zrefclever_sort_labels:
1611 {
```

Store label types sequence.

```
1612   \seq_clear:N \l__zrefclever_label_types_seq
1613   \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
1614   {
1615     \seq_map_function:NN \l__zrefclever_zcref_labels_seq
1616     \__zrefclever_label_type_put_new_right:n
1617   }
```

Sort.

```
1618   \seq_sort:Nn \l__zrefclever_zcref_labels_seq
1619   {
1620     \zref@ifrefundefined {##1}
1621     {
1622       \zref@ifrefundefined {##2}
1623       {
1624         % Neither label is defined.
1625         \sort_return_same:
1626       }
1627       {
1628         % The second label is defined, but the first isn't, leave the
1629         % undefined first (to be more visible).
1630         \sort_return_same:
1631       }
1632     }
1633     {
1634       \zref@ifrefundefined {##2}
1635       {
1636         % The first label is defined, but the second isn't, bring the
1637         % second forward.
1638         \sort_return_swapped:
1639       }
1640       {
1641         % The interesting case: both labels are defined. References
1642         % to the "default" property or to the "page" are quite
1643         % different with regard to sorting, so we branch them here to
1644         % specialized functions.
1645         \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
```

```

1646         { \_zrefclever_sort_page:nn {##1} {##2} }
1647         { \_zrefclever_sort_default:nn {##1} {##2} }
1648     }
1649 }
1650 }
1651 }

```

(End definition for _zrefclever_sort_labels:.)

_zrefclever_label_type_put_new_right:n

Auxiliary function used to store the order in which reference types appear in the label list supplied by the user in \zceref. It is expected to be run inside _zrefclever_sort_labels:, and stores the types sequence in \l_zrefclever_label_types_seq. I have tried to handle the same task inside \seq_sort:Nn in _zrefclever_sort_labels: to spare mapping over \l_zrefclever_zceref_labels_seq, but it turned out it not to be easy to rely on the order the labels get processed at that point, since the variable is being sorted there. Besides, the mapping is simple, not a particularly expensive operation. Anyway, this keeps things clean.

```

    \_zrefclever_label_type_put_new_right:n {<label>}

1652 \cs_new_protected:Npn \_zrefclever_label_type_put_new_right:n #1
1653 {
1654   \_zrefclever_def_extract:Nnnn
1655   \l\_zrefclever_label_type_a_tl {#1} { zc@type } { \c_empty_tl }
1656   \seq_if_in:NVF \l\_zrefclever_label_types_seq
1657   \l\_zrefclever_label_type_a_tl
1658   {
1659     \seq_put_right:NV \l\_zrefclever_label_types_seq
1660     \l\_zrefclever_label_type_a_tl
1661   }
1662 }

```

(End definition for _zrefclever_label_type_put_new_right:n.)

_zrefclever_sort_default:nn

The heavy-lifting function for sorting of defined labels for “default” references (that is, a standard reference, not to “page”). This function is expected to be called within the sorting loop of _zrefclever_sort_labels: and receives the pair of labels being considered for a change of order or not. It should *always* “return” either \sort_return_same: or \sort_return_swapped:.

```

    \_zrefclever_sort_default:nn {<label a>} {<label b>}

1663 \cs_new_protected:Npn \_zrefclever_sort_default:nn #1#2
1664 {
1665   \_zrefclever_def_extract:Nnnn
1666   \l\_zrefclever_label_type_a_tl {#1} { zc@type } { \c_empty_tl }
1667   \_zrefclever_def_extract:Nnnn
1668   \l\_zrefclever_label_type_b_tl {#2} { zc@type } { \c_empty_tl }
1669
1670   \bool_if:nTF
1671   {
1672     % The second label has a type, but the first doesn't, leave the
1673     % undefined first (to be more visible).
1674     \tl_if_empty_p:N \l\_zrefclever_label_type_a_tl &&
1675     ! \tl_if_empty_p:N \l\_zrefclever_label_type_b_tl

```



```

1676 }
1677 { \sort_return_same: }
1678 {
1679   \bool_if:nTF
1680   {
1681     % The first label has a type, but the second doesn't, bring the
1682     % second forward.
1683     ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1684     \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1685   }
1686   { \sort_return_swapped: }
1687   {
1688     \bool_if:nTF
1689     {
1690       % The interesting case: both labels have a type...
1691       ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1692       ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1693     }
1694     {
1695       \tl_if_eq:NNTF
1696       \l__zrefclever_label_type_a_tl
1697       \l__zrefclever_label_type_b_tl
1698       % ...and it's the same type.
1699       { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
1700       % ...and they are different types.
1701       { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
1702     }
1703     {
1704       % Neither label has a type. We can't do much of meaningful
1705       % here, but if it's the same counter, compare it.
1706       \exp_args:Nxx \tl_if_eq:nnTF
1707       { \__zrefclever_extract_unexp:nnn {#1} {zc@counter} { } }
1708       { \__zrefclever_extract_unexp:nnn {#2} {zc@counter} { } }
1709       {
1710         \int_compare:nNnTF
1711         { \__zrefclever_extract:nnn {#1} {zc@cntval} { -1 } }
1712         >
1713         { \__zrefclever_extract:nnn {#2} {zc@cntval} { -1 } }
1714         { \sort_return_swapped: }
1715         { \sort_return_same: }
1716       }
1717       { \sort_return_same: }
1718     }
1719   }
1720 }
1721 }

```

(End definition for __zrefclever_sort_default:nn.)

```

\__zrefclever_sort_default_same_type:nn      \__zrefclever_sort_default_same_type:nn {<label a>} {<label b>}
1722 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
1723 {
1724   \__zrefclever_def_extract:Nnnn \l__zrefclever_label_enclval_a_tl
1725   {#1} {zc@enclval} { \c_empty_tl }

```

```

1726 \tl_reverse:N \l__zrefclever_label_enclval_a_tl
1727 \__zrefclever_def_extract:Nnnn \l__zrefclever_label_enclval_b_tl
1728   {#2} { zc@enclval } { \c_empty_tl }
1729 \tl_reverse:N \l__zrefclever_label_enclval_b_tl
1730 \__zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_a_tl
1731   {#1} { externaldocument } { \c_empty_tl }
1732 \__zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_b_tl
1733   {#2} { externaldocument } { \c_empty_tl }
1734
1735 \bool_set_false:N \l__zrefclever_sort_decided_bool
1736
1737 % First we check if there's any "external document" difference (coming
1738 % from 'zref-xr') and, if so, sort based on that.
1739 \tl_if_eq:NNF
1740   \l__zrefclever_label_extdoc_a_tl
1741   \l__zrefclever_label_extdoc_b_tl
1742   {
1743     \bool_if:nTF
1744     {
1745       \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
1746       ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
1747     }
1748     {
1749       \bool_set_true:N \l__zrefclever_sort_decided_bool
1750       \sort_return_same:
1751     }
1752     {
1753       \bool_if:nTF
1754       {
1755         ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
1756         \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
1757       }
1758       {
1759         \bool_set_true:N \l__zrefclever_sort_decided_bool
1760         \sort_return_swapped:
1761       }
1762       {
1763         \bool_set_true:N \l__zrefclever_sort_decided_bool
1764         % Two different "external documents": last resort, sort by the
1765         % document name itself.
1766         \str_compare:eNeTF
1767         { \l__zrefclever_label_extdoc_b_tl } <
1768         { \l__zrefclever_label_extdoc_a_tl }
1769         { \sort_return_swapped: }
1770         { \sort_return_same: }
1771       }
1772     }
1773   }
1774
1775 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1776 {
1777   \bool_if:nTF
1778   {
1779     % Both are empty: neither label has any (further) "enclosing

```

```

1780 % counters" (left).
1781 \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
1782 \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
1783 }
1784 {
1785   \bool_set_true:N \l__zrefclever_sort_decided_bool
1786   \int_compare:nNnTF
1787     { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
1788     >
1789     { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
1790     { \sort_return_swapped: }
1791     { \sort_return_same: }
1792 }
1793 {
1794   \bool_if:nTF
1795   {
1796     % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
1797     \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
1798   }
1799   {
1800     \bool_set_true:N \l__zrefclever_sort_decided_bool
1801     \int_compare:nNnTF
1802       { \__zrefclever_extract:nnn {#1} { zc@cntval } { } }
1803       >
1804       { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1805       { \sort_return_swapped: }
1806       { \sort_return_same: }
1807   }
1808   {
1809     \bool_if:nTF
1810     {
1811       % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
1812       \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
1813     }
1814     {
1815       \bool_set_true:N \l__zrefclever_sort_decided_bool
1816       \int_compare:nNnTF
1817         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1818         <
1819         { \__zrefclever_extract:nnn {#2} { zc@cntval } { } }
1820         { \sort_return_same: }
1821         { \sort_return_swapped: }
1822     }
1823     {
1824       % Neither is empty: we can compare the values of the
1825       % current enclosing counter in the loop, if they are
1826       % equal, we are still in the loop, if they are not, a
1827       % sorting decision can be made directly.
1828       \int_compare:nNnTF
1829         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1830         =
1831         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1832         {
1833           \tl_set:Nx \l__zrefclever_label_enclval_a_tl

```

```

1834         { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1835         \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1836         { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1837     }
1838     {
1839         \bool_set_true:N \l__zrefclever_sort_decided_bool
1840         \int_compare:nNnTF
1841         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1842         >
1843         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1844         { \sort_return_swapped: }
1845         { \sort_return_same: }
1846     }
1847 }
1848 }
1849 }
1850 }
1851 }

```

(End definition for `__zrefclever_sort_default_same_type:nn`.)

`__zrefclever_sort_default_different_types:nn`

```

\__zrefclever_sort_default_different_types:nn {<label a>} {<label b>}

```

```

1852 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
1853 {

```

Retrieve sort priorities for `<label a>` and `<label b>`. `\l__zrefclever_typesort_seq` was stored in reverse sequence, and we compute the sort priorities in the negative range, so that we can implicitly rely on ‘0’ being the “last value”.

```

1854     \int_zero:N \l__zrefclever_sort_prior_a_int
1855     \int_zero:N \l__zrefclever_sort_prior_b_int
1856     \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1857     {
1858         \tl_if_eq:nnTF {##2} {{othertypes}}
1859         {
1860             \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
1861             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1862             \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
1863             { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1864         }
1865         {
1866             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1867             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1868             {
1869                 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
1870                 { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1871             }
1872         }
1873     }

```

Then do the actual sorting.

```

1874     \bool_if:nTF
1875     {
1876         \int_compare_p:nNn
1877         { \l__zrefclever_sort_prior_a_int } <

```

```

1878         { \l__zrefclever_sort_prior_b_int }
1879     }
1880     { \sort_return_same: }
1881     {
1882         \bool_if:nTF
1883         {
1884             \int_compare_p:nNn
1885             { \l__zrefclever_sort_prior_a_int } >
1886             { \l__zrefclever_sort_prior_b_int }
1887         }
1888         { \sort_return_swapped: }
1889         {
1890             % Sort priorities are equal: the type that occurs first in
1891             % ‘labels’, as given by the user, is kept (or brought) forward.
1892             \seq_map_inline:Nn \l__zrefclever_label_types_seq
1893             {
1894                 \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1895                 { \seq_map_break:n { \sort_return_same: } }
1896                 {
1897                     \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
1898                     { \seq_map_break:n { \sort_return_swapped: } }
1899                 }
1900             }
1901         }
1902     }
1903 }

```

(End definition for `__zrefclever_sort_default_different_types:nn`.)

`__zrefclever_sort_page:nn` The sorting function for sorting of defined labels for references to “page”. This function is expected to be called within the sorting loop of `__zrefclever_sort_labels:` and receives the pair of labels being considered for a change of order or not. It should *always* “return” either `\sort_return_same:` or `\sort_return_swapped:`. Compared to the sorting of default labels, this is a piece of cake (thanks to `abspage`).

```

\__zrefclever_sort_page:nn {<label a>} {<label b>}

1904 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
1905 {
1906     \int_compare:nNnTF
1907     { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
1908     >
1909     { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
1910     { \sort_return_swapped: }
1911     { \sort_return_same: }
1912 }

```

(End definition for `__zrefclever_sort_page:nn`.)

8 Typesetting

“Typesetting” the reference, which here includes the parsing of the labels and eventual compression of labels in sequence into ranges, is definitely the “crux” of `zref-clever`. This

because we process the label set as a stack, in a single pass, and hence “parsing”, “compressing”, and “typesetting” must be decided upon at the same time, making it difficult to slice the job into more specific and self-contained tasks. So, do bear this in mind before you curse me for the length of some of the functions below, or before a more orthodox “docstripper” complains about me not sticking to code commenting conventions to keep the code more readable in the .dtx file.

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

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

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

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

`\l_zrefclever_next_is_same_bool` when their values are actually equal.

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

Variables

Auxiliary variables for `_zrefclever_typeset_refs`: main stack control.

```
\l_zrefclever_typeset_labels_seq
\l_zrefclever_typeset_last_bool
\l_zrefclever_last_of_type_bool
1913 \seq_new:N \l__zrefclever_typeset_labels_seq
1914 \bool_new:N \l__zrefclever_typeset_last_bool
1915 \bool_new:N \l__zrefclever_last_of_type_bool
```

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

Auxiliary variables for `_zrefclever_typeset_refs`: main counters.

```
\l_zrefclever_type_count_int
\l_zrefclever_label_count_int
1916 \int_new:N \l__zrefclever_type_count_int
1917 \int_new:N \l__zrefclever_label_count_int
```

(End definition for `\l_zrefclever_type_count_int` and `\l__zrefclever_label_count_int`.)

Auxiliary variables for `_zrefclever_typeset_refs`: main “queue” control and storage.

```
\l__zrefclever_label_a_tl
\l__zrefclever_label_b_tl
\l_zrefclever_typeset_queue_prev_tl
\l_zrefclever_typeset_queue_curr_tl
\l_zrefclever_type_first_label_tl
\l_zrefclever_type_first_label_type_tl
1918 \tl_new:N \l__zrefclever_label_a_tl
1919 \tl_new:N \l__zrefclever_label_b_tl
1920 \tl_new:N \l__zrefclever_typeset_queue_prev_tl
1921 \tl_new:N \l__zrefclever_typeset_queue_curr_tl
1922 \tl_new:N \l__zrefclever_type_first_label_tl
1923 \tl_new:N \l__zrefclever_type_first_label_type_tl
```

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

Auxiliary variables for `_zrefclever_typeset_refs`: type name handling.

```
\l_zrefclever_type_name_tl
\l_zrefclever_name_in_link_bool
\l_zrefclever_name_format_tl
\l_zrefclever_name_format_fallback_tl
\l_zrefclever_type_name_gender_tl
1924 \tl_new:N \l__zrefclever_type_name_tl
1925 \bool_new:N \l__zrefclever_name_in_link_bool
1926 \tl_new:N \l__zrefclever_name_format_tl
1927 \tl_new:N \l__zrefclever_name_format_fallback_tl
1928 \tl_new:N \l__zrefclever_type_name_gender_tl
```

(End definition for `\l_zrefclever_type_name_tl` and others.)

```

\l_zrefclever_range_count_int
\l_zrefclever_range_same_count_int
\l_zrefclever_range_beg_label_tl
\l_zrefclever_next_maybe_range_bool
\l_zrefclever_next_is_same_bool

```

Auxiliary variables for `__zrefclever_typeset_refs`: range handling.

```

1929 \int_new:N \l__zrefclever_range_count_int
1930 \int_new:N \l__zrefclever_range_same_count_int
1931 \tl_new:N \l__zrefclever_range_beg_label_tl
1932 \bool_new:N \l_zrefclever_next_maybe_range_bool
1933 \bool_new:N \l__zrefclever_next_is_same_bool

```

(End definition for `\l_zrefclever_range_count_int` and others.)

```

\l__zrefclever_tpairsep_tl
\l__zrefclever_tlistsep_tl
\l__zrefclever_tlastsep_tl
\l__zrefclever_namesep_tl
\l__zrefclever_pairsep_tl
\l__zrefclever_listsep_tl
\l__zrefclever_lastsep_tl
\l__zrefclever_rangesep_tl
\l_zrefclever_refpre_out_tl
\l_zrefclever_refpos_out_tl
\l_zrefclever_refpre_in_tl
\l_zrefclever_refpos_in_tl
\l__zrefclever_namefont_tl
\l_zrefclever_reffont_out_tl
\l_zrefclever_reffont_in_tl

```

Auxiliary variables for `__zrefclever_typeset_refs`: separators, refpre/pos and font options.

```

1934 \tl_new:N \l__zrefclever_tpairsep_tl
1935 \tl_new:N \l__zrefclever_tlistsep_tl
1936 \tl_new:N \l__zrefclever_tlastsep_tl
1937 \tl_new:N \l__zrefclever_namesep_tl
1938 \tl_new:N \l__zrefclever_pairsep_tl
1939 \tl_new:N \l__zrefclever_listsep_tl
1940 \tl_new:N \l__zrefclever_lastsep_tl
1941 \tl_new:N \l__zrefclever_rangesep_tl
1942 \tl_new:N \l_zrefclever_refpre_out_tl
1943 \tl_new:N \l_zrefclever_refpos_out_tl
1944 \tl_new:N \l_zrefclever_refpre_in_tl
1945 \tl_new:N \l_zrefclever_refpos_in_tl
1946 \tl_new:N \l__zrefclever_namefont_tl
1947 \tl_new:N \l_zrefclever_reffont_out_tl
1948 \tl_new:N \l_zrefclever_reffont_in_tl

```

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

Main functions

`__zrefclever_typeset_refs`: Main typesetting function for `\zceref`.

