

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 | 17 |
| 4.6 | Options | 24 |
| 5 | Configuration | 38 |
| 5.1 | \zcsetup | 38 |
| 5.2 | \zcRefTypeSetup | 39 |
| 5.3 | \zcLanguageSetup | 40 |
| 6 | User interface | 44 |
| 6.1 | \zceref | 44 |
| 6.2 | \zcpageref | 46 |
| 7 | Sorting | 46 |
| 8 | Typesetting | 54 |

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

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

| | | |
|-----------|--|------------|
| 9 | Compatibility | 80 |
| 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.

`\l__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.

```

\l__zrefclever_get_enclosing_counters_value:n {<counter>}

56 \cs_new:Npn \l__zrefclever_get_enclosing_counters_value:n #1
57 {
58   \cs_if_exist:cT { c@ \l__zrefclever_counter_reset_by:n {#1} }
59   {
60     { \int_use:c { c@ \l__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.

```

```

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
```

(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}

340 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
341 {
342   \tl_if_empty:nF {#1}
343   {
344     \prop_if_in:NnTF \g__zrefclever_languages_prop {#2}
345     {
346       \exp_args:NNnx
347       \prop_gput:Nnn \g__zrefclever_languages_prop {#1}
```

```

348         { \prop_item:Nn \g__zrefclever_languages_prop {#2} }
349     }
350     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
351 }
352 }
353 \@onlypreamble \zcDeclareLanguageAlias

```

(End definition for \zcDeclareLanguageAlias.)

```

354 \keys_define:nn { zref-clever / declarelang }
355 {
356     declension .code:n =
357     {
358         \prop_gput:cnn
359         { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
360         { declension } {#1}
361     } ,
362     declension .value_required:n = true ,
363     gender .code:n =
364     {
365         \prop_gput:cnn
366         { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
367         { gender } {#1}
368     } ,
369     gender .value_required:n = true ,
370     allcaps .code:n =
371     {
372         \prop_gput:cnn
373         { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
374         { allcaps } { true }