```

1949 \cs_new_protected:Npn \__zrefclever_typeset_refs:
1950 {
1951   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
1952   \l__zrefclever_zceref_labels_seq
1953   \tl_clear:N \l_zrefclever_typeset_queue_prev_tl
1954   \tl_clear:N \l_zrefclever_typeset_queue_curr_tl
1955   \tl_clear:N \l__zrefclever_type_first_label_tl
1956   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1957   \tl_clear:N \l__zrefclever_range_beg_label_tl
1958   \int_zero:N \l_zrefclever_label_count_int
1959   \int_zero:N \l_zrefclever_type_count_int
1960   \int_zero:N \l_zrefclever_range_count_int
1961   \int_zero:N \l_zrefclever_range_same_count_int
1962
1963   % Get type block options (not type-specific).
1964   \__zrefclever_get_ref_string:nN { tpairsep }
1965   \l__zrefclever_tpairsep_tl
1966   \__zrefclever_get_ref_string:nN { tlistsep }
1967   \l__zrefclever_tlistsep_tl
1968   \__zrefclever_get_ref_string:nN { tlastsep }
1969   \l__zrefclever_tlastsep_tl
1970

```



```

1971 % Process label stack.
1972 \bool_set_false:N \l__zrefclever_typeset_last_bool
1973 \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1974 {
1975   \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
1976   \l__zrefclever_label_a_tl
1977   \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
1978   {
1979     \tl_clear:N \l__zrefclever_label_b_tl
1980     \bool_set_true:N \l__zrefclever_typeset_last_bool
1981   }
1982   {
1983     \seq_get_left:NN \l__zrefclever_typeset_labels_seq
1984     \l__zrefclever_label_b_tl
1985   }
1986 }
1987 \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1988 {
1989   \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1990   \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1991 }
1992 {
1993   \__zrefclever_def_extract:Nvnn \l__zrefclever_label_type_a_tl
1994   \l__zrefclever_label_a_tl { zc@type } { \c_empty_tl }
1995   \__zrefclever_def_extract:Nvnn \l__zrefclever_label_type_b_tl
1996   \l__zrefclever_label_b_tl { zc@type } { \c_empty_tl }
1997 }
1998
1999 % First, we establish whether the "current label" (i.e. 'a') is the
2000 % last one of its type. This can happen because the "next label"
2001 % (i.e. 'b') is of a different type (or different definition status),
2002 % or because we are at the end of the list.
2003 \bool_if:NTF \l__zrefclever_typeset_last_bool
2004 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
2005 {
2006   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
2007   {
2008     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
2009     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
2010     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
2011   }
2012   {
2013     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
2014     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
2015     {
2016       % Neither is undefined, we must check the types.
2017       \bool_if:nTF
2018       {
2019         % Both empty: same "type".
2020         \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
2021         \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
2022       }
2023       { \bool_set_false:N \l__zrefclever_last_of_type_bool }
2024       {

```

```

2025         \bool_if:nTF
2026         {
2027             % Neither empty: compare types.
2028             ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl
2029             &&
2030             ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
2031         }
2032         {
2033             \tl_if_eq:NNTF
2034             \l__zrefclever_label_type_a_tl
2035             \l__zrefclever_label_type_b_tl
2036             {
2037                 \bool_set_false:N
2038                 \l__zrefclever_last_of_type_bool
2039             }
2040             {
2041                 \bool_set_true:N
2042                 \l__zrefclever_last_of_type_bool
2043             }
2044         }
2045         % One empty, the other not: different "types".
2046         {
2047             \bool_set_true:N
2048             \l__zrefclever_last_of_type_bool
2049         }
2050     }
2051 }
2052 }
2053 }
2054
2055 % Handle warnings in case of reference or type undefined.
2056 \zref@refused { \l__zrefclever_label_a_tl }
2057 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
2058 {}
2059 {
2060     \tl_if_empty:NT \l__zrefclever_label_type_a_tl
2061     {
2062         \msg_warning:nmx { zref-clever } { missing-type }
2063         { \l__zrefclever_label_a_tl }
2064     }
2065 }
2066
2067 % Get type-specific separators, refpre/pos and font options, once per
2068 % type.
2069 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
2070 {
2071     \__zrefclever_get_ref_string:nN { namesep      }
2072     \l__zrefclever_namesep_tl
2073     \__zrefclever_get_ref_string:nN { rangesep     }
2074     \l__zrefclever_rangesep_tl
2075     \__zrefclever_get_ref_string:nN { pairsep      }
2076     \l__zrefclever_pairsep_tl
2077     \__zrefclever_get_ref_string:nN { listsep      }
2078     \l__zrefclever_listsep_tl

```

```

2079         \__zrefclever_get_ref_string:nN { lastsep      }
2080         \l__zrefclever_lastsep_tl
2081         \__zrefclever_get_ref_string:nN { refpre       }
2082         \l__zrefclever_refpre_out_tl
2083         \__zrefclever_get_ref_string:nN { refpos       }
2084         \l__zrefclever_refpos_out_tl
2085         \__zrefclever_get_ref_string:nN { refpre-in    }
2086         \l__zrefclever_refpre_in_tl
2087         \__zrefclever_get_ref_string:nN { refpos-in    }
2088         \l__zrefclever_refpos_in_tl
2089         \__zrefclever_get_ref_font:nN   { namefont     }
2090         \l__zrefclever_namefont_tl
2091         \__zrefclever_get_ref_font:nN   { reffont      }
2092         \l__zrefclever_reffont_out_tl
2093         \__zrefclever_get_ref_font:nN   { reffont-in   }
2094         \l__zrefclever_reffont_in_tl
2095     }
2096
2097     % Here we send this to a couple of auxiliary functions.
2098     \bool_if:NTF \l__zrefclever_last_of_type_bool
2099     % There exists no next label of the same type as the current.
2100     { \__zrefclever_typeset_refs_last_of_type: }
2101     % There exists a next label of the same type as the current.
2102     { \__zrefclever_typeset_refs_not_last_of_type: }
2103 }
2104 }

```

(End definition for `__zrefclever_typeset_refs:`.)

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

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

```

2105 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
2106 {
2107     % Process the current label to the current queue.
2108     \int_case:nnF { \l__zrefclever_label_count_int }
2109     {
2110         % It is the last label of its type, but also the first one, and that's
2111         % what matters here: just store it.
2112         { 0 }
2113         {
2114             \tl_set:NV \l__zrefclever_type_first_label_tl
2115             \l__zrefclever_label_a_tl
2116             \tl_set:NV \l__zrefclever_type_first_label_type_tl
2117             \l__zrefclever_label_type_a_tl
2118         }
2119     }

```

```

2120 % The last is the second: we have a pair (if not repeated).
2121 { 1 }
2122 {
2123   \int_compare:nNnF { \l__zrefclever_range_same_count_int } = { 1 }
2124   {
2125     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2126     {
2127       \exp_not:V \l__zrefclever_pairsep_tl
2128       \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2129     }
2130   }
2131 }
2132 }
2133 % Last is third or more of its type: without repetition, we'd have the
2134 % last element on a list, but control for possible repetition.
2135 {
2136   \int_case:nnF { \l__zrefclever_range_count_int }
2137   {
2138     % There was no range going on.
2139     { 0 }
2140     {
2141       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2142       {
2143         \exp_not:V \l__zrefclever_lastsep_tl
2144         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2145       }
2146     }
2147     % Last in the range is also the second in it.
2148     { 1 }
2149     {
2150       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2151       {
2152         % We know 'range_beg_label' is not empty, since this is the
2153         % second element in the range, but the third or more in the
2154         % type list.
2155         \exp_not:V \l__zrefclever_listsep_tl
2156         \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
2157         \int_compare:nNnF
2158         { \l__zrefclever_range_same_count_int } = { 1 }
2159         {
2160           \exp_not:V \l__zrefclever_lastsep_tl
2161           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2162         }
2163       }
2164     }
2165   }
2166   % Last in the range is third or more in it.
2167   {
2168     \int_case:nnF
2169     {
2170       \l__zrefclever_range_count_int -
2171       \l__zrefclever_range_same_count_int
2172     }
2173     {

```

```

2174 % Repetition, not a range.
2175 { 0 }
2176 {
2177 % If 'range_beg_label' is empty, it means it was also the
2178 % first of the type, and hence was already handled.
2179 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2180 {
2181 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2182 {
2183 \exp_not:V \l__zrefclever_lastsep_tl
2184 \__zrefclever_get_ref:V
2185 \l__zrefclever_range_beg_label_tl
2186 }
2187 }
2188 }
2189 % A 'range', but with no skipped value, treat as list.
2190 { 1 }
2191 {
2192 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2193 {
2194 % Ditto.
2195 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2196 {
2197 \exp_not:V \l__zrefclever_listsep_tl
2198 \__zrefclever_get_ref:V
2199 \l__zrefclever_range_beg_label_tl
2200 }
2201 \exp_not:V \l__zrefclever_lastsep_tl
2202 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2203 }
2204 }
2205 }
2206 {
2207 % An actual range.
2208 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2209 {
2210 % Ditto.
2211 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2212 {
2213 \exp_not:V \l__zrefclever_lastsep_tl
2214 \__zrefclever_get_ref:V
2215 \l__zrefclever_range_beg_label_tl
2216 }
2217 \exp_not:V \l__zrefclever_rangesep_tl
2218 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2219 }
2220 }
2221 }
2222 }
2223
2224 % Handle "range" option. The idea is simple: if the queue is not empty,
2225 % we replace it with the end of the range (or pair). We can still
2226 % retrieve the end of the range from 'label_a' since we know to be
2227 % processing the last label of its type at this point.

```

```

2228 \bool_if:NT \l__zrefclever_typeset_range_bool
2229 {
2230   \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2231   {
2232     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2233     { }
2234     {
2235       \msg_warning:nxx { zref-clever } { single-element-range }
2236       { \l__zrefclever_type_first_label_type_tl }
2237     }
2238   }
2239   {
2240     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
2241     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2242     { }
2243     {
2244       \__zrefclever_labels_in_sequence:nn
2245       { \l__zrefclever_type_first_label_tl }
2246       { \l__zrefclever_label_a_tl }
2247     }
2248     \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
2249     {
2250       \bool_if:NTF \l__zrefclever_next_maybe_range_bool
2251       { \exp_not:V \l__zrefclever_pairsep_tl }
2252       { \exp_not:V \l__zrefclever_rangesep_tl }
2253       \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2254     }
2255   }
2256 }
2257
2258 % Now that the type block is finished, we can add the name and the first
2259 % ref to the queue. Also, if "typeset" option is not "both", handle it
2260 % here as well.
2261 \__zrefclever_type_name_setup:
2262 \bool_if:nTF
2263 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
2264 {
2265   \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
2266   { \__zrefclever_get_ref_first: }
2267 }
2268 {
2269   \bool_if:nTF
2270   { \l__zrefclever_typeset_ref_bool }
2271   {
2272     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
2273     { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
2274   }
2275   {
2276     \bool_if:nTF
2277     { \l__zrefclever_typeset_name_bool }
2278     {
2279       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
2280       {
2281         \bool_if:NTF \l__zrefclever_name_in_link_bool

```

```

2282         {
2283             \exp_not:N \group_begin:
2284             \exp_not:V \l__zrefclever_namefont_tl
2285             % It's two '@s', but escaped for DocStrip.
2286             \exp_not:N \hyper@link
2287             {
2288                 \__zrefclever_extract_url_unexp:V
2289                 \l__zrefclever_type_first_label_tl
2290             }
2291             {
2292                 \__zrefclever_extract_unexp:Vnn
2293                 \l__zrefclever_type_first_label_tl
2294                 { anchor } { }
2295             }
2296             { \exp_not:V \l__zrefclever_type_name_tl }
2297             \exp_not:N \group_end:
2298         }
2299         {
2300             \exp_not:N \group_begin:
2301             \exp_not:V \l__zrefclever_namefont_tl
2302             \exp_not:V \l__zrefclever_type_name_tl
2303             \exp_not:N \group_end:
2304         }
2305     }
2306 }
2307 {
2308     % Logically, this case would correspond to "typeset=none", but
2309     % it should not occur, given that the options are set up to
2310     % typeset either "ref" or "name". Still, leave here a
2311     % sensible fallback, equal to the behavior of "both".
2312     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
2313         { \__zrefclever_get_ref_first: }
2314 }
2315 }
2316 }
2317
2318 % Typeset the previous type, if there is one.
2319 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
2320 {
2321     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
2322     { \l__zrefclever_tlistsep_tl }
2323     \l__zrefclever_typeset_queue_prev_tl
2324 }
2325
2326 % Wrap up loop, or prepare for next iteration.
2327 \bool_if:NTF \l__zrefclever_typeset_last_bool
2328 {
2329     % We are finishing, typeset the current queue.
2330     \int_case:nnF { \l__zrefclever_type_count_int }
2331     {
2332         % Single type.
2333         { 0 }
2334         { \l__zrefclever_typeset_queue_curr_tl }
2335         % Pair of types.

```

```

2336         { 1 }
2337     {
2338         \l__zrefclever_tpairsep_tl
2339         \l__zrefclever_typeset_queue_curr_tl
2340     }
2341 }
2342 {
2343     % Last in list of types.
2344     \l__zrefclever_tlastsep_tl
2345     \l__zrefclever_typeset_queue_curr_tl
2346 }
2347 % And nudge in case of multitype reference.
2348 \bool_lazy_all:nT
2349 {
2350     { \l__zrefclever_nudge_enabled_bool }
2351     { \l__zrefclever_nudge_multitype_bool }
2352     { \int_compare_p:nNn { \l__zrefclever_type_count_int } > { 1 } }
2353 }
2354 { \msg_warning:nn { zref-clever } { nudge-multitype } }
2355 }
2356 {
2357     % There are further labels, set variables for next iteration.
2358     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
2359     \l__zrefclever_typeset_queue_curr_tl
2360     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
2361     \tl_clear:N \l__zrefclever_type_first_label_tl
2362     \tl_clear:N \l__zrefclever_type_first_label_type_tl
2363     \tl_clear:N \l__zrefclever_range_beg_label_tl
2364     \int_zero:N \l__zrefclever_label_count_int
2365     \int_incr:N \l__zrefclever_type_count_int
2366     \int_zero:N \l__zrefclever_range_count_int
2367     \int_zero:N \l__zrefclever_range_same_count_int
2368 }
2369 }

```

(End definition for __zrefclever_typeset_refs_last_of_type:.)

__zrefclever_typeset_refs_not_last_of_type:

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

```

2370 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
2371 {
2372     % Signal if next label may form a range with the current one (only
2373     % considered if compression is enabled in the first place).
2374     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
2375     \bool_set_false:N \l__zrefclever_next_is_same_bool
2376     \bool_if:NT \l__zrefclever_typeset_compress_bool
2377     {
2378         \zref@ifrefundefined { \l__zrefclever_label_a_tl }
2379         { }
2380         {
2381             \__zrefclever_labels_in_sequence:nn
2382             { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
2383         }
2384     }
2385 }

```



```

2386 % Process the current label to the current queue.
2387 \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
2388 {
2389   % Current label is the first of its type (also not the last, but it
2390   % doesn't matter here): just store the label.
2391   \tl_set:NV \l__zrefclever_type_first_label_tl
2392     \l__zrefclever_label_a_tl
2393   \tl_set:NV \l__zrefclever_type_first_label_type_tl
2394     \l__zrefclever_label_type_a_tl
2395
2396   % If the next label may be part of a range, we set 'range_beg_label'
2397   % to "empty" (we deal with it as the "first", and must do it there, to
2398   % handle hyperlinking), but also step the range counters.
2399   \bool_if:NT \l__zrefclever_next_maybe_range_bool
2400   {
2401     \tl_clear:N \l__zrefclever_range_beg_label_tl
2402     \int_incr:N \l__zrefclever_range_count_int
2403     \bool_if:NT \l__zrefclever_next_is_same_bool
2404       { \int_incr:N \l__zrefclever_range_same_count_int }
2405   }
2406 }
2407 {
2408   % Current label is neither the first (nor the last) of its type.
2409   \bool_if:NTF \l__zrefclever_next_maybe_range_bool
2410   {
2411     % Starting, or continuing a range.
2412     \int_compare:nNnTF
2413       { \l__zrefclever_range_count_int } = { 0 }
2414     {
2415       % There was no range going, we are starting one.
2416       \tl_set:NV \l__zrefclever_range_beg_label_tl
2417         \l__zrefclever_label_a_tl
2418       \int_incr:N \l__zrefclever_range_count_int
2419       \bool_if:NT \l__zrefclever_next_is_same_bool
2420         { \int_incr:N \l__zrefclever_range_same_count_int }
2421     }
2422     {
2423       % Second or more in the range, but not the last.
2424       \int_incr:N \l__zrefclever_range_count_int
2425       \bool_if:NT \l__zrefclever_next_is_same_bool
2426         { \int_incr:N \l__zrefclever_range_same_count_int }
2427     }
2428   }
2429   {
2430     % Next element is not in sequence: there was no range, or we are
2431     % closing one.
2432     \int_case:nnF { \l__zrefclever_range_count_int }
2433     {
2434       % There was no range going on.
2435       { 0 }
2436       {
2437         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2438           {
2439             \exp_not:V \l__zrefclever_listsep_tl

```

```

2440         \l__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2441     }
2442 }
2443 % Last is second in the range: if 'range_same_count' is also
2444 % '1', it's a repetition (drop it), otherwise, it's a "pair
2445 % within a list", treat as list.
2446 { 1 }
2447 {
2448     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2449     {
2450         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2451         {
2452             \exp_not:V \l__zrefclever_listsep_tl
2453             \l__zrefclever_get_ref:V
2454             \l__zrefclever_range_beg_label_tl
2455         }
2456         \int_compare:nNnF
2457         { \l__zrefclever_range_same_count_int } = { 1 }
2458         {
2459             \exp_not:V \l__zrefclever_listsep_tl
2460             \l__zrefclever_get_ref:V
2461             \l__zrefclever_label_a_tl
2462         }
2463     }
2464 }
2465 }
2466 {
2467 % Last is third or more in the range: if 'range_count' and
2468 % 'range_same_count' are the same, its a repetition (drop it),
2469 % if they differ by '1', its a list, if they differ by more,
2470 % it is a real range.
2471 \int_case:nnF
2472 {
2473     \l__zrefclever_range_count_int -
2474     \l__zrefclever_range_same_count_int
2475 }
2476 {
2477     { 0 }
2478     {
2479         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2480         {
2481             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2482             {
2483                 \exp_not:V \l__zrefclever_listsep_tl
2484                 \l__zrefclever_get_ref:V
2485                 \l__zrefclever_range_beg_label_tl
2486             }
2487         }
2488     }
2489     { 1 }
2490     {
2491         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2492         {
2493             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl

```

```

2494         {
2495             \exp_not:V \l__zrefclever_listsep_tl
2496             \__zrefclever_get_ref:V
2497             \l__zrefclever_range_beg_label_tl
2498         }
2499         \exp_not:V \l__zrefclever_listsep_tl
2500         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2501     }
2502 }
2503 }
2504 {
2505     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2506     {
2507         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2508         {
2509             \exp_not:V \l__zrefclever_listsep_tl
2510             \__zrefclever_get_ref:V
2511             \l__zrefclever_range_beg_label_tl
2512         }
2513         \exp_not:V \l__zrefclever_rangesep_tl
2514         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2515     }
2516 }
2517 }
2518 % Reset counters.
2519 \int_zero:N \l__zrefclever_range_count_int
2520 \int_zero:N \l__zrefclever_range_same_count_int
2521 }
2522 }
2523 % Step label counter for next iteration.
2524 \int_incr:N \l__zrefclever_label_count_int
2525 }

```

(End definition for `__zrefclever_typeset_refs_not_last_of_type:.`)

Aux functions

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

use the `n` signature jargon). We also need to prevent premature expansion of material that can't be expanded at this point (e.g. grouping, `\zref@default` or `\hyper@@link`). In a nutshell, the job of these two functions is putting the pieces in place, but with proper expansion control.

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

```

2526 \cs_new_protected:Npn \_zrefclever_ref_default:
2527   { \zref@default }
2528 \cs_new_protected:Npn \_zrefclever_name_default:
2529   { \zref@default }

```

(End definition for `_zrefclever_ref_default:` and `_zrefclever_name_default:.`)

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

```

      \_zrefclever_get_ref:n {<label>}

2530 \cs_new:Npn \_zrefclever_get_ref:n #1
2531 {
2532   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
2533   {
2534     \bool_if:nTF
2535     {
2536       \l__zrefclever_use_hyperref_bool &&
2537       ! \l__zrefclever_link_star_bool
2538     }
2539     {
2540       \exp_not:N \group_begin:
2541       \exp_not:V \l__zrefclever_reffont_out_tl
2542       \exp_not:V \l__zrefclever_refpre_out_tl
2543       \exp_not:N \group_begin:
2544       \exp_not:V \l__zrefclever_reffont_in_tl
2545       % It's two '@s', but escaped for DocStrip.
2546       \exp_not:N \hyper@@link
2547       { \_zrefclever_extract_url_unexp:n {#1} }
2548       { \_zrefclever_extract_unexp:nnn {#1} { anchor } { } }
2549       {
2550         \exp_not:V \l__zrefclever_refpre_in_tl
2551         \_zrefclever_extract_unexp:nvn {#1}
2552         { \l__zrefclever_ref_property_tl } { }
2553         \exp_not:V \l__zrefclever_refpos_in_tl
2554       }
2555       \exp_not:N \group_end:
2556       \exp_not:V \l__zrefclever_refpos_out_tl
2557       \exp_not:N \group_end:
2558     }
2559     {
2560       \exp_not:N \group_begin:
2561       \exp_not:V \l__zrefclever_reffont_out_tl

```

```

2562         \exp_not:V \l__zrefclever_refpre_out_tl
2563         \exp_not:N \group_begin:
2564         \exp_not:V \l__zrefclever_reffont_in_tl
2565         \exp_not:V \l__zrefclever_refpre_in_tl
2566         \__zrefclever_extract_unexp:nvn {#1}
2567         { \l__zrefclever_ref_property_tl } { }
2568         \exp_not:V \l__zrefclever_refpos_in_tl
2569         \exp_not:N \group_end:
2570         \exp_not:V \l__zrefclever_refpos_out_tl
2571         \exp_not:N \group_end:
2572     }
2573 }
2574 { \__zrefclever_ref_default: }
2575 }
2576 \cs_generate_variant:Nn \__zrefclever_get_ref:n { V }

```

(End definition for __zrefclever_get_ref:n.)

__zrefclever_get_ref_first: Handles a complete reference block for the first label of its type to be accumulated in the “queue”, including “pre” and “pos” elements, hyperlinking, and the reference type “name”. It does not receive arguments, but relies on being called in the appropriate place in __zrefclever_typeset_refs_last_of_type: where a number of variables are expected to be appropriately set for it to consume. Prominently among those is \l__zrefclever_type_first_label_tl, but it also expected to be called right after __zrefclever_type_name_setup: which sets \l__zrefclever_type_name_tl and \l__zrefclever_name_in_link_bool which it uses.

```

2577 \cs_new:Npn \__zrefclever_get_ref_first:
2578 {
2579     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2580     { \__zrefclever_ref_default: }
2581     {
2582         \bool_if:NTF \l__zrefclever_name_in_link_bool
2583         {
2584             \zref@ifrefcontainsprop
2585             { \l__zrefclever_type_first_label_tl }
2586             { \l__zrefclever_ref_property_tl }
2587             {
2588                 % It's two '@s', but escaped for DocStrip.
2589                 \exp_not:N \hyper@@link
2590                 {
2591                     \__zrefclever_extract_url_unexp:V
2592                     \l__zrefclever_type_first_label_tl
2593                 }
2594                 {
2595                     \__zrefclever_extract_unexp:Vnn
2596                     \l__zrefclever_type_first_label_tl { anchor } { }
2597                 }
2598                 {
2599                     \exp_not:N \group_begin:
2600                     \exp_not:V \l__zrefclever_namefont_tl
2601                     \exp_not:V \l__zrefclever_type_name_tl
2602                     \exp_not:N \group_end:
2603                     \exp_not:V \l__zrefclever_namesep_tl
2604                     \exp_not:N \group_begin:

```

```

2605         \exp_not:V \l__zrefclever_reffont_out_tl
2606         \exp_not:V \l__zrefclever_refpre_out_tl
2607         \exp_not:N \group_begin:
2608         \exp_not:V \l__zrefclever_reffont_in_tl
2609         \exp_not:V \l__zrefclever_refpre_in_tl
2610         \__zrefclever_extract_unexp:Vvn
2611         \l__zrefclever_type_first_label_tl
2612         { \l__zrefclever_ref_property_tl } { }
2613         \exp_not:V \l__zrefclever_refpos_in_tl
2614         \exp_not:N \group_end:
2615         % hyperlink makes it's own group, we'd like to close the
2616         % 'refpre-out' group after 'refpos-out', but... we close
2617         % it here, and give the trailing 'refpos-out' its own
2618         % group. This will result that formatting given to
2619         % 'refpre-out' will not reach 'refpos-out', but I see no
2620         % alternative, and this has to be handled specially.
2621         \exp_not:N \group_end:
2622     }
2623     \exp_not:N \group_begin:
2624     % Ditto: special treatment.
2625     \exp_not:V \l__zrefclever_reffont_out_tl
2626     \exp_not:V \l__zrefclever_refpos_out_tl
2627     \exp_not:N \group_end:
2628 }
2629 {
2630     \exp_not:N \group_begin:
2631     \exp_not:V \l__zrefclever_namefont_tl
2632     \exp_not:V \l__zrefclever_type_name_tl
2633     \exp_not:N \group_end:
2634     \exp_not:V \l__zrefclever_namesep_tl
2635     \__zrefclever_ref_default:
2636 }
2637 }
2638 {
2639     \tl_if_empty:NTF \l__zrefclever_type_name_tl
2640     {
2641         \__zrefclever_name_default:
2642         \exp_not:V \l__zrefclever_namesep_tl
2643     }
2644     {
2645         \exp_not:N \group_begin:
2646         \exp_not:V \l__zrefclever_namefont_tl
2647         \exp_not:V \l__zrefclever_type_name_tl
2648         \exp_not:N \group_end:
2649         \exp_not:V \l__zrefclever_namesep_tl
2650     }
2651     \zref@ifrefcontainsprop
2652     { \l__zrefclever_type_first_label_tl }
2653     { \l__zrefclever_ref_property_tl }
2654     {
2655         \bool_if:nTF
2656         {
2657             \l__zrefclever_use_hyperref_bool &&
2658             ! \l__zrefclever_link_star_bool

```

```

2659     }
2660     {
2661         \exp_not:N \group_begin:
2662         \exp_not:V \l__zrefclever_reffont_out_tl
2663         \exp_not:V \l__zrefclever_refpre_out_tl
2664         \exp_not:N \group_begin:
2665         \exp_not:V \l__zrefclever_reffont_in_tl
2666         % It's two '@s', but escaped for DocStrip.
2667         \exp_not:N \hyper@@link
2668         {
2669             \__zrefclever_extract_url_unexp:V
2670             \l__zrefclever_type_first_label_tl
2671         }
2672         {
2673             \__zrefclever_extract_unexp:Vnn
2674             \l__zrefclever_type_first_label_tl { anchor } { }
2675         }
2676         {
2677             \exp_not:V \l__zrefclever_refpre_in_tl
2678             \__zrefclever_extract_unexp:Vnn
2679             \l__zrefclever_type_first_label_tl
2680             { \l__zrefclever_ref_property_tl } { }
2681             \exp_not:V \l__zrefclever_refpos_in_tl
2682         }
2683         \exp_not:N \group_end:
2684         \exp_not:V \l__zrefclever_refpos_out_tl
2685         \exp_not:N \group_end:
2686     }
2687     {
2688         \exp_not:N \group_begin:
2689         \exp_not:V \l__zrefclever_reffont_out_tl
2690         \exp_not:V \l__zrefclever_refpre_out_tl
2691         \exp_not:N \group_begin:
2692         \exp_not:V \l__zrefclever_reffont_in_tl
2693         \exp_not:V \l__zrefclever_refpre_in_tl
2694         \__zrefclever_extract_unexp:Vnn
2695         \l__zrefclever_type_first_label_tl
2696         { \l__zrefclever_ref_property_tl } { }
2697         \exp_not:V \l__zrefclever_refpos_in_tl
2698         \exp_not:N \group_end:
2699         \exp_not:V \l__zrefclever_refpos_out_tl
2700         \exp_not:N \group_end:
2701     }
2702     }
2703     { \__zrefclever_ref_default: }
2704   }
2705 }
2706 }

```

(End definition for `__zrefclever_get_ref_first:.`)

`__zrefclever_type_name_setup:` Auxiliary function to `__zrefclever_typeset_refs_last_of_type:.` It is responsible for setting the type name variable `\l__zrefclever_type_name_tl` and `\l__zrefclever_name_in_link_bool`. If a type name can't be found, `\l__zrefclever_type_name_tl` is cleared. The function takes no arguments, but is expected to be called

in `__zrefclever_typeset_refs_last_of_type:` right before `__zrefclever_get_ref_first:`, which is the main consumer of the variables it sets, though not the only one (and hence this cannot be moved into `__zrefclever_get_ref_first:` itself). It also expects a number of relevant variables to have been appropriately set, and which it uses, prominently `\l__zrefclever_type_first_label_type_tl`, but also the queue itself in `\l__zrefclever_typeset_queue_curr_tl`, which should be “ready except for the first label”, and the type counter `\l__zrefclever_type_count_int`.

```

2707 \cs_new_protected:Npn \__zrefclever_type_name_setup:
2708 {
2709   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2710   { \tl_clear:N \l__zrefclever_type_name_tl }
2711   {
2712     \tl_if_empty:NTF \l__zrefclever_type_first_label_type_tl
2713     { \tl_clear:N \l__zrefclever_type_name_tl }
2714     {
2715       % Determine whether we should use capitalization, abbreviation,
2716       % and plural.
2717       \bool_lazy_or:nnTF
2718       { \l__zrefclever_capitalize_bool }
2719       {
2720         \l__zrefclever_capitalize_first_bool &&
2721         \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2722       }
2723       { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
2724       { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
2725       % If the queue is empty, we have a singular, otherwise, plural.
2726       \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2727       { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
2728       { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
2729       \bool_lazy_and:nnTF
2730       { \l__zrefclever_abbrev_bool }
2731       {
2732         ! \int_compare_p:nNn
2733         { \l__zrefclever_type_count_int } = { 0 } ||
2734         ! \l__zrefclever_noabbrev_first_bool
2735       }
2736       {
2737         \tl_set:NV \l__zrefclever_name_format_fallback_tl
2738         \l__zrefclever_name_format_tl
2739         \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
2740       }
2741       { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
2742
2743       % Handle number and gender nudges.
2744       \bool_if:NT \l__zrefclever_nudge_enabled_bool
2745       {
2746         \bool_if:NTF \l__zrefclever_nudge_singular_bool
2747         {
2748           \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
2749           {
2750             \msg_warning:nnx { zref-clever }
2751             { nudge-plural-when-sg }
2752             { \l__zrefclever_type_first_label_type_tl }

```



```

2753     }
2754 }
2755 {
2756   \bool_lazy_all:nT
2757   {
2758     { \l__zrefclever_nudge_comptosing_bool }
2759     { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
2760     {
2761       \int_compare_p:nNn
2762         { \l__zrefclever_label_count_int } > { 0 }
2763     }
2764   }
2765   {
2766     \msg_warning:nnx { zref-clever }
2767       { nudge-comptosing }
2768       { \l__zrefclever_type_first_label_type_tl }
2769   }
2770 }
2771 \bool_lazy_and:nnT
2772 { \l__zrefclever_nudge_gender_bool }
2773 { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
2774 {
2775   \__zrefclever_get_type_transl:xxnNF
2776   { \l__zrefclever_ref_language_tl }
2777   { \l__zrefclever_type_first_label_type_tl }
2778   { gender }
2779   \l__zrefclever_type_name_gender_tl
2780   { \tl_clear:N \l__zrefclever_type_name_gender_tl }
2781 \tl_if_eq:NNF
2782   \l__zrefclever_ref_gender_tl
2783   \l__zrefclever_type_name_gender_tl
2784   {
2785     \tl_if_empty:NTF \l__zrefclever_type_name_gender_tl
2786     {
2787       \msg_warning:nnxxx { zref-clever }
2788       { nudge-gender-not-declared-for-type }
2789       { \l__zrefclever_ref_gender_tl }
2790       { \l__zrefclever_type_first_label_type_tl }
2791       { \l__zrefclever_ref_language_tl }
2792     }
2793     {
2794       \msg_warning:nnxxxx { zref-clever }
2795       { nudge-gender-mismatch }
2796       { \l__zrefclever_type_first_label_type_tl }
2797       { \l__zrefclever_ref_gender_tl }
2798       { \l__zrefclever_type_name_gender_tl }
2799       { \l__zrefclever_ref_language_tl }
2800     }
2801   }
2802 }
2803 }
2804
2805 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
2806 {

```

```

2807 \prop_get:cVNF
2808 {
2809     l__zrefclever_type_
2810     \l__zrefclever_type_first_label_type_tl _options_prop
2811 }
2812 \l__zrefclever_name_format_tl
2813 \l__zrefclever_type_name_tl
2814 {
2815     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
2816     {
2817         \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
2818         \tl_put_left:NV \l__zrefclever_name_format_tl
2819             \l__zrefclever_ref_decl_case_tl
2820     }
2821     \__zrefclever_get_type_transl:xxxNF
2822     { \l__zrefclever_ref_language_tl }
2823     { \l__zrefclever_type_first_label_type_tl }
2824     { \l__zrefclever_name_format_tl }
2825     \l__zrefclever_type_name_tl
2826     {
2827         \tl_clear:N \l__zrefclever_type_name_tl
2828         \msg_warning:nxxx { zref-clever } { missing-name }
2829             { \l__zrefclever_name_format_tl }
2830             { \l__zrefclever_type_first_label_type_tl }
2831     }
2832 }
2833 }
2834 {
2835     \prop_get:cVNF
2836     {
2837         l__zrefclever_type_
2838         \l__zrefclever_type_first_label_type_tl _options_prop
2839     }
2840     \l__zrefclever_name_format_tl
2841     \l__zrefclever_type_name_tl
2842     {
2843         \prop_get:cVNF
2844         {
2845             l__zrefclever_type_
2846             \l__zrefclever_type_first_label_type_tl _options_prop
2847         }
2848         \l__zrefclever_name_format_fallback_tl
2849         \l__zrefclever_type_name_tl
2850         {
2851             \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
2852             {
2853                 \tl_put_left:Nn
2854                     \l__zrefclever_name_format_tl { - }
2855                 \tl_put_left:NV \l__zrefclever_name_format_tl
2856                     \l__zrefclever_ref_decl_case_tl
2857                 \tl_put_left:Nn
2858                     \l__zrefclever_name_format_fallback_tl { - }
2859                 \tl_put_left:NV
2860                     \l__zrefclever_name_format_fallback_tl

```

```

2861         \l__zrefclever_ref_decl_case_tl
2862     }
2863     \__zrefclever_get_type_transl:xxxNF
2864     { \l__zrefclever_ref_language_tl }
2865     { \l__zrefclever_type_first_label_type_tl }
2866     { \l__zrefclever_name_format_tl }
2867     \l__zrefclever_type_name_tl
2868     {
2869         \__zrefclever_get_type_transl:xxxNF
2870         { \l__zrefclever_ref_language_tl }
2871         { \l__zrefclever_type_first_label_type_tl }
2872         { \l__zrefclever_name_format_fallback_tl }
2873         \l__zrefclever_type_name_tl
2874         {
2875             \tl_clear:N \l__zrefclever_type_name_tl
2876             \msg_warning:nxxx { zref-clever }
2877             { missing-name }
2878             { \l__zrefclever_name_format_tl }
2879             { \l__zrefclever_type_first_label_type_tl }
2880         }
2881     }
2882 }
2883 }
2884 }
2885 }
2886 }
2887
2888 % Signal whether the type name is to be included in the hyperlink or not.
2889 \bool_lazy_any:nTF
2890 {
2891     { ! \l__zrefclever_use_hyperref_bool }
2892     { \l__zrefclever_link_star_bool }
2893     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
2894     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2895 }
2896 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2897 {
2898     \bool_lazy_any:nTF
2899     {
2900         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2901         {
2902             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2903             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2904         }
2905         {
2906             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2907             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2908             \l__zrefclever_typeset_last_bool &&
2909             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2910         }
2911     }
2912     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2913     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2914 }

```

2915 }

(End definition for `_zrefclever_type_name_setup:`.)

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

```
2916 \cs_new:Npn \_zrefclever_extract_url_unexp:n #1
2917 {
2918   \zref@ifpropundefined { urluse }
2919   { \_zrefclever_extract_unexp:nnn {#1} { url } { \c_empty_tl } }
2920   {
2921     \zref@ifrefcontainsprop {#1} { urluse }
2922     { \_zrefclever_extract_unexp:nnn {#1} { urluse } { \c_empty_tl } }
2923     { \_zrefclever_extract_unexp:nnn {#1} { url } { \c_empty_tl } }
2924   }
2925 }
2926 \cs_generate_variant:Nn \_zrefclever_extract_url_unexp:n { V }
```

(End definition for `_zrefclever_extract_url_unexp:n`.)

`_zrefclever_labels_in_sequence:nn` Auxiliary function to `_zrefclever_typeset_refs_not_last_of_type:`. Sets `\l__zrefclever_next_maybe_range_bool` to true if `<label b>` comes in immediate sequence from `<label a>`. And sets both `\l__zrefclever_next_maybe_range_bool` and `\l__zrefclever_next_is_same_bool` to true if the two labels are the “same” (that is, have the same counter value). These two boolean variables are the basis for all range and compression handling inside `_zrefclever_typeset_refs_not_last_of_type:`, so this function is expected to be called at its beginning, if compression is enabled.

```
\_zrefclever_labels_in_sequence:nn {<label a>} {<label b>}

2927 \cs_new_protected:Npn \_zrefclever_labels_in_sequence:nn #1#2
2928 {
2929   \_zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_a_tl
2930   {#1} { externaldocument } { \c_empty_tl }
2931   \_zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_b_tl
2932   {#2} { externaldocument } { \c_empty_tl }
2933
2934   \tl_if_eq:NNT
2935   \l__zrefclever_label_extdoc_a_tl
2936   \l__zrefclever_label_extdoc_b_tl
2937   {
2938     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2939     {
2940       \exp_args:Nxx \tl_if_eq:nnT
2941       { \_zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
2942       { \_zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
2943       {
2944         \int_compare:nNnTF
2945         { \_zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
2946         =
2947         { \_zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
2948         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
```

```

2949         {
2950             \int_compare:nNnT
2951                 { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
2952                 =
2953                 { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
2954                 {
2955                     \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2956                     \bool_set_true:N \l__zrefclever_next_is_same_bool
2957                 }
2958             }
2959         }
2960     }
2961     {
2962         \exp_args:Nxx \tl_if_eq:nnT
2963         { \__zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
2964         { \__zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
2965         {
2966             \exp_args:Nxx \tl_if_eq:nnT
2967             { \__zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
2968             { \__zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
2969             {
2970                 \int_compare:nNnTF
2971                     { \__zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
2972                     =
2973                     { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
2974                     { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2975                     {
2976                         \int_compare:nNnT
2977                             { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
2978                             =
2979                             { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
2980                             {
2981                                 \bool_set_true:N
2982                                 \l__zrefclever_next_maybe_range_bool
2983                                 \exp_args:Nxx \tl_if_eq:nnT
2984                                 {
2985                                     \__zrefclever_extract_unexp:nvn {#1}
2986                                     { l__zrefclever_ref_property_tl } { }
2987                                 }
2988                                 {
2989                                     \__zrefclever_extract_unexp:nvn {#2}
2990                                     { l__zrefclever_ref_property_tl } { }
2991                                 }
2992                                 {
2993                                     \bool_set_true:N
2994                                     \l__zrefclever_next_is_same_bool
2995                                 }
2996                             }
2997                         }
2998                     }
2999                 }
3000             }
3001         }
3002     }

```

(End definition for `_zrefclever_labels_in_sequence:nn`.)

Finally, a couple of functions for retrieving options values, according to the relevant precedence rules. They both receive an *<option>* as argument, and store the retrieved value in *<tl variable>*. Though these are mostly general functions (for a change...), they are not completely so, they rely on the current state of `\l_zrefclever_label_type_a_tl`, as set during the processing of the label stack. This could be easily generalized, of course, but I don't think it is worth it, `\l_zrefclever_label_type_a_tl` is indeed what we want in all practical cases. The difference between `_zrefclever_get_ref_string:nN` and `_zrefclever_get_ref_font:nN` is the kind of option each should be used for. `_zrefclever_get_ref_string:nN` is meant for the general options, and attempts to find values for them in all precedence levels (four plus “fall-back”). `_zrefclever_get_ref_font:nN` is intended for “font” options, which cannot be “language-specific”, thus for these we just search general options and type options.

```

\_zrefclever_get_ref_string:nN      \_zrefclever_get_ref_string:nN {<option>} {<tl variable>}
3003 \cs_new_protected:Npn \_zrefclever_get_ref_string:nN #1#2
3004 {
3005   % First attempt: general options.
3006   \prop_get:NnNF \l_zrefclever_ref_options_prop {#1} #2
3007   {
3008     % If not found, try type specific options.
3009     \bool_lazy_all:nTF
3010     {
3011       { ! \tl_if_empty_p:N \l_zrefclever_label_type_a_tl }
3012       {
3013         \prop_if_exist_p:c
3014         {
3015           l_zrefclever_type_
3016           \l_zrefclever_label_type_a_tl _options_prop
3017         }
3018       }
3019       {
3020         \prop_if_in_p:cn
3021         {
3022           l_zrefclever_type_
3023           \l_zrefclever_label_type_a_tl _options_prop
3024         }
3025         {#1}
3026       }
3027     }
3028     {
3029       \prop_get:cnN
3030       {
3031         l_zrefclever_type_
3032         \l_zrefclever_label_type_a_tl _options_prop
3033       }
3034       {#1} #2
3035     }
3036     {
3037       % If not found, try type specific translations.
3038       \_zrefclever_get_type_transl:xxnNF
3039       { \l_zrefclever_ref_language_tl }
3040       { \l_zrefclever_label_type_a_tl }

```

```

3041         {#1} #2
3042     {
3043         % If not found, try default translations.
3044         \__zrefclever_get_default_transl:xnNF
3045         { \l__zrefclever_ref_language_tl }
3046         {#1} #2
3047     {
3048         % If not found, try fallback.
3049         \__zrefclever_get_fallback_transl:nNF {#1} #2
3050     {
3051         \tl_clear:N #2
3052         \msg_warning:nnn { zref-clever }
3053         { missing-string } {#1}
3054     }
3055 }
3056 }
3057 }
3058 }
3059 }

```

(End definition for __zrefclever_get_ref_string:nN.)

```

\__zrefclever_get_ref_font:nN      \__zrefclever_get_ref_font:nN {<option>} {<tl variable>}
3060 \cs_new_protected:Npn \__zrefclever_get_ref_font:nN #1#2
3061 {
3062     % First attempt: general options.
3063     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
3064     {
3065         % If not found, try type specific options.
3066         \bool_lazy_and:nnTF
3067         { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
3068         {
3069             \prop_if_exist_p:c
3070             {
3071                 l__zrefclever_type_
3072                 \l__zrefclever_label_type_a_tl _options_prop
3073             }
3074         {
3075             \prop_get:cnNF
3076             {
3077                 l__zrefclever_type_
3078                 \l__zrefclever_label_type_a_tl _options_prop
3079             }
3080             {#1} #2
3081             { \tl_clear:N #2 }
3082         }
3083     }
3084     { \tl_clear:N #2 }
3085 }
3086 }

```

(End definition for __zrefclever_get_ref_font:nN.)

9 Compatibility

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

9.1 `\footnote`

I’d love not to have to tamper with the `\footnote`’s machinery. . . . However, it is too basic a feature not to work out-of-the-box and, unfortunately, it neither uses `\refstepcounter` nor sets `\@currentcounter`. So there’s really not much to do here except trust in the new hook management system.

I have made a feature request though, for having `\@currentcounter` recorded there too: <https://github.com/latex3/latex2e/issues/687>.

CHECK See if the FR has been implemented or not and, if so, remove this.

```
3087 \tl_new:N \l__zrefclever_footnote_type_tl
3088 \tl_set:Nn \l__zrefclever_footnote_type_tl { footnote }
3089 \AddToHook { env / minipage / begin }
3090 { \tl_set:Nn \l__zrefclever_footnote_type_tl { mpfootnote } }
3091 \AddToHook { cmd / @makefntext / before }
3092 {
3093   \__zrefclever_zcsetup:x
3094   { currentcounter = \l__zrefclever_footnote_type_tl }
3095 }
```

9.2 `\appendix`

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

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

```
3096 \AddToHook { cmd / appendix / before }
3097 {
3098   \__zrefclever_zcsetup:n
3099   {
3100     countertype =
3101     {
3102       chapter      = appendix ,
3103       section      = appendix ,
3104       subsection   = appendix ,

```



```

3105         subsubsection = appendix ,
3106     }
3107 }
3108 }

```

Depending on the definition of `\appendix`, using the hook may lead to trouble with the first released version of `ltxcmdhooks` (the one released with the 2021-06-01 kernel). Particularly, if the definition of the command being hooked at contains a double hash mark (`##`) the patch to add the hook, if it needs to be done with the `\scantokens` method, may fail noisily (see <https://tex.stackexchange.com/q/617905>, thanks Phelype Oleinik). The 2021-11-15 kernel release should already handle this gracefully. In the meantime, given we cannot really expect to know what `\appendix` may contain in general, since it potentially gets redefined in quite a number of classes and packages, a user facing workaround may be needed in case of trouble. Phelype Oleinik recommends activating/providing the generic hook in question, so that `ltxcmdhooks` considers the patch as already done, and do the patch ourselves with `etoolbox` (<https://tex.stackexchange.com/a/617998>). Like so:

```

\IfFormatAtLeastTF{2021-11-15}%
  {\ActivateGenericHook}%
  {\ProvideHook}%
  {cmd/appendix/before}
\usepackage{etoolbox}
\pretocmd\appendix
  {\UseHook{cmd/appendix/before}}
  {}{\FAILED}

```

9.3 appendix package

These settings also apply to the memoir class, since it “emulates” the loading of the appendix package.

```

3109 \AddToHook { begindocument }
3110 {
3111     \@ifpackageloaded { appendix }
3112     {
3113         \newcounter { zc@appendix }
3114         \newcounter { zc@save@appendix }
3115         \setcounter { zc@appendix } { 0 }
3116         \setcounter { zc@save@appendix } { 0 }
3117         \cs_if_exist:cTF { chapter }
3118         {
3119             \__zrefclever_zcsetup:n
3120             { counterresetby = { chapter = zc@appendix } }
3121         }
3122         {
3123             \cs_if_exist:cT { section }
3124             {
3125                 \__zrefclever_zcsetup:n
3126                 { counterresetby = { section = zc@appendix } }
3127             }
3128         }
3129     }
\AddToHook { env / appendices / begin }

```

```

3130     {
3131         \stepcounter { zc@save@appendix }
3132         \setcounter { zc@appendix } { \value { zc@save@appendix } }
3133         \__zrefclever_zcsetup:n
3134         {
3135             countertype =
3136             {
3137                 chapter      = appendix ,
3138                 section      = appendix ,
3139                 subsection   = appendix ,
3140                 subsubsection = appendix ,
3141             }
3142         }
3143     }
3144     \AddToHook { env / appendices / end }
3145     { \setcounter { zc@appendix } { 0 } }
3146     \AddToHook { cmd / appendix / before }
3147     {
3148         \stepcounter { zc@save@appendix }
3149         \setcounter { zc@appendix } { \value { zc@save@appendix } }
3150     }
3151     \AddToHook { env / subappendices / begin }
3152     {
3153         \__zrefclever_zcsetup:n
3154         {
3155             countertype =
3156             {
3157                 section      = appendix ,
3158                 subsection   = appendix ,
3159                 subsubsection = appendix ,
3160             } ,
3161         }
3162     }
3163     \msg_info:nnn { zref-clever } { compat-package } { appendix }
3164 }
3165 {}
3166 }

```

9.4 amsmath package

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

```

3167 \AddToHook { begindocument }
3168 {
3169     \@ifpackageloaded { amsmath }
3170     {

```

First, we define a function for label setting inside `amsmath` math environments, we want it to set both `\zlabel` and `\label`. We may “get a ride” but not steal the place altogether. This makes for potentially redundant labels, but seems a good compromise. We *must* use the lower level `\zref@label` in this context, and hence also handle protection with `\zref@wrapper@babel`, because `\zlabel` makes itself no-op when `\label` is equal to `\ltx@gobble`, and that’s precisely the case inside the `multline` environment (and, damn!, I took a beating of this detail...).

```

3171 \cs_set_nopar:Npn \__zrefclever_ltxlabel:n #1
3172 {
3173   \__zrefclever_orig_ltxlabel:n {#1}
3174   \zref@wrapper@babel \zref@label {#1}
3175 }

```

Then we must store the original value of `\ltx@label`, which is the macro actually responsible for setting the labels inside `amsmath`'s math environments. And, after that, redefine it to be `__zrefclever_ltxlabel:n` instead. We must handle `hyperref` here, which comes very late in the preamble, and which loads `nameref` at `begindocument`, which in turn, lets `\ltx@label` be `\label`. This has to come after `nameref`. `cleveref` also redefines it, and comes even later, but this procedure is not compatible with it. Technically, some care is needed here, probably mostly on the documentation side. If `cleveref` comes last and hence its redefinition takes precedence, this is of little consequence to `zref-clever` except that we won't be able to refer to the labels in `amsmath`'s environments with `\zcref`. However, if `cleveref`'s definition is overwritten by `zref-clever`, this may be a substantial problem for `cleveref`, since it will find the label, but it won't contain the data it is expecting. Therefore, if for some reason `cleveref` is being used alongside `cleveref`, it is due to follow the latter's documented recommendation to load it last. And use `\cref` to make references to those. CHECK Should I just make this no-op in case 'cleveref' is loaded?

```

3176 \IfFormatAtLeastTF { 2021-11-15 }
3177 {
3178   \@ifpackageloaded { hyperref }
3179   {
3180     \AddToHook { package / nameref / after }
3181     {
3182       \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3183       \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3184     }
3185   }
3186   {
3187     \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3188     \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3189   }
3190 }
3191 {
3192   \@ifpackageloaded { hyperref }
3193   {
3194     \@ifpackageloaded { nameref }
3195     {
3196       \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3197       \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3198     }
3199     {
3200       \AddToHook { package / after / nameref }
3201       {
3202         \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3203         \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3204       }
3205     }
3206   }
3207 }

```

```

3208         \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3209         \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3210     }
3211 }

```

The `subequations` environment uses `parentequation` and `equation` as counters, but only the later is subject to `\refstepcounter`. What happens is: at the start, `equation` is refstepped, it is then stored in `parentequation` and set to ‘0’ and, at the end of the environment it is restored to the value of `parentequation`. So, here, we really must specify manually `currentcounter` and the resetting. Note that, for `subequations`, `\zlabel` works just fine (that is, if given immediately after `\begin{subequations}`, to refer to the parent equation).

```

3212 \AddToHook { env / subequations / begin }
3213 {
3214     \__zrefclever_zcsetup:x
3215     {
3216         counterresetby =
3217         {
3218             parentequation =
3219             \__zrefclever_counter_reset_by:n { equation } ,
3220             equation = parentequation ,
3221         } ,
3222         currentcounter = parentequation ,
3223         countertype = { parentequation = equation } ,
3224     }
3225 }

```

`amsmath` does use `\refstepcounter` for the `equation` counter throughout. But we still have to set `currentcounter` manually for two reasons. First: `\tag`, which naturally does not change the counter, and just sets `\@currentlabel`. Thus a label to a tag gets `\@currentcounter` from whatever came last, normally the current sectioning command. And we also include the starred environments here, so that we can get proper data for `\tagged` equations even if the environment is unnumbered. Second, since we had to manually set `currentcounter` to `parentequation` in `subequations`, we also have to manually set it to `equation` in environments which may be used within it. The `xxalignat` environment is not included, because it is “starred” by default (i.e. unnumbered), and does not display or accepts labels or tags anyway. The `-ed` (`gathered`, `aligned`, and `alignedat`) and `cases` environments “must appear within an enclosing math environment”. Same logic applies to other environments defined or redefined by the package, like `array`, `matrix` and variations. Finally, `split` too can only be used as part of another environment.

```

3226 \clist_map_inline:nn
3227 {
3228     equation ,
3229     equation* ,
3230     align ,
3231     align* ,
3232     alignat ,
3233     alignat* ,
3234     flalign ,
3235     flalign* ,
3236     xalignat ,
3237     xalignat* ,

```

```

3238         gather ,
3239         gather* ,
3240         multiline ,
3241         multiline* ,
3242     }
3243     {
3244         \AddToHook { env / #1 / begin }
3245         { \_zrefclever_zcsetup:n { currentcounter = equation } }
3246     }

```

And a last touch of care for amsmath’s refinements: make the equation references `\textup`.

```

3247     \zcRefTypeSetup { equation } { reffont = \upshape }
3248     \msg_info:nnn { zref-clever } { compat-package } { amsmath }
3249 }
3250 {}
3251 }

```

9.5 mathtools package

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

Hence, as far as I can tell, the only cross-reference related feature to deal with is the `showonlyrefs` option, whose machinery involves writing an extra internal label to the `.aux` file to track for labels which get actually referred to. This is a little more involved, and implies in doing special handling inside `\zcref`, but the feature is very cool, so it’s worth it.

```

3252 \bool_new:N \l__zrefclever_mathtools_showonlyrefs_bool
3253 \AddToHook { begindocument }
3254 {
3255     \@ifpackageloaded { mathtools }
3256     {
3257         \MH_if_boolean:nT { show_only_refs }
3258         {
3259             \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
3260             \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
3261             {
3262                 \@bsphack
3263                 \seq_map_inline:Nn #1
3264                 {
3265                     \exp_args:Nx \tl_if_eq:nnTF
3266                     { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
3267                     { equation }
3268                     {
3269                         \protected@write \@auxout { }
3270                         { \string \MT@newlabel {##1} }
3271                     }
3272                 }
3273                 \exp_args:Nx \tl_if_eq:nnT
3274                 { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
3275                 { parentequation }

```

```

3276             {
3277                 \protected@write \@auxout { }
3278                 { \string \MT@newlabel {##1} }
3279             }
3280         }
3281     }
3282     \@esphack
3283 }
3284 \msg_info:nnn { zref-clever } { compat-package } { mathtools }
3285 }
3286 }
3287 {}
3288 }

```

9.6 breqn package

From the `breqn` documentation: “Use of the normal `\label` command instead of the `label` option works, I think, most of the time (untested)”. Indeed, light testing suggest it does work for `\zlabel` just as well. However, if it happens not to work, there was no easy alternative handle I could find. In particular, it does not seem viable to leverage the `label=` option without hacking the package internals, even if the case of doing so would not be specially tricky, just “not very civil”.

```

3289 \AddToHook { begindocument }
3290 {
3291     \@ifpackageloaded { breqn }
3292     {

```

Contrary to the practice in `amsmath`, which prints `\tag` even in unnumbered environments, the starred environments from `breqn` don’t typeset any tag/number at all, even for a manually given `number=` as an option. So, even if one can actually set a label in them, it is not really meaningful to make a reference to them.

```

3293     \AddToHook { env / dgroup / begin }
3294     {
3295         \__zrefclever_zcsetup:x
3296         {
3297             counterresetby =
3298             {
3299                 parentequation =
3300                 \__zrefclever_counter_reset_by:n { equation } ,
3301                 equation = parentequation ,
3302             } ,
3303             currentcounter = parentequation ,
3304             countertype = { parentequation = equation } ,
3305         }
3306     }
3307     \clist_map_inline:nn
3308     {
3309         dmath ,
3310         dseries ,
3311         darray ,
3312     }
3313     {
3314         \AddToHook { env / #1 / begin }

```

```

3315         { \_zrefclever_zcsetup:n { currentcounter = equation } }
3316     }
3317 }
3318 {}
3319 }

```

9.7 listings package

```

3320 \AddToHook { begindocument }
3321 {
3322     \@ifpackageloaded { listings }
3323     {
3324         \_zrefclever_zcsetup:n
3325         {
3326             countertype =
3327             {
3328                 lstlisting = listing ,
3329                 lstnumber = line ,
3330             } ,
3331             counterresetby = { lstnumber = lstlisting } ,
3332         }
3333         \lst@AddToHook { Init }
3334         {

```

Set (also) a `\zlabel` with the label received in the `label=` option from the `lstlisting` environment.

```

3335         \tl_if_empty:NF \lst@label
3336         { \zlabel { \lst@label } }

```

The correct place to set `currentcounter` to `lstnumber` is indeed the `Init` hook, since `listings` itself sets `\@currentlabel` to `\thelstnumber` in the same hook. See section “Line numbers” of ‘`texdoc listings-devel`’ (the `.dtx`), and search for the definition of macro `\c@lstnumber`. Note that `listings` *does use* `\refstepcounter{lstnumber}`, but does so in the `EveryPar` hook, and there must be some grouping involved such that `\@currentcounter` ends up not being visible to the label. Indeed, the fact that `listings` manually sets `\@currentlabel` to `\thelstnumber` is a signal that the work of `\refstepcounter` is being restrained somehow.

```

3337         \_zrefclever_zcsetup:n { currentcounter = lstnumber }
3338     }
3339     \msg_info:nnn { zref-clever } { compat-package } { listings }
3340 }
3341 {}
3342 }

```

9.8 enumitem package

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

The only real reason to “renew” `enumerate` itself is to change `{\max-depth}`. `\renewlist` *hard-codes* `max-depth` in the environment’s definition (well, just as the kernel does), so we cannot retrieve this information from any sort of variable. But `\renewlist`

also creates any needed missing counters, so we can use their existence to make the appropriate settings. In the end, the existence of the counters is indeed what matters from `zref-clever`'s perspective. Since the first four are defined by the kernel and already setup for `zref-clever` by default, we start from 5, and stop at the first non-existent `\c@enumN` counter.

```

3343 \AddToHook { begindocument }
3344 {
3345   \@ifpackageloaded { enumitem }
3346   {
3347     \int_set:Nn \l_tmpa_int { 5 }
3348     \bool_while_do:nn
3349     {
3350       \cs_if_exist_p:c
3351       { c@ enum \int_to_roman:n { \l_tmpa_int } }
3352     }
3353     {
3354       \__zrefclever_zcsetup:x
3355       {
3356         counterresetby =
3357         {
3358           enum \int_to_roman:n { \l_tmpa_int } =
3359           enum \int_to_roman:n { \l_tmpa_int - 1 }
3360         } ,
3361         countertype =
3362         { enum \int_to_roman:n { \l_tmpa_int } = item } ,
3363       }
3364       \int_incr:N \l_tmpa_int
3365     }
3366     \int_compare:nNnT { \l_tmpa_int } > { 5 }
3367     { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
3368   }
3369   {}
3370 }
3371 \</package>

```

10 Dictionaries

10.1 English

```

3372 \<*package>
3373 \zcDeclareLanguage { english }
3374 \zcDeclareLanguageAlias { american } { english }
3375 \zcDeclareLanguageAlias { australian } { english }
3376 \zcDeclareLanguageAlias { british } { english }
3377 \zcDeclareLanguageAlias { canadian } { english }
3378 \zcDeclareLanguageAlias { newzealand } { english }
3379 \zcDeclareLanguageAlias { UKenglish } { english }
3380 \zcDeclareLanguageAlias { USenglish } { english }
3381 \</package>
3382 \<*dict-english>
3383 namesep = {\nobreakspace} ,

```



```

3384 pairsep = {\~and\nobreakspace} ,
3385 listsep = {,\~} ,
3386 lastsep = {\~and\nobreakspace} ,
3387 tpairsep = {\~and\nobreakspace} ,
3388 tlistsep = {,\~} ,
3389 tlastsep = {,\~and\nobreakspace} ,
3390 notesep = {\~} ,
3391 rangeseq = {\~to\nobreakspace} ,
3392
3393 type = part ,
3394     Name-sg = Part ,
3395     name-sg = part ,
3396     Name-pl = Parts ,
3397     name-pl = parts ,
3398
3399 type = chapter ,
3400     Name-sg = Chapter ,
3401     name-sg = chapter ,
3402     Name-pl = Chapters ,
3403     name-pl = chapters ,
3404
3405 type = section ,
3406     Name-sg = Section ,
3407     name-sg = section ,
3408     Name-pl = Sections ,
3409     name-pl = sections ,
3410
3411 type = paragraph ,
3412     Name-sg = Paragraph ,
3413     name-sg = paragraph ,
3414     Name-pl = Paragraphs ,
3415     name-pl = paragraphs ,
3416     Name-sg-ab = Par. ,
3417     name-sg-ab = par. ,
3418     Name-pl-ab = Par. ,
3419     name-pl-ab = par. ,
3420
3421 type = appendix ,
3422     Name-sg = Appendix ,
3423     name-sg = appendix ,
3424     Name-pl = Appendices ,
3425     name-pl = appendices ,
3426
3427 type = subappendix ,
3428     Name-sg = Appendix ,
3429     name-sg = appendix ,
3430     Name-pl = Appendices ,
3431     name-pl = appendices ,
3432
3433 type = page ,
3434     Name-sg = Page ,
3435     name-sg = page ,
3436     Name-pl = Pages ,
3437     name-pl = pages ,

```

```

3438     name-sg-ab = p. ,
3439     name-pl-ab = pp. ,
3440     rangesep = {\textendash} ,
3441
3442     type = line ,
3443     Name-sg = Line ,
3444     name-sg = line ,
3445     Name-pl = Lines ,
3446     name-pl = lines ,
3447
3448     type = figure ,
3449     Name-sg = Figure ,
3450     name-sg = figure ,
3451     Name-pl = Figures ,
3452     name-pl = figures ,
3453     Name-sg-ab = Fig. ,
3454     name-sg-ab = fig. ,
3455     Name-pl-ab = Figs. ,
3456     name-pl-ab = figs. ,
3457
3458     type = table ,
3459     Name-sg = Table ,
3460     name-sg = table ,
3461     Name-pl = Tables ,
3462     name-pl = tables ,
3463
3464     type = item ,
3465     Name-sg = Item ,
3466     name-sg = item ,
3467     Name-pl = Items ,
3468     name-pl = items ,
3469
3470     type = footnote ,
3471     Name-sg = Footnote ,
3472     name-sg = footnote ,
3473     Name-pl = Footnotes ,
3474     name-pl = footnotes ,
3475
3476     type = note ,
3477     Name-sg = Note ,
3478     name-sg = note ,
3479     Name-pl = Notes ,
3480     name-pl = notes ,
3481
3482     type = equation ,
3483     Name-sg = Equation ,
3484     name-sg = equation ,
3485     Name-pl = Equations ,
3486     name-pl = equations ,
3487     Name-sg-ab = Eq. ,
3488     name-sg-ab = eq. ,
3489     Name-pl-ab = Eqs. ,
3490     name-pl-ab = eqs. ,
3491     refpre-in = {() ,

```

```

3492     refpos-in = {} } ,
3493
3494 type = theorem ,
3495     Name-sg = Theorem ,
3496     name-sg = theorem ,
3497     Name-pl = Theorems ,
3498     name-pl = theorems ,
3499
3500 type = lemma ,
3501     Name-sg = Lemma ,
3502     name-sg = lemma ,
3503     Name-pl = Lemmas ,
3504     name-pl = lemmas ,
3505
3506 type = corollary ,
3507     Name-sg = Corollary ,
3508     name-sg = corollary ,
3509     Name-pl = Corollaries ,
3510     name-pl = corollaries ,
3511
3512 type = proposition ,
3513     Name-sg = Proposition ,
3514     name-sg = proposition ,
3515     Name-pl = Propositions ,
3516     name-pl = propositions ,
3517
3518 type = definition ,
3519     Name-sg = Definition ,
3520     name-sg = definition ,
3521     Name-pl = Definitions ,
3522     name-pl = definitions ,
3523
3524 type = proof ,
3525     Name-sg = Proof ,
3526     name-sg = proof ,
3527     Name-pl = Proofs ,
3528     name-pl = proofs ,
3529
3530 type = result ,
3531     Name-sg = Result ,
3532     name-sg = result ,
3533     Name-pl = Results ,
3534     name-pl = results ,
3535
3536 type = remark ,
3537     Name-sg = Remark ,
3538     name-sg = remark ,
3539     Name-pl = Remarks ,
3540     name-pl = remarks ,
3541
3542 type = example ,
3543     Name-sg = Example ,
3544     name-sg = example ,
3545     Name-pl = Examples ,

```

```

3546   name-pl = examples ,
3547
3548   type = algorithm ,
3549     Name-sg = Algorithm ,
3550     name-sg = algorithm ,
3551     Name-pl = Algorithms ,
3552     name-pl = algorithms ,
3553
3554   type = listing ,
3555     Name-sg = Listing ,
3556     name-sg = listing ,
3557     Name-pl = Listings ,
3558     name-pl = listings ,
3559
3560   type = exercise ,
3561     Name-sg = Exercise ,
3562     name-sg = exercise ,
3563     Name-pl = Exercises ,
3564     name-pl = exercises ,
3565
3566   type = solution ,
3567     Name-sg = Solution ,
3568     name-sg = solution ,
3569     Name-pl = Solutions ,
3570     name-pl = solutions ,
3571 </dict-english>

```

10.2 German

```

3572 <*package>
3573 \zcDeclareLanguage
3574   [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
3575   { german }
3576 \zcDeclareLanguageAlias { austrian    } { german }
3577 \zcDeclareLanguageAlias { germanb     } { german }
3578 \zcDeclareLanguageAlias { ngerman     } { german }
3579 \zcDeclareLanguageAlias { naustrian   } { german }
3580 \zcDeclareLanguageAlias { nswissgerman } { german }
3581 \zcDeclareLanguageAlias { swissgerman } { german }
3582 </package>
3583 <*dict-german>
3584 namesep = {\nobreakspace} ,
3585 pairsep  = {\und\nobreakspace} ,
3586 listsep  = { , ~ } ,
3587 lastsep  = {\und\nobreakspace} ,
3588 tpairsep = {\und\nobreakspace} ,
3589 tlistsep = { , ~ } ,
3590 tlastsep = {\und\nobreakspace} ,
3591 notesep  = { ~ } ,
3592 rangesep = {\bis\nobreakspace} ,
3593
3594 type = part ,
3595 gender = m ,
3596 case = N ,

```

```

3597     Name-sg = Teil ,
3598     Name-pl = Teile ,
3599     case = A ,
3600     Name-sg = Teil ,
3601     Name-pl = Teile ,
3602     case = D ,
3603     Name-sg = Teil ,
3604     Name-pl = Teilen ,
3605     case = G ,
3606     Name-sg = Teiles ,
3607     Name-pl = Teile ,
3608
3609 type = chapter ,
3610     gender = n ,
3611     case = N ,
3612     Name-sg = Kapitel ,
3613     Name-pl = Kapitel ,
3614     case = A ,
3615     Name-sg = Kapitel ,
3616     Name-pl = Kapitel ,
3617     case = D ,
3618     Name-sg = Kapitel ,
3619     Name-pl = Kapiteln ,
3620     case = G ,
3621     Name-sg = Kapiteln ,
3622     Name-pl = Kapitel ,
3623
3624 type = section ,
3625     gender = m ,
3626     case = N ,
3627     Name-sg = Abschnitt ,
3628     Name-pl = Abschnitte ,
3629     case = A ,
3630     Name-sg = Abschnitt ,
3631     Name-pl = Abschnitte ,
3632     case = D ,
3633     Name-sg = Abschnitt ,
3634     Name-pl = Abschnitten ,
3635     case = G ,
3636     Name-sg = Abschnitts ,
3637     Name-pl = Abschnitte ,
3638
3639 type = paragraph ,
3640     gender = m ,
3641     case = N ,
3642     Name-sg = Absatz ,
3643     Name-pl = Absätze ,
3644     case = A ,
3645     Name-sg = Absatz ,
3646     Name-pl = Absätze ,
3647     case = D ,
3648     Name-sg = Absatz ,
3649     Name-pl = Absätzen ,
3650     case = G ,

```

```

3651     Name-sg = Absatzes ,
3652     Name-pl = Absätze ,
3653
3654 type = appendix ,
3655     gender = m ,
3656     case = N ,
3657     Name-sg = Anhang ,
3658     Name-pl = Anhänge ,
3659     case = A ,
3660     Name-sg = Anhang ,
3661     Name-pl = Anhänge ,
3662     case = D ,
3663     Name-sg = Anhang ,
3664     Name-pl = Anhängen ,
3665     case = G ,
3666     Name-sg = Anhangs ,
3667     Name-pl = Anhänge ,
3668
3669 type = subappendix ,
3670     gender = m ,
3671     case = N ,
3672     Name-sg = Anhang ,
3673     Name-pl = Anhänge ,
3674     case = A ,
3675     Name-sg = Anhang ,
3676     Name-pl = Anhänge ,
3677     case = D ,
3678     Name-sg = Anhang ,
3679     Name-pl = Anhängen ,
3680     case = G ,
3681     Name-sg = Anhangs ,
3682     Name-pl = Anhänge ,
3683
3684 type = page ,
3685     gender = f ,
3686     case = N ,
3687     Name-sg = Seite ,
3688     Name-pl = Seiten ,
3689     case = A ,
3690     Name-sg = Seite ,
3691     Name-pl = Seiten ,
3692     case = D ,
3693     Name-sg = Seite ,
3694     Name-pl = Seiten ,
3695     case = G ,
3696     Name-sg = Seite ,
3697     Name-pl = Seiten ,
3698     rangesep = {\textendash} ,
3699
3700 type = line ,
3701     gender = f ,
3702     case = N ,
3703     Name-sg = Zeile ,
3704     Name-pl = Zeilen ,

```

```

3705 case = A ,
3706     Name-sg = Zeile ,
3707     Name-pl = Zeilen ,
3708 case = D ,
3709     Name-sg = Zeile ,
3710     Name-pl = Zeilen ,
3711 case = G ,
3712     Name-sg = Zeile ,
3713     Name-pl = Zeilen ,
3714
3715 type = figure ,
3716     gender = f ,
3717     case = N ,
3718         Name-sg = Abbildung ,
3719         Name-pl = Abbildungen ,
3720         Name-sg-ab = Abb. ,
3721         Name-pl-ab = Abb. ,
3722     case = A ,
3723         Name-sg = Abbildung ,
3724         Name-pl = Abbildungen ,
3725         Name-sg-ab = Abb. ,
3726         Name-pl-ab = Abb. ,
3727     case = D ,
3728         Name-sg = Abbildung ,
3729         Name-pl = Abbildungen ,
3730         Name-sg-ab = Abb. ,
3731         Name-pl-ab = Abb. ,
3732     case = G ,
3733         Name-sg = Abbildung ,
3734         Name-pl = Abbildungen ,
3735         Name-sg-ab = Abb. ,
3736         Name-pl-ab = Abb. ,
3737
3738 type = table ,
3739     gender = f ,
3740     case = N ,
3741         Name-sg = Tabelle ,
3742         Name-pl = Tabellen ,
3743     case = A ,
3744         Name-sg = Tabelle ,
3745         Name-pl = Tabellen ,
3746     case = D ,
3747         Name-sg = Tabelle ,
3748         Name-pl = Tabellen ,
3749     case = G ,
3750         Name-sg = Tabelle ,
3751         Name-pl = Tabellen ,
3752
3753 type = item ,
3754     gender = m ,
3755     case = N ,
3756         Name-sg = Punkt ,
3757         Name-pl = Punkte ,
3758     case = A ,

```

```

3759     Name-sg = Punkt ,
3760     Name-pl = Punkte ,
3761     case = D ,
3762     Name-sg = Punkt ,
3763     Name-pl = Punkten ,
3764     case = G ,
3765     Name-sg = Punktes ,
3766     Name-pl = Punkte ,
3767
3768 type = footnote ,
3769     gender = f ,
3770     case = N ,
3771     Name-sg = Fußnote ,
3772     Name-pl = Fußnoten ,
3773     case = A ,
3774     Name-sg = Fußnote ,
3775     Name-pl = Fußnoten ,
3776     case = D ,
3777     Name-sg = Fußnote ,
3778     Name-pl = Fußnoten ,
3779     case = G ,
3780     Name-sg = Fußnote ,
3781     Name-pl = Fußnoten ,
3782
3783 type = note ,
3784     gender = f ,
3785     case = N ,
3786     Name-sg = Anmerkung ,
3787     Name-pl = Anmerkungen ,
3788     case = A ,
3789     Name-sg = Anmerkung ,
3790     Name-pl = Anmerkungen ,
3791     case = D ,
3792     Name-sg = Anmerkung ,
3793     Name-pl = Anmerkungen ,
3794     case = G ,
3795     Name-sg = Anmerkung ,
3796     Name-pl = Anmerkungen ,
3797
3798 type = equation ,
3799     gender = f ,
3800     case = N ,
3801     Name-sg = Gleichung ,
3802     Name-pl = Gleichungen ,
3803     case = A ,
3804     Name-sg = Gleichung ,
3805     Name-pl = Gleichungen ,
3806     case = D ,
3807     Name-sg = Gleichung ,
3808     Name-pl = Gleichungen ,
3809     case = G ,
3810     Name-sg = Gleichung ,
3811     Name-pl = Gleichungen ,
3812     refpre-in = {() ,

```



```

3813   refpos-in = {} } ,
3814
3815   type = theorem ,
3816   gender = n ,
3817   case = N ,
3818       Name-sg = Theorem ,
3819       Name-pl = Theoreme ,
3820   case = A ,
3821       Name-sg = Theorem ,
3822       Name-pl = Theoreme ,
3823   case = D ,
3824       Name-sg = Theorem ,
3825       Name-pl = Theoremen ,
3826   case = G ,
3827       Name-sg = Theorems ,
3828       Name-pl = Theoreme ,
3829
3830   type = lemma ,
3831   gender = n ,
3832   case = N ,
3833       Name-sg = Lemma ,
3834       Name-pl = Lemmata ,
3835   case = A ,
3836       Name-sg = Lemma ,
3837       Name-pl = Lemmata ,
3838   case = D ,
3839       Name-sg = Lemma ,
3840       Name-pl = Lemmata ,
3841   case = G ,
3842       Name-sg = Lemmas ,
3843       Name-pl = Lemmata ,
3844
3845   type = corollary ,
3846   gender = n ,
3847   case = N ,
3848       Name-sg = Korollar ,
3849       Name-pl = Korollare ,
3850   case = A ,
3851       Name-sg = Korollar ,
3852       Name-pl = Korollare ,
3853   case = D ,
3854       Name-sg = Korollar ,
3855       Name-pl = Korollaren ,
3856   case = G ,
3857       Name-sg = Korollars ,
3858       Name-pl = Korollare ,
3859
3860   type = proposition ,
3861   gender = m ,
3862   case = N ,
3863       Name-sg = Satz ,
3864       Name-pl = Sätze ,
3865   case = A ,
3866       Name-sg = Satz ,

```

```

3867     Name-pl = Sätze ,
3868     case = D ,
3869     Name-sg = Satz ,
3870     Name-pl = Sätzen ,
3871     case = G ,
3872     Name-sg = Satzes ,
3873     Name-pl = Sätze ,
3874
3875 type = definition ,
3876     gender = f ,
3877     case = N ,
3878     Name-sg = Definition ,
3879     Name-pl = Definitionen ,
3880     case = A ,
3881     Name-sg = Definition ,
3882     Name-pl = Definitionen ,
3883     case = D ,
3884     Name-sg = Definition ,
3885     Name-pl = Definitionen ,
3886     case = G ,
3887     Name-sg = Definition ,
3888     Name-pl = Definitionen ,
3889
3890 type = proof ,
3891     gender = m ,
3892     case = N ,
3893     Name-sg = Beweis ,
3894     Name-pl = Beweise ,
3895     case = A ,
3896     Name-sg = Beweis ,
3897     Name-pl = Beweise ,
3898     case = D ,
3899     Name-sg = Beweis ,
3900     Name-pl = Beweisen ,
3901     case = G ,
3902     Name-sg = Beweises ,
3903     Name-pl = Beweise ,
3904
3905 type = result ,
3906     gender = n ,
3907     case = N ,
3908     Name-sg = Ergebnis ,
3909     Name-pl = Ergebnisse ,
3910     case = A ,
3911     Name-sg = Ergebnis ,
3912     Name-pl = Ergebnisse ,
3913     case = D ,
3914     Name-sg = Ergebnis ,
3915     Name-pl = Ergebnissen ,
3916     case = G ,
3917     Name-sg = Ergebnisses ,
3918     Name-pl = Ergebnisse ,
3919
3920 type = remark ,

```

```

3921 gender = f ,
3922 case = N ,
3923     Name-sg = Bemerkung ,
3924     Name-pl = Bemerkungen ,
3925 case = A ,
3926     Name-sg = Bemerkung ,
3927     Name-pl = Bemerkungen ,
3928 case = D ,
3929     Name-sg = Bemerkung ,
3930     Name-pl = Bemerkungen ,
3931 case = G ,
3932     Name-sg = Bemerkung ,
3933     Name-pl = Bemerkungen ,
3934
3935 type = example ,
3936 gender = n ,
3937 case = N ,
3938     Name-sg = Beispiel ,
3939     Name-pl = Beispiele ,
3940 case = A ,
3941     Name-sg = Beispiel ,
3942     Name-pl = Beispiele ,
3943 case = D ,
3944     Name-sg = Beispiel ,
3945     Name-pl = Beispielen ,
3946 case = G ,
3947     Name-sg = Beispiels ,
3948     Name-pl = Beispiele ,
3949
3950 type = algorithm ,
3951 gender = m ,
3952 case = N ,
3953     Name-sg = Algorithmus ,
3954     Name-pl = Algorithmen ,
3955 case = A ,
3956     Name-sg = Algorithmus ,
3957     Name-pl = Algorithmen ,
3958 case = D ,
3959     Name-sg = Algorithmus ,
3960     Name-pl = Algorithmen ,
3961 case = G ,
3962     Name-sg = Algorithmus ,
3963     Name-pl = Algorithmen ,
3964
3965 type = listing ,
3966 gender = n ,
3967 case = N ,
3968     Name-sg = Listing ,
3969     Name-pl = Listings ,
3970 case = A ,
3971     Name-sg = Listing ,
3972     Name-pl = Listings ,
3973 case = D ,
3974     Name-sg = Listing ,

```

```

3975     Name-pl = Listings ,
3976     case = G ,
3977     Name-sg = Listings ,
3978     Name-pl = Listings ,
3979
3980 type = exercise ,
3981     gender = f ,
3982     case = N ,
3983     Name-sg = Übungsaufgabe ,
3984     Name-pl = Übungsaufgaben ,
3985     case = A ,
3986     Name-sg = Übungsaufgabe ,
3987     Name-pl = Übungsaufgaben ,
3988     case = D ,
3989     Name-sg = Übungsaufgabe ,
3990     Name-pl = Übungsaufgaben ,
3991     case = G ,
3992     Name-sg = Übungsaufgabe ,
3993     Name-pl = Übungsaufgaben ,
3994
3995 type = solution ,
3996     gender = f ,
3997     case = N ,
3998     Name-sg = Lösung ,
3999     Name-pl = Lösungen ,
4000     case = A ,
4001     Name-sg = Lösung ,
4002     Name-pl = Lösungen ,
4003     case = D ,
4004     Name-sg = Lösung ,
4005     Name-pl = Lösungen ,
4006     case = G ,
4007     Name-sg = Lösung ,
4008     Name-pl = Lösungen ,
4009 </dict-german>

```

10.3 French

```

4010 <*package>
4011 \zcDeclareLanguage [ gender = { f , m } ] { french }
4012 \zcDeclareLanguageAlias { acadian } { french }
4013 \zcDeclareLanguageAlias { canadien } { french }
4014 \zcDeclareLanguageAlias { francais } { french }
4015 \zcDeclareLanguageAlias { frenchb } { french }
4016 </package>
4017 <*dict-french>
4018 namesep = {\nobreakspace} ,
4019 pairsep = {\~et\nobreakspace} ,
4020 listsep = { ,~ } ,
4021 lastsep = {\~et\nobreakspace} ,
4022 tpairsep = {\~et\nobreakspace} ,
4023 tlistsep = { ,~ } ,
4024 tlastsep = {\~et\nobreakspace} ,
4025 notesep = { ~ } ,

```

```

4026 rangesep = {\~\nobreakspace} ,
4027
4028 type = part ,
4029     gender = f ,
4030     Name-sg = Partie ,
4031     name-sg = partie ,
4032     Name-pl = Parties ,
4033     name-pl = parties ,
4034
4035 type = chapter ,
4036     gender = m ,
4037     Name-sg = Chapitre ,
4038     name-sg = chapitre ,
4039     Name-pl = Chapitres ,
4040     name-pl = chapitres ,
4041
4042 type = section ,
4043     gender = f ,
4044     Name-sg = Section ,
4045     name-sg = section ,
4046     Name-pl = Sections ,
4047     name-pl = sections ,
4048
4049 type = paragraph ,
4050     gender = m ,
4051     Name-sg = Paragraphe ,
4052     name-sg = paragraphe ,
4053     Name-pl = Paragraphes ,
4054     name-pl = paragraphes ,
4055
4056 type = appendix ,
4057     gender = f ,
4058     Name-sg = Annexe ,
4059     name-sg = annexe ,
4060     Name-pl = Annexes ,
4061     name-pl = annexes ,
4062
4063 type = subappendix ,
4064     gender = f ,
4065     Name-sg = Annexe ,
4066     name-sg = annexe ,
4067     Name-pl = Annexes ,
4068     name-pl = annexes ,
4069
4070 type = page ,
4071     gender = f ,
4072     Name-sg = Page ,
4073     name-sg = page ,
4074     Name-pl = Pages ,
4075     name-pl = pages ,
4076     rangesep = {\textendash} ,
4077
4078 type = line ,
4079     gender = f ,

```

```

4080 Name-sg = Ligne ,
4081 name-sg = ligne ,
4082 Name-pl = Lignes ,
4083 name-pl = lignes ,
4084
4085 type = figure ,
4086 gender = f ,
4087 Name-sg = Figure ,
4088 name-sg = figure ,
4089 Name-pl = Figures ,
4090 name-pl = figures ,
4091
4092 type = table ,
4093 gender = f ,
4094 Name-sg = Table ,
4095 name-sg = table ,
4096 Name-pl = Tables ,
4097 name-pl = tables ,
4098
4099 type = item ,
4100 gender = m ,
4101 Name-sg = Point ,
4102 name-sg = point ,
4103 Name-pl = Points ,
4104 name-pl = points ,
4105
4106 type = footnote ,
4107 gender = f ,
4108 Name-sg = Note ,
4109 name-sg = note ,
4110 Name-pl = Notes ,
4111 name-pl = notes ,
4112
4113 type = note ,
4114 gender = f ,
4115 Name-sg = Note ,
4116 name-sg = note ,
4117 Name-pl = Notes ,
4118 name-pl = notes ,
4119
4120 type = equation ,
4121 gender = f ,
4122 Name-sg = Équation ,
4123 name-sg = équation ,
4124 Name-pl = Équations ,
4125 name-pl = équations ,
4126 refpre-in = {()} ,
4127 refpos-in = {} ,
4128
4129 type = theorem ,
4130 gender = m ,
4131 Name-sg = Théorème ,
4132 name-sg = théorème ,
4133 Name-pl = Théorèmes ,

```

```

4134     name-pl = théorèmes ,
4135
4136 type = lemma ,
4137     gender = m ,
4138     Name-sg = Lemme ,
4139     name-sg = lemme ,
4140     Name-pl = Lemmes ,
4141     name-pl = lemmes ,
4142
4143 type = corollary ,
4144     gender = m ,
4145     Name-sg = Corollaire ,
4146     name-sg = corollaire ,
4147     Name-pl = Corollaires ,
4148     name-pl = corollaires ,
4149
4150 type = proposition ,
4151     gender = f ,
4152     Name-sg = Proposition ,
4153     name-sg = proposition ,
4154     Name-pl = Propositions ,
4155     name-pl = propositions ,
4156
4157 type = definition ,
4158     gender = f ,
4159     Name-sg = Définition ,
4160     name-sg = définition ,
4161     Name-pl = Définitions ,
4162     name-pl = définitions ,
4163
4164 type = proof ,
4165     gender = f ,
4166     Name-sg = Démonstration ,
4167     name-sg = démonstration ,
4168     Name-pl = Démonstrations ,
4169     name-pl = démonstrations ,
4170
4171 type = result ,
4172     gender = m ,
4173     Name-sg = Résultat ,
4174     name-sg = résultat ,
4175     Name-pl = Résultats ,
4176     name-pl = résultats ,
4177
4178 type = remark ,
4179     gender = f ,
4180     Name-sg = Remarque ,
4181     name-sg = remarque ,
4182     Name-pl = Remarques ,
4183     name-pl = remarques ,
4184
4185 type = example ,
4186     gender = m ,
4187     Name-sg = Exemple ,

```

```

4188   name-sg = exemple ,
4189   Name-pl = Exemples ,
4190   name-pl = exemples ,
4191
4192   type = algorithm ,
4193   gender = m ,
4194   Name-sg = Algorithme ,
4195   name-sg = algorithme ,
4196   Name-pl = Algorithmes ,
4197   name-pl = algorithmes ,
4198
4199   type = listing ,
4200   gender = f ,
4201   Name-sg = Liste ,
4202   name-sg = liste ,
4203   Name-pl = Listes ,
4204   name-pl = listes ,
4205
4206   type = exercise ,
4207   gender = m ,
4208   Name-sg = Exercice ,
4209   name-sg = exercice ,
4210   Name-pl = Exercices ,
4211   name-pl = exercices ,
4212
4213   type = solution ,
4214   gender = f ,
4215   Name-sg = Solution ,
4216   name-sg = solution ,
4217   Name-pl = Solutions ,
4218   name-pl = solutions ,
4219 </dict-french>

```

10.4 Portuguese

```

4220 <*package>
4221 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
4222 \zcDeclareLanguageAlias { brazilian } { portuguese }
4223 \zcDeclareLanguageAlias { brazil } { portuguese }
4224 \zcDeclareLanguageAlias { portuges } { portuguese }
4225 </package>
4226 <*dict-portuguese>
4227 namesep = {\nobreakspace} ,
4228 pairsep = {\~e\nobreakspace} ,
4229 listsep = { ,~ } ,
4230 lastsep = {\~e\nobreakspace} ,
4231 tpairsep = {\~e\nobreakspace} ,
4232 tlistsep = { ,~ } ,
4233 tlastsep = {\~e\nobreakspace} ,
4234 notesep = {~} ,
4235 rangesep = {\~a\nobreakspace} ,
4236
4237 type = part ,
4238 gender = f ,

```



```

4239 Name-sg = Parte ,
4240 name-sg = parte ,
4241 Name-pl = Partes ,
4242 name-pl = partes ,
4243
4244 type = chapter ,
4245 gender = m ,
4246 Name-sg = Capítulo ,
4247 name-sg = capítulo ,
4248 Name-pl = Capítulos ,
4249 name-pl = capítulos ,
4250
4251 type = section ,
4252 gender = f ,
4253 Name-sg = Seção ,
4254 name-sg = seção ,
4255 Name-pl = Seções ,
4256 name-pl = seções ,
4257
4258 type = paragraph ,
4259 gender = m ,
4260 Name-sg = Parágrafo ,
4261 name-sg = parágrafo ,
4262 Name-pl = Parágrafos ,
4263 name-pl = parágrafos ,
4264 Name-sg-ab = Par. ,
4265 name-sg-ab = par. ,
4266 Name-pl-ab = Par. ,
4267 name-pl-ab = par. ,
4268
4269 type = appendix ,
4270 gender = m ,
4271 Name-sg = Apêndice ,
4272 name-sg = apêndice ,
4273 Name-pl = Apêndices ,
4274 name-pl = apêndices ,
4275
4276 type = subappendix ,
4277 gender = m ,
4278 Name-sg = Apêndice ,
4279 name-sg = apêndice ,
4280 Name-pl = Apêndices ,
4281 name-pl = apêndices ,
4282
4283 type = page ,
4284 gender = f ,
4285 Name-sg = Página ,
4286 name-sg = página ,
4287 Name-pl = Páginas ,
4288 name-pl = páginas ,
4289 name-sg-ab = p. ,
4290 name-pl-ab = pp. ,
4291 rangesep = {\textendash} ,
4292

```

```

4293 type = line ,
4294     gender = f ,
4295     Name-sg = Linha ,
4296     name-sg = linha ,
4297     Name-pl = Linhas ,
4298     name-pl = linhas ,
4299
4300 type = figure ,
4301     gender = f ,
4302     Name-sg = Figura ,
4303     name-sg = figura ,
4304     Name-pl = Figuras ,
4305     name-pl = figuras ,
4306     Name-sg-ab = Fig. ,
4307     name-sg-ab = fig. ,
4308     Name-pl-ab = Figs. ,
4309     name-pl-ab = figs. ,
4310
4311 type = table ,
4312     gender = f ,
4313     Name-sg = Tabela ,
4314     name-sg = tabela ,
4315     Name-pl = Tabelas ,
4316     name-pl = tabelas ,
4317
4318 type = item ,
4319     gender = m ,
4320     Name-sg = Item ,
4321     name-sg = item ,
4322     Name-pl = Itens ,
4323     name-pl = itens ,
4324
4325 type = footnote ,
4326     gender = f ,
4327     Name-sg = Nota ,
4328     name-sg = nota ,
4329     Name-pl = Notas ,
4330     name-pl = notas ,
4331
4332 type = note ,
4333     gender = f ,
4334     Name-sg = Nota ,
4335     name-sg = nota ,
4336     Name-pl = Notas ,
4337     name-pl = notas ,
4338
4339 type = equation ,
4340     gender = f ,
4341     Name-sg = Equação ,
4342     name-sg = equação ,
4343     Name-pl = Equações ,
4344     name-pl = equações ,
4345     Name-sg-ab = Eq. ,
4346     name-sg-ab = eq. ,

```

```

4347 Name-pl-ab = Eqs. ,
4348 name-pl-ab = eqs. ,
4349 refpre-in = {} ,
4350 refpos-in = {} ,
4351
4352 type = theorem ,
4353 gender = m ,
4354 Name-sg = Teorema ,
4355 name-sg = teorema ,
4356 Name-pl = Teoremas ,
4357 name-pl = teoremas ,
4358
4359 type = lemma ,
4360 gender = m ,
4361 Name-sg = Lema ,
4362 name-sg = lema ,
4363 Name-pl = Lemas ,
4364 name-pl = lemas ,
4365
4366 type = corollary ,
4367 gender = m ,
4368 Name-sg = Corolário ,
4369 name-sg = corolário ,
4370 Name-pl = Corolários ,
4371 name-pl = corolários ,
4372
4373 type = proposition ,
4374 gender = f ,
4375 Name-sg = Proposição ,
4376 name-sg = proposição ,
4377 Name-pl = Proposições ,
4378 name-pl = proposições ,
4379
4380 type = definition ,
4381 gender = f ,
4382 Name-sg = Definição ,
4383 name-sg = definição ,
4384 Name-pl = Definições ,
4385 name-pl = definições ,
4386
4387 type = proof ,
4388 gender = f ,
4389 Name-sg = Demonstração ,
4390 name-sg = demonstração ,
4391 Name-pl = Demonstrações ,
4392 name-pl = demonstrações ,
4393
4394 type = result ,
4395 gender = m ,
4396 Name-sg = Resultado ,
4397 name-sg = resultado ,
4398 Name-pl = Resultados ,
4399 name-pl = resultados ,
4400

```

```

4401 type = remark ,
4402   gender = f ,
4403   Name-sg = Observação ,
4404   name-sg = observação ,
4405   Name-pl = Observações ,
4406   name-pl = observações ,
4407
4408 type = example ,
4409   gender = m ,
4410   Name-sg = Exemplo ,
4411   name-sg = exemplo ,
4412   Name-pl = Exemplos ,
4413   name-pl = exemplos ,
4414
4415 type = algorithm ,
4416   gender = m ,
4417   Name-sg = Algoritmo ,
4418   name-sg = algoritmo ,
4419   Name-pl = Algoritmos ,
4420   name-pl = algoritmos ,
4421
4422 type = listing ,
4423   gender = f ,
4424   Name-sg = Listagem ,
4425   name-sg = listagem ,
4426   Name-pl = Listagens ,
4427   name-pl = listagens ,
4428
4429 type = exercise ,
4430   gender = m ,
4431   Name-sg = Exercício ,
4432   name-sg = exercício ,
4433   Name-pl = Exercícios ,
4434   name-pl = exercícios ,
4435
4436 type = solution ,
4437   gender = f ,
4438   Name-sg = Solução ,
4439   name-sg = solução ,
4440   Name-pl = Soluções ,
4441   name-pl = soluções ,
4442 </dict-portuguese>

```

10.5 Spanish

```

4443 <*package>
4444 \zcDeclareLanguage [ gender = { f , m } ] { spanish }
4445 </package>
4446 <*dict-spanish>
4447 namesep = {\nobreakspace} ,
4448 pairsep = {\~y\nobreakspace} ,
4449 listsep = { , ~ } ,
4450 lastsep = {\~y\nobreakspace} ,
4451 tpairsep = {\~y\nobreakspace} ,

```

```

4452 tlistsep = {,~} ,
4453 tlastsep = {~y\nobreakspace} ,
4454 notesep = {~} ,
4455 rangesep = {~a\nobreakspace} ,
4456
4457 type = part ,
4458   gender = f ,
4459   Name-sg = Parte ,
4460   name-sg = parte ,
4461   Name-pl = Partes ,
4462   name-pl = partes ,
4463
4464 type = chapter ,
4465   gender = m ,
4466   Name-sg = Capítulo ,
4467   name-sg = capítulo ,
4468   Name-pl = Capítulos ,
4469   name-pl = capítulos ,
4470
4471 type = section ,
4472   gender = f ,
4473   Name-sg = Sección ,
4474   name-sg = sección ,
4475   Name-pl = Secciones ,
4476   name-pl = secciones ,
4477
4478 type = paragraph ,
4479   gender = m ,
4480   Name-sg = Párrafo ,
4481   name-sg = párrafo ,
4482   Name-pl = Párrafos ,
4483   name-pl = párrafos ,
4484
4485 type = appendix ,
4486   gender = m ,
4487   Name-sg = Apéndice ,
4488   name-sg = apéndice ,
4489   Name-pl = Apéndices ,
4490   name-pl = apéndices ,
4491
4492 type = subappendix ,
4493   gender = m ,
4494   Name-sg = Apéndice ,
4495   name-sg = apéndice ,
4496   Name-pl = Apéndices ,
4497   name-pl = apéndices ,
4498
4499 type = page ,
4500   gender = f ,
4501   Name-sg = Página ,
4502   name-sg = página ,
4503   Name-pl = Páginas ,
4504   name-pl = páginas ,
4505   rangesep = {\textendash} ,

```

```

4506
4507 type = line ,
4508     gender = f ,
4509     Name-sg = Línea ,
4510     name-sg = línea ,
4511     Name-pl = Líneas ,
4512     name-pl = líneas ,
4513
4514 type = figure ,
4515     gender = f ,
4516     Name-sg = Figura ,
4517     name-sg = figura ,
4518     Name-pl = Figuras ,
4519     name-pl = figuras ,
4520
4521 type = table ,
4522     gender = m ,
4523     Name-sg = Cuadro ,
4524     name-sg = cuadro ,
4525     Name-pl = Cuadros ,
4526     name-pl = cuadros ,
4527
4528 type = item ,
4529     gender = m ,
4530     Name-sg = Punto ,
4531     name-sg = punto ,
4532     Name-pl = Puntos ,
4533     name-pl = puntos ,
4534
4535 type = footnote ,
4536     gender = f ,
4537     Name-sg = Nota ,
4538     name-sg = nota ,
4539     Name-pl = Notas ,
4540     name-pl = notas ,
4541
4542 type = note ,
4543     gender = f ,
4544     Name-sg = Nota ,
4545     name-sg = nota ,
4546     Name-pl = Notas ,
4547     name-pl = notas ,
4548
4549 type = equation ,
4550     gender = f ,
4551     Name-sg = Ecuación ,
4552     name-sg = ecuación ,
4553     Name-pl = Ecuaciones ,
4554     name-pl = ecuaciones ,
4555     refpre-in = {(} ,
4556     refpos-in = {)} ,
4557
4558 type = theorem ,
4559     gender = m ,

```

```

4560 Name-sg = Teorema ,
4561 name-sg = teorema ,
4562 Name-pl = Teoremas ,
4563 name-pl = teoremas ,
4564
4565 type = lemma ,
4566 gender = m ,
4567 Name-sg = Lema ,
4568 name-sg = lema ,
4569 Name-pl = Lemas ,
4570 name-pl = lemas ,
4571
4572 type = corollary ,
4573 gender = m ,
4574 Name-sg = Corolario ,
4575 name-sg = corolario ,
4576 Name-pl = Corolarios ,
4577 name-pl = corolarios ,
4578
4579 type = proposition ,
4580 gender = f ,
4581 Name-sg = Proposición ,
4582 name-sg = proposición ,
4583 Name-pl = Proposiciones ,
4584 name-pl = proposiciones ,
4585
4586 type = definition ,
4587 gender = f ,
4588 Name-sg = Definición ,
4589 name-sg = definición ,
4590 Name-pl = Definiciones ,
4591 name-pl = definiciones ,
4592
4593 type = proof ,
4594 gender = f ,
4595 Name-sg = Demostración ,
4596 name-sg = demostración ,
4597 Name-pl = Demostraciones ,
4598 name-pl = demostraciones ,
4599
4600 type = result ,
4601 gender = m ,
4602 Name-sg = Resultado ,
4603 name-sg = resultado ,
4604 Name-pl = Resultados ,
4605 name-pl = resultados ,
4606
4607 type = remark ,
4608 gender = f ,
4609 Name-sg = Observación ,
4610 name-sg = observación ,
4611 Name-pl = Observaciones ,
4612 name-pl = observaciones ,
4613

```

```

4614 type = example ,
4615   gender = m ,
4616   Name-sg = Ejemplo ,
4617   name-sg = ejemplo ,
4618   Name-pl = Ejemplos ,
4619   name-pl = ejemplos ,
4620
4621 type = algorithm ,
4622   gender = m ,
4623   Name-sg = Algoritmo ,
4624   name-sg = algoritmo ,
4625   Name-pl = Algoritmos ,
4626   name-pl = algoritmos ,
4627
4628 type = listing ,
4629   gender = m ,
4630   Name-sg = Listado ,
4631   name-sg = listado ,
4632   Name-pl = Listados ,
4633   name-pl = listados ,
4634
4635 type = exercise ,
4636   gender = m ,
4637   Name-sg = Ejercicio ,
4638   name-sg = ejercicio ,
4639   Name-pl = Ejercicios ,
4640   name-pl = ejercicios ,
4641
4642 type = solution ,
4643   gender = f ,
4644   Name-sg = Solución ,
4645   name-sg = solución ,
4646   Name-pl = Soluciones ,
4647   name-pl = soluciones ,
4648 </dict-spanish>

```

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.

	Symbols	. 100, 651, 666, 810, 846, 871, 909,
<code>\</code>	112, 118, 127, 128, 133, 134,	911, 961, 1142, 1158, 1179, 3089,
	139, 140, 145, 146, 198, 206, 207, 217	3091, 3096, 3109, 3129, 3144, 3146,
<code>\{</code>	198	3151, 3167, 3180, 3200, 3212, 3244,
<code>\}</code>	198	3253, 3289, 3293, 3314, 3320, 3343
<code>†</code>	internal commands:	<code>\appendix</code> 80, 81
	<code>\l_zrefclever_current_counter_tl</code>	<code>\appendixname</code> 80
	5	
	A	B
<code>\AddToHook</code>		<code>\babelname</code> 856
		<code>\babelprovide</code> 14, 27

\begin	84	\cs_new_eq:NN	3182, 3187, 3196, 3202, 3208
bool commands:		\cs_new_protected:Npn	246, 380, 448, 456, 462, 633, 969, 1316, 1405, 1411, 1551, 1610, 1652, 1663, 1722, 1852, 1904, 1949, 2105, 2370, 2526, 2528, 2707, 2927, 3003, 3060, 3260
\bool_case_true:	2	\cs_new_protected:Npx	99
\bool_if:NTF	430, 441, 814, 818, 1585, 2003, 2098, 2228, 2250, 2281, 2327, 2376, 2399, 2403, 2409, 2419, 2425, 2582, 2744, 2746	\cs_set_eq:NN	103, 3183, 3188, 3197, 3203, 3209
\bool_if:nTF	68, 1670, 1679, 1688, 1743, 1753, 1777, 1794, 1809, 1874, 1882, 2017, 2025, 2262, 2269, 2276, 2534, 2655	\cs_set_nopar:Npn	3171
\bool_lazy_all:nTF	2348, 2756, 3009		
\bool_lazy_and:nnTF	1559, 1577, 2729, 2771, 3066	E	
\bool_lazy_any:nTF	2889, 2898	\endinput	12
\bool_lazy_or:nnTF	1563, 2717	exp commands:	
\bool_new:N	379, 687, 688, 713, 737, 746, 753, 754, 767, 768, 787, 788, 1077, 1078, 1079, 1080, 1081, 1172, 1173, 1593, 1606, 1914, 1915, 1925, 1932, 1933, 3252	\exp_args:NNe	35, 38
\bool_set:Nn	1556	\exp_args:NNNo	248
\bool_set_false:N	700, 704, 795, 804, 805, 820, 1089, 1093, 1100, 1108, 1109, 1110, 1195, 1735, 1972, 2009, 2023, 2037, 2240, 2374, 2375, 2896, 2913	\exp_args:NNnx	370
\bool_set_true:N	451, 694, 695, 699, 705, 794, 799, 800, 1065, 1087, 1094, 1099, 1116, 1118, 1120, 1123, 1124, 1125, 1183, 1188, 1749, 1759, 1763, 1785, 1800, 1815, 1839, 1980, 2004, 2010, 2014, 2041, 2047, 2912, 2948, 2955, 2956, 2974, 2981, 2993, 3259	\exp_args:NNo	248, 254
\bool_until_do:Nn	1775, 1973	\exp_args:NNx	397, 413, 971, 976, 1017, 1373, 1389
\bool_while_do:nn	3348	\exp_args:Nnx	458
		\exp_args:No	254
C		\exp_args:Nx	391, 3265, 3273
clist commands:		\exp_args:Nxx	1706, 2940, 2962, 2966, 2983
\clist_map_inline:nn	1111, 1238, 3226, 3307	\exp_not:N	68, 2283, 2286, 2297, 2300, 2303, 2540, 2543, 2546, 2555, 2557, 2560, 2563, 2569, 2571, 2589, 2599, 2602, 2604, 2607, 2614, 2621, 2623, 2627, 2630, 2633, 2645, 2648, 2661, 2664, 2667, 2683, 2685, 2688, 2691, 2698, 2700
\counterwithin	4	\exp_not:n	255, 2127, 2143, 2155, 2160, 2183, 2197, 2201, 2213, 2217, 2251, 2252, 2284, 2296, 2301, 2302, 2439, 2452, 2459, 2483, 2495, 2499, 2509, 2513, 2541, 2542, 2544, 2550, 2553, 2556, 2561, 2562, 2564, 2565, 2568, 2570, 2600, 2601, 2603, 2605, 2606, 2608, 2609, 2613, 2625, 2626, 2631, 2632, 2634, 2642, 2646, 2647, 2649, 2662, 2663, 2665, 2677, 2681, 2684, 2689, 2690, 2692, 2693, 2697, 2699
\cref	83	\ExplSyntaxOn	15, 393
cs commands:			
\cs_generate_variant:Nn	65, 251, 257, 447, 455, 1318, 1410, 1416, 2576, 2926	F	
\cs_if_exist:NTF	25, 28, 46, 49, 58, 78, 3117, 3123	file commands:	
\cs_if_exist_p:N	3350	\file_get:nnNTF	391
\cs_new:Npn	56, 66, 76, 87, 252, 258, 2530, 2577, 2916	\fmtversion	3
		\footnote	80

G		1205, 1234, 1260, 1284, 1294, 1305, 1329, 1341, 1417, 1477, 1498, 1521
group commands:		<code>\keys_set:nn</code> . . . 15, 29, 30, 39, 44, 334, 423, 1189, 1317, 1324, 1399, 1554
<code>\group_begin:</code>		keyval commands: <code>\keyval_parse:nnn</code> 1209, 1264
. 102, 325, 382, 450, 1368, 1553, 1567, 2283, 2300, 2540, 2543, 2560, 2563, 2599, 2604, 2607, 2623, 2630, 2645, 2661, 2664, 2688, 2691		
<code>\group_end:</code> 105, 337, 445, 453, 1402, 1570, 1590, 2297, 2303, 2555, 2557, 2569, 2571, 2602, 2614, 2621, 2627, 2633, 2648, 2683, 2685, 2698, 2700		
I		L
<code>\IfBooleanTF</code> 1596		<code>\label</code> 82, 83, 86
<code>\ifdraft</code> 1092		<code>\labelformat</code> 3
<code>\IfFormatAtLeastTF</code> 3, 4, 3176		<code>\language</code> 26, 850
<code>\ifoptionfinal</code> 1098		
<code>\input</code> 14, 15		
int commands:		M
<code>\int_case:nnTF</code>		<code>\mainbabelname</code> 26, 857
. . . 2108, 2136, 2168, 2330, 2432, 2471		<code>\MessageBreak</code> 10
<code>\int_compare:nNnTF</code>		MH commands:
. 1710, 1786, 1801, 1816, 1828, 1840, 1860, 1862, 1906, 2069, 2123, 2157, 2319, 2321, 2387, 2412, 2456, 2944, 2950, 2970, 2976, 3366		<code>\MH_if_boolean:nTF</code> 3257
<code>\int_compare_p:nNn</code> 1876, 1884, 2352, 2721, 2732, 2761, 2909		msg commands:
<code>\int_eval:n</code> 99		<code>\msg_info:nnn</code> 505, 531, 561, 3163, 3248, 3284, 3339, 3367
<code>\int_incr:N</code> 2365, 2402, 2404, 2418, 2420, 2424, 2426, 2524, 3364		<code>\msg_info:nnnn</code> 480, 487, 512
<code>\int_new:N</code>		<code>\msg_info:nnnnn</code> 499
. . . 1607, 1608, 1916, 1917, 1929, 1930		<code>\msg_line_context:</code>
<code>\int_set:Nn</code> 1861, 1863, 1867, 1870, 3347		. . . 111, 117, 121, 123, 126, 132, 138, 144, 150, 155, 160, 165, 170, 176, 181, 184, 187, 192, 196, 202, 205, 211, 216, 223, 228, 232, 234, 237, 241
<code>\int_to_roman:n</code> 3351, 3358, 3359, 3362		<code>\msg_new:nnn</code> 109, 115, 120, 122, 124, 130, 136, 142, 148, 153, 158, 163, 168, 173, 178, 183, 185, 190, 195, 197, 199, 201, 203, 209, 214, 219, 221, 226, 231, 233, 235, 240, 242, 244
<code>\int_use:N</code> 47, 50, 54, 60		<code>\msg_note:nnn</code> 426
<code>\int_zero:N</code>		<code>\msg_warning:nn</code>
. . . 1854, 1855, 1958, 1959, 1960, 1961, 2364, 2366, 2367, 2519, 2520		. . . 656, 681, 819, 825, 1163, 1200, 2354
<code>\l_tmpa_int</code> 3347, 3351, 3358, 3359, 3362, 3364, 3366		<code>\msg_warning:nnn</code> 329, 374, 432, 442, 897, 942, 959, 1129, 1140, 1266, 1333, 1401, 1454, 1489, 1528, 2062, 2235, 2750, 2766, 3052
iow commands:		<code>\msg_warning:nnnn</code>
<code>\iow_char:N</code> 990, 1007, 1045, 1070, 1211, 1429, 1436, 1466, 2828, 2876
. 112, 118, 127, 128, 133, 134, 139, 140, 145, 146, 198, 206, 207, 217		<code>\msg_warning:nnnnn</code> . . . 1031, 1448, 2787
		<code>\msg_warning:nnnnnn</code> 2794
K		N
keys commands:		<code>\newcounter</code> 4, 3113, 3114
<code>\keys_define:nn</code>		<code>\NewDocumentCommand</code>
. . . 39, 340, 468, 523, 540, 554, 640, 670, 677, 689, 714, 723, 738, 747, 755, 769, 781, 789, 822, 829, 867, 914, 956, 963, 1083, 1144, 1151, 1153, 1160, 1167, 1174, 1184, 1196,		323, 364, 1314, 1319, 1366, 1549, 1594
		<code>\nobreakspace</code> 582, 3383, 3384, 3386, 3387, 3389, 3391, 3584, 3585, 3587, 3588, 3590, 3592, 4018, 4019, 4021, 4022, 4024, 4026, 4227, 4228, 4230, 4231, 4233, 4235, 4447, 4448, 4450, 4451, 4453, 4455

P		
\PackageError	7	\seq_if_in:NnTF 387, 484, 509, 1004, 1042, 1240, 1433, 1458, 1656
\pagenumbering	7	\seq_map_break:n 90, 1895, 1898
\pageref	45	\seq_map_function:NN 1615
prg commands:		\seq_map_indexed_inline:Nn 24, 1856
\prg_generate_conditional_		\seq_map_inline:Nn 520, 537, 551, 1291, 1326, 1338, 1474, 1495, 1518, 1892, 3263
variant:Nnn	606, 622	\seq_map_tokens:Nn 72
\prg_new_protected_conditional:Npnn	592, 608, 625	\seq_new:N 263, 264, 308, 315, 378, 722, 1233, 1592, 1609, 1913
\prg_return_false:	602, 604, 618, 620, 631	\seq_pop_left:NN 1975
\prg_return_true:	601, 617, 630	\seq_put_right:Nn 1242, 1659
\ProcessKeysOptions	1313	\seq_reverse:N 728
prop commands:		\seq_set_eq:NN 1951
\prop_get:NnN	3029	\seq_set_from_clist:Nn 397, 413, 727, 976, 1017, 1373, 1389, 1555
\prop_get:NnNTF	384, 595, 598, 611, 614, 628, 971, 1369, 2807, 2835, 2843, 3006, 3063, 3076	\seq_sort:Nn 48, 1618
\prop_gput:Nnn	331, 344, 351, 358, 371, 1407, 1413	\setcounter 3115, 3116, 3132, 3145, 3149
\prop_gput_if_new:Nnn	458, 464	sort commands:
\prop_gset_from_keyval:Nn	576	\sort_return_same: 48, 53, 1625, 1630, 1677, 1715, 1717, 1750, 1770, 1791, 1806, 1820, 1845, 1880, 1895, 1911
\prop_if_exist:NTF	1321	\sort_return_swapped: 48, 53, 1638, 1686, 1714, 1760, 1769, 1790, 1805, 1821, 1844, 1888, 1898, 1910
\prop_if_exist_p:N	3013, 3069	\stepcounter 3131, 3148
\prop_if_in:NnTF	35, 328, 368, 892, 937	str commands:
\prop_if_in_p:Nn	69, 3020	\str_case:nnTF 873, 918, 1113
\prop_item:Nn	38, 70, 372, 400, 416, 979, 1020, 1057, 1376, 1392	\str_compare:nNnTF 1766
\prop_new:N	322, 332, 575, 1204, 1259, 1290, 1322	\str_if_eq:nnTF 89, 1055
\prop_put:Nnn	637, 1301, 1356	\str_if_eq_p:nn 2894, 2900, 2902, 2906
\prop_remove:Nn	636, 1300, 1348	\str_new:N 828
\providecommand	3	\str_set:Nn 833, 835, 837, 839
\ProvidesExplPackage	14	\string 3270, 3278
\ProvidesFile	14	
R		
\refstepcounter	3, 80, 84, 87	
\renewlist	87	
\RequirePackage	16, 17, 18, 19, 20, 815, 1155, 1176	
S		
\scantokens	81	
seq commands:		
\seq_clear:N	734, 1612	
\seq_const_from_clist:Nn	265, 273, 286, 298, 301	
\seq_gconcat:NNN	309, 312, 316, 319	
\seq_get_left:NN	410, 489, 1000, 1011, 1386, 1438, 1983	
\seq_gput_right:Nn	424, 435	
\seq_if_empty:NTF	407, 478, 497, 986, 1027, 1383, 1427, 1446, 1977	
T		
\tag	84, 86	
TeX and L ^A T _E X 2 _ε commands:		
\@Alph	80	
\@addtoreset	4	
\@auxout	3269, 3277	
\@bsphack	383, 3262	
\@chapapp	80	
\@currentcounter	3, 5, 37, 80, 84, 87, 28, 29, 49, 50, 1288	
\@currentlabel	3, 84, 87	
\@elt	5	
\@esphack	444, 3282	
\@ifl@t@r	3	
\@ifpackageloaded	653, 668, 812, 848, 854, 1181, 3111, 3169, 3178, 3192, 3194, 3255, 3291, 3322, 3345	

<code>\@onlypreamble</code>	339, 377, 1404	<code>\tl_head:N</code>	
<code>\@raw@opt@<package>.sty</code>	33	.. 1804, 1817, 1829, 1831, 1841, 1843	
<code>\bbl@loaded</code>	27	<code>\tl_if_empty:NTF</code>	80, 503,
<code>\bbl@main@language</code>	26, 851	528, 545, 559, 565, 988, 998, 1029,	
<code>\c@</code>	4	1040, 1068, 1452, 1482, 1503, 1526,	
<code>\c@enumN</code>	87	1532, 1571, 2060, 2230, 2639, 2726,	
<code>\c@lstnumber</code>	87	2748, 2785, 2805, 2815, 2851, 3335	
<code>\c@page</code>	7, 103	<code>\tl_if_empty:nTF</code>	326,
<code>\cl@</code>	5	366, 472, 635, 1421, 2179, 2195,	
<code>\hyper@@link</code> 67, 2286, 2546, 2589, 2667		2211, 2450, 2481, 2493, 2507, 2712	
<code>\lst@AddToHook</code>	3333	<code>\tl_if_empty_p:N</code>	
<code>\lst@label</code>	3335, 3336	.. 1674, 1675, 1683, 1684, 1691,	
<code>\ltx@gobble</code>	82	1692, 2020, 2021, 2028, 2030, 2759,	
<code>\ltx@label</code> 83, 3182, 3183, 3187, 3188,		2773, 2893, 2903, 2907, 3011, 3067	
3196, 3197, 3202, 3203, 3208, 3209		<code>\tl_if_empty_p:n</code>	1745, 1746,
<code>\MT@newlabel</code>	3270, 3278	1755, 1756, 1781, 1782, 1797, 1812	
<code>\p@...</code>	3	<code>\tl_if_eq:NNTF</code>	
<code>\protected@write</code>	3269, 3277	.. 1695, 1739, 2033, 2781, 2934	
<code>\zref@addprop</code> 22, 32, 43, 53, 55, 97, 108		<code>\tl_if_eq:NnTF</code>	1613, 1645,
<code>\zref@default</code>	67, 68, 2527, 2529	1866, 1869, 1894, 1897, 1987, 2938	
<code>\zref@extractdefault</code>		<code>\tl_if_eq:nTF</code>	1706, 1858,
.. 10, 75, 249, 255, 259		2940, 2962, 2966, 2983, 3265, 3273	
<code>\zref@ifpropundefined</code>	22, 2918	<code>\tl_if_novalue:nTF</code>	1299, 1346
<code>\zref@ifrefcontainsprop</code>		<code>\tl_map_break:n</code>	90
.. 22, 2532, 2584, 2651, 2921		<code>\tl_map_tokens:Nn</code>	82
<code>\zref@ifrefundefined</code>		<code>\tl_new:N</code>	98, 260, 261,
1620, 1622, 1634, 2006, 2008, 2013,		262, 639, 843, 844, 845, 955, 1082,	
2057, 2232, 2241, 2378, 2579, 2709		1150, 1166, 1283, 1600, 1601, 1602,	
<code>\zref@label</code>	82, 3174	1603, 1604, 1605, 1918, 1919, 1920,	
<code>\ZREF@mainlist</code> 22, 32, 43, 53, 55, 97, 108		1921, 1922, 1923, 1924, 1926, 1927,	
<code>\zref@newprop</code>		1928, 1931, 1934, 1935, 1936, 1937,	
.. 5, 7, 21, 23, 33, 44, 54, 92, 107		1938, 1939, 1940, 1941, 1942, 1943,	
<code>\zref@refused</code>	2056	1944, 1945, 1946, 1947, 1948, 3087	
<code>\zref@wrapper@babel</code> 44, 82, 1550, 3174		<code>\tl_put_left:Nn</code> . 2265, 2272, 2312,	
<code>\textendash</code> 586, 3440, 3698, 4076, 4291, 4505		2817, 2818, 2853, 2855, 2857, 2859	
<code>\textup</code>	85	<code>\tl_put_right:Nn</code>	2125, 2141,
<code>\the</code>	3	2150, 2181, 2192, 2208, 2437, 2448,	
<code>\thechapter</code>	80	2479, 2491, 2505, 2727, 2728, 2739	
<code>\thelstnumber</code>	87	<code>\tl_reverse:N</code>	1726, 1729
<code>\thepage</code>	6, 7, 104	<code>\tl_set:Nn</code>	
<code>\thesection</code>	80	.. 248, 333, 474, 485, 644, 646,	
tl commands:		648, 654, 657, 673, 682, 850, 851,	
<code>\c_empty_tl</code>	1655, 1666,	856, 857, 860, 861, 864, 877, 885,	
1668, 1725, 1728, 1731, 1733, 1994,		894, 899, 922, 930, 939, 944, 1323,	
1996, 2919, 2922, 2923, 2930, 2932		1423, 1434, 1833, 1835, 1989, 1990,	
<code>\c_novalue_tl</code>	1296, 1343	2114, 2116, 2248, 2279, 2391, 2393,	
<code>\tl_clear:N</code>		2416, 2723, 2724, 2737, 3088, 3090	
.. 396, 408, 473, 994, 1036, 1049,		<code>\tl_set_eq:NN</code>	2358
1073, 1372, 1384, 1422, 1953, 1954,		<code>\tl_tail:N</code>	1834, 1836
1955, 1956, 1957, 1979, 2360, 2361,		<code>\l_tmpa_tl</code>	394, 423, 1573, 1574
2362, 2363, 2401, 2710, 2713, 2741,			
2780, 2827, 2875, 3051, 3082, 3084			
<code>\tl_gset:Nn</code>	104		
		U	
		<code>\upshape</code>	3247

use commands:
 \use:N 26, 29

V

\value 3132, 3149

Z

\zcDeclareLanguage 13, 30,
 44, 323, 3373, 3573, 4011, 4221, 4444
\zcDeclareLanguageAlias
 14, 364, 3374, 3375,
 3376, 3377, 3378, 3379, 3380, 3576,
 3577, 3578, 3579, 3580, 3581, 4012,
 4013, 4014, 4015, 4222, 4223, 4224
\zcLanguageSetup 11, 14–16, 38, 40, 41, 1366
\zcpageref 45, 1594
\zceref 28, 33, 37, 38,
 44–48, 55, 56, 83, 85, 1549, 1597, 1598
\zcRefTypeSetup ... 11, 38, 39, 1319, 3247
\zcsetup 26, 33, 37, 38, 1314
\zlabel 82, 84, 86, 87, 3336
zrefcheck commands:
 \zrefcheck_zceref_beg_label: .. 1562
 \zrefcheck_zceref_end_label_-
 maybe: 1581
 \zrefcheck_zceref_run_checks_on_-
 labels:n 1582
zrefclever internal commands:
 \l_zrefclever_abbrev_bool
 767, 771, 2730
 \l_zrefclever_capitalize_bool ..
 30, 753, 757, 1065, 2718
 \l_zrefclever_capitalize_first_-
 bool 754, 763, 2720
 __zrefclever_counter_reset_by:n
 . 6, 35, 36, 58, 60, 62, 66, 3219, 3300
 __zrefclever_counter_reset_by_-
 aux:nn 73, 76
 __zrefclever_counter_reset_by_-
 aux:nnn 83, 87
 \l_zrefclever_counter_resetby_-
 prop 5, 36, 69, 70, 1259, 1271
 \l_zrefclever_counter_resetters_-
 seq . 5, 35, 36, 72, 1233, 1240, 1243
 \l_zrefclever_counter_type_prop
 4, 35, 35, 38, 1204, 1216
 \l_zrefclever_current_counter_-
 tl 3, 37, 21, 25,
 26, 36, 39, 41, 46, 47, 95, 1283, 1286
 \l_zrefclever_current_language_-
 tl .. 26, 845, 850, 856, 860, 886, 931
 __zrefclever_declare_default_-
 transl:nnn ... 41, 1405, 1484, 1505
 __zrefclever_declare_type_-
 transl:nnnn
 41, 1405, 1460, 1510, 1534, 1540
 __zrefclever_def_extract:Nnnn ..
 10,
 246, 1654, 1665, 1667, 1724, 1727,
 1730, 1732, 1993, 1995, 2929, 2931
 \g__zrefclever_dict_(language)_prop
 15
 \l_zrefclever_dict_decl_case_tl
 260, 408, 411, 485, 490, 565,
 569, 1384, 1387, 1434, 1439, 1532, 1543
 \l_zrefclever_dict_declension_-
 seq 260, 398, 407, 410, 478, 484,
 489, 977, 986, 1000, 1004, 1011,
 1374, 1383, 1386, 1427, 1433, 1438
 \l_zrefclever_dict_gender_seq ..
 260, 414, 497,
 509, 1018, 1027, 1042, 1390, 1446, 1458
 \l_zrefclever_dict_language_tl .
 260, 333,
 345, 352, 359, 385, 389, 392, 403,
 419, 425, 427, 433, 436, 459, 465,
 481, 488, 500, 513, 596, 599, 612,
 615, 973, 982, 1023, 1060, 1370,
 1379, 1395, 1430, 1437, 1449, 1461,
 1467, 1485, 1506, 1511, 1535, 1541
 __zrefclever_extract:nnn
 .. 10, 258, 1711, 1713, 1787, 1789,
 1802, 1819, 1907, 1909, 2945, 2947,
 2951, 2953, 2971, 2973, 2977, 2979
 __zrefclever_extract_unexp:nnn .
 10, 75, 252,
 1707, 1708, 2292, 2548, 2551, 2566,
 2595, 2610, 2673, 2678, 2694, 2919,
 2922, 2923, 2941, 2942, 2963, 2964,
 2967, 2968, 2985, 2989, 3266, 3274
 __zrefclever_extract_url_-
 unexp:n 2288, 2547, 2591, 2669, 2916
 \g__zrefclever_fallback_dict_-
 prop 11, 575, 576, 628
 \l_zrefclever_footnote_type_tl .
 3087, 3088, 3090, 3094
 __zrefclever_get_default_-
 transl:nnN 11, 609, 623
 __zrefclever_get_default_-
 transl:nnNTF 21, 608, 3044
 __zrefclever_get_enclosing_-
 counters_value:n . 5, 6, 56, 61, 94
 __zrefclever_get_fallback_-
 transl:nN 626
 __zrefclever_get_fallback_-
 transl:nNTF 21, 624, 3049

```

\__zrefclever_get_ref:n .....
..... 67, 68, 2128, 2144,
2156, 2161, 2184, 2198, 2202, 2214,
2218, 2253, 2273, 2440, 2453, 2460,
2484, 2496, 2500, 2510, 2514, 2530
\__zrefclever_get_ref_first: ...
..... 67, 68, 71, 2266, 2313, 2577
\__zrefclever_get_ref_font:nN 11,
20, 37, 78, 79, 2089, 2091, 2093, 3060
\__zrefclever_get_ref_string:nN .
..... 11, 19, 37, 78, 1573, 1964,
1966, 1968, 2071, 2073, 2075, 2077,
2079, 2081, 2083, 2085, 2087, 3003
\__zrefclever_get_type_transl:nnnN
..... 11, 593, 607
\__zrefclever_get_type_transl:nnnNTF
20, 592, 2775, 2821, 2863, 2869, 3038
\l_zrefclever_label_a_tl .....
. 54, 1918, 1976, 1994, 2006, 2056,
2057, 2063, 2115, 2128, 2144, 2161,
2202, 2218, 2246, 2253, 2378, 2382,
2392, 2417, 2440, 2461, 2500, 2514
\l_zrefclever_label_b_tl .....
..... 54, 1918,
1979, 1984, 1996, 2008, 2013, 2382
\l_zrefclever_label_count_int ..
..... 54, 1916, 1958,
2069, 2108, 2364, 2387, 2524, 2762
\l_zrefclever_label_enclval_a_-
tl ..... 1600, 1724, 1726, 1781,
1797, 1817, 1829, 1833, 1834, 1841
\l_zrefclever_label_enclval_b_-
tl ..... 1600, 1727, 1729, 1782,
1804, 1812, 1831, 1835, 1836, 1843
\l_zrefclever_label_extdoc_a_tl
..... 1600, 1730,
1740, 1745, 1755, 1768, 2929, 2935
\l_zrefclever_label_extdoc_b_tl
..... 1600, 1732,
1741, 1746, 1756, 1767, 2931, 2936
\l_zrefclever_label_type_a_tl ..
..... 77, 1600, 1655, 1657,
1660, 1666, 1674, 1683, 1691, 1696,
1866, 1894, 1989, 1993, 2020, 2028,
2034, 2060, 2117, 2394, 3011, 3016,
3023, 3032, 3040, 3067, 3072, 3079
\l_zrefclever_label_type_b_tl ..
..... 1600,
1668, 1675, 1684, 1692, 1697, 1869,
1897, 1990, 1995, 2021, 2030, 2035
\__zrefclever_label_type_put_-
new_right:n .... 46, 48, 1616, 1652
\l_zrefclever_label_types_seq ..
.... 48, 1609, 1612, 1656, 1659, 1892

\__zrefclever_labels_in_sequence:nn
..... 55, 76, 2244, 2381, 2927
\g_zrefclever_languages_prop ...
..... 14, 322, 328, 331, 368, 371,
372, 384, 595, 611, 892, 937, 971, 1369
\l_zrefclever_last_of_type_bool
..... 54, 1913, 2004, 2009, 2010,
2014, 2023, 2038, 2042, 2048, 2098
\l_zrefclever_lastsep_tl . 1934,
2080, 2143, 2160, 2183, 2201, 2213
\l_zrefclever_link_star_bool ...
..... 1556, 1592, 2537, 2658, 2892
\l_zrefclever_listsep_tl .....
... 1934, 2078, 2155, 2197, 2439,
2452, 2459, 2483, 2495, 2499, 2509
\l_zrefclever_load_dict_-
verbose_bool ... 379, 430, 441, 451
\g_zrefclever_loaded_dictionaries_-
seq ..... 378, 388, 424, 435
\__zrefclever_ltxlabel:n .....
83, 3171, 3183, 3188, 3197, 3203, 3209
\l_zrefclever_main_language_tl .
..... 26, 844,
851, 857, 861, 865, 878, 900, 923, 945
\__zrefclever_mathtools_showonlyrefs:n
..... 1587, 3260
\l_zrefclever_mathtools_-
showonlyrefs_bool 1585, 3252, 3259
\__zrefclever_name_default: .....
..... 2526, 2641
\l_zrefclever_name_format_-
fallback_tl ..... 1924, 2737,
2741, 2805, 2848, 2858, 2860, 2872
\l_zrefclever_name_format_tl ...
... 1924, 2723, 2724, 2727, 2728,
2738, 2739, 2812, 2817, 2818, 2824,
2829, 2840, 2854, 2855, 2866, 2878
\l_zrefclever_name_in_link_bool
..... 69,
71, 1924, 2281, 2582, 2896, 2912, 2913
\l_zrefclever_namefont_tl 1934,
2090, 2284, 2301, 2600, 2631, 2646
\l_zrefclever_nameinlink_str ...
..... 828, 833,
835, 837, 839, 2894, 2900, 2902, 2906
\l_zrefclever_namesep_tl .....
.. 1934, 2072, 2603, 2634, 2642, 2649
\l_zrefclever_next_is_same_bool
..... 54, 76, 1929,
2375, 2403, 2419, 2425, 2956, 2994
\l_zrefclever_next_maybe_range_-
bool .....
.. 54, 76, 1929, 2240, 2250, 2374,
2399, 2409, 2948, 2955, 2974, 2982

```

\l_zrefclever_noabbrev_first_- bool 768, 777, 2734	_zrefclever_ref_default: 2526, 2574, 2580, 2635, 2703
\l_zrefclever_nudge_comptosing_- bool ... 1079, 1109, 1118, 1124, 2758	\l_zrefclever_ref_gender_tl 1029, 1035, 1036, 1040, 1043, 1048, 1049, 1082, 1146, 2773, 2782, 2789, 2797
\l_zrefclever_nudge_enabled_- bool 1077, 1087, 1089, 1093, 1094, 1099, 1100, 2350, 2744	\l_zrefclever_ref_language_tl 26, 27, 29, 30, 843, 864, 877, 880, 885, 888, 894, 899, 903, 913, 922, 925, 930, 933, 939, 944, 948, 972, 992, 1010, 1033, 1047, 1072, 1557, 2776, 2791, 2799, 2822, 2864, 2870, 3039, 3045
\l_zrefclever_nudge_gender_bool 1081, 1110, 1120, 1125, 2772	\c_zrefclever_ref_options_font_- seq 12, 20, 265
\l_zrefclever_nudge_multitype_- bool ... 1078, 1108, 1116, 1123, 2351	\c_zrefclever_ref_options_- genders_seq 265
\l_zrefclever_nudge_singular_- bool 1080, 1136, 2746	\c_zrefclever_ref_options_- necessarily_not_type_specific_- seq 19, 265, 521, 1327, 1475
_zrefclever_orig_ltxlabel:n 3173, 3182, 3187, 3196, 3202, 3208	\c_zrefclever_ref_options_- possibly_type_specific_seq 19, 265, 538, 1496
_zrefclever_page_format_aux: 99, 103	\l_zrefclever_ref_options_prop . . 37, 39, 1290, 1300, 1301, 3006, 3063
\g_zrefclever_page_format_tl 7, 98, 104, 107	\c_zrefclever_ref_options_- reference_seq 265, 1292
\l_zrefclever_pairsep_tl 1934, 2076, 2127, 2251	\c_zrefclever_ref_options_type_- names_seq 265, 552, 1519
_zrefclever_process_language_- options: 969, 1558	\c_zrefclever_ref_options_- typesetup_seq 265, 1339
_zrefclever_prop_put_non_- empty:Nnn 21, 633, 1215, 1270	\l_zrefclever_ref_property_tl 22, 639, 644, 646, 648, 654, 657, 673, 682, 1613, 1645, 1987, 2532, 2586, 2653, 2938
_zrefclever_provide_dict_- default_transl:nn 17, 456, 529, 546	\l_zrefclever_ref_typeset_font_- tl 1150, 1152, 1568
_zrefclever_provide_dict_type_- transl:nn 17, 456, 510, 547, 566, 568	\l_zrefclever_reffont_in_tl 1934, 2094, 2544, 2564, 2608, 2665, 2692
_zrefclever_provide_dictionary:n 11, 15, 17, 44, 380, 452, 913, 924, 932, 947, 1557	\l_zrefclever_reffont_out_tl 1934, 2092, 2541, 2561, 2605, 2625, 2662, 2689
_zrefclever_provide_dictionary_- verbose:n ... 17, 448, 879, 887, 902	\l_zrefclever_refpos_in_tl 1934, 2088, 2553, 2568, 2613, 2681, 2697
\l_zrefclever_range_beg_label_- tl 54, 1929, 1957, 2156, 2179, 2185, 2195, 2199, 2211, 2215, 2363, 2401, 2416, 2450, 2454, 2481, 2485, 2493, 2497, 2507, 2511	\l_zrefclever_refpos_out_tl 1934, 2084, 2556, 2570, 2626, 2684, 2699
\l_zrefclever_range_count_int 54, 1929, 1960, 2136, 2170, 2366, 2402, 2413, 2418, 2424, 2432, 2473, 2519	\l_zrefclever_refpre_in_tl 1934, 2086, 2550, 2565, 2609, 2677, 2693
\l_zrefclever_range_same_count_- int 54, 1929, 1961, 2123, 2158, 2171, 2367, 2404, 2420, 2426, 2457, 2474, 2520	\l_zrefclever_refpre_out_tl 1934, 2082, 2542, 2562, 2606, 2663, 2690
\l_zrefclever_rangesep_tl 1934, 2074, 2217, 2252, 2513	\l_zrefclever_setup_type_tl 17, 260, 396, 460, 473, 474, 503, 528, 545, 559, 1323, 1351, 1359, 1372, 1422, 1423, 1452, 1462, 1482, 1503, 1512, 1526, 1536, 1542
\l_zrefclever_ref_decl_case_tl . . 30, 955, 965, 988, 993, 994, 998, 1001, 1005, 1009, 1012, 1068, 1071, 1073, 2815, 2819, 2851, 2856, 2861	

\l_zrefclever_sort_decided_bool	\l_zrefclever_typeset_labels_-
..... 1606 , 1735 , 1749 , 1759 ,	seq 53 , 1913 , 1951 , 1975 , 1977 , 1983
1763 , 1775 , 1785 , 1800 , 1815 , 1839	\l_zrefclever_typeset_last_bool
_zrefclever_sort_default:nn 54 , 1913 ,
..... 48 , 1647 , 1663	1972 , 1973 , 1980 , 2003 , 2327 , 2908
_zrefclever_sort_default_-	\l_zrefclever_typeset_name_bool
different_types:nn 688 , 695 , 700 , 705 , 2263 , 2277
..... 24 , 46 , 52 , 1701 , 1852	\l_zrefclever_typeset_queue_-
_zrefclever_sort_default_same_-	curr_tl
type:nn	54 , 67 ,
46 , 49 , 1699 , 1722	71 , 1918 , 1954 , 2125 , 2141 , 2150 ,
_zrefclever_sort_labels:	2181 , 2192 , 2208 , 2230 , 2248 , 2265 ,
..... 46 , 48 , 53 , 1566 , 1610	2272 , 2279 , 2312 , 2334 , 2339 , 2345 ,
_zrefclever_sort_page:nn	2359 , 2360 , 2437 , 2448 , 2479 , 2491 ,
..... 53 , 1646 , 1904	2505 , 2726 , 2748 , 2759 , 2903 , 2907
\l_zrefclever_sort_prior_a_int .	\l_zrefclever_typeset_queue_-
..... 1607 ,	prev_tl . 54 , 1918 , 1953 , 2323 , 2358
1854 , 1860 , 1861 , 1867 , 1877 , 1885	\l_zrefclever_typeset_range_-
\l_zrefclever_sort_prior_b_int .	bool
..... 1607 ,	746 , 749 , 1565 , 2228
1855 , 1862 , 1863 , 1870 , 1878 , 1886	\l_zrefclever_typeset_ref_bool .
\l_zrefclever_tlastsep_tl 687 , 694 , 699 , 704 , 2263 , 2270
..... 1934 , 1969 , 2344	_zrefclever_typeset_refs:
\l_zrefclever_tlistsep_tl 53 , 55 , 56 , 1569 , 1949
..... 1934 , 1967 , 2322	_zrefclever_typeset_refs_last_-
\l_zrefclever_tpairsep_tl	of_type: . 59 , 67 , 69 , 71 , 2100 , 2105
..... 1934 , 1965 , 2338	_zrefclever_typeset_refs_not_-
\l_zrefclever_type_<type>-	last_of_type:
options_prop 55 , 59 , 67 , 76 , 2102 , 2370
39	\l_zrefclever_typeset_sort_bool
\l_zrefclever_type_count_int 713 , 716 , 1564
.. 54 , 71 , 1916 , 1959 , 2319 , 2321 ,	\l_zrefclever_typesort_seq
2330 , 2352 , 2365 , 2721 , 2733 , 2909 24 , 52 , 722 , 727 , 728 , 734 , 1856
\l_zrefclever_type_first_label_-	\l_zrefclever_use_hyperref_bool
tl 54 , 69 , 1918 , 1955 , 2114 , 2232 , 787 , 794 ,
2241 , 2245 , 2273 , 2289 , 2293 , 2361 ,	799 , 804 , 814 , 820 , 2536 , 2657 , 2891
2391 , 2579 , 2585 , 2592 , 2596 , 2611 ,	\l_zrefclever_warn_hyperref_-
2652 , 2670 , 2674 , 2679 , 2695 , 2709	bool
\l_zrefclever_type_first_label_-	788 , 795 , 800 , 805 , 818
type_tl	_zrefclever_zceref:nnn 30 , 1550 , 1551
54 , 71 , 1918 , 1956 ,	_zrefclever_zceref:nnnn 44 , 47 , 1551
2116 , 2236 , 2362 , 2393 , 2712 , 2752 ,	\l_zrefclever_zceref_labels_seq .
2768 , 2777 , 2790 , 2796 , 2810 , 2823 , 47 , 48 , 1555 ,
2830 , 2838 , 2846 , 2865 , 2871 , 2879	1583 , 1588 , 1592 , 1615 , 1618 , 1952
\l_zrefclever_type_name_gender_-	\l_zrefclever_zceref_note_tl ...
tl 1924 , 2779 , 2780 , 2783 , 2785 , 2798 1166 , 1169 , 1571 , 1575
_zrefclever_type_name_setup: ..	\l_zrefclever_zceref_with_check_-
..... 11 , 69 , 2261 , 2707	bool
\l_zrefclever_type_name_tl	1173 , 1188 , 1561 , 1579
..... 69 , 71 ,	_zrefclever_zcsetup:n
1924 , 2296 , 2302 , 2601 , 2632 , 2639 , 38 , 1315 , 1316 , 3093 ,
2647 , 2710 , 2713 , 2813 , 2825 , 2827 ,	3098 , 3119 , 3125 , 3133 , 3153 , 3214 ,
2841 , 2849 , 2867 , 2873 , 2875 , 2893	3245 , 3295 , 3315 , 3324 , 3337 , 3354
\l_zrefclever_typeset_compress_-	\l_zrefclever_zrefcheck_-
bool	available_bool
737 , 740 , 2376 1172 , 1183 , 1195 , 1560 , 1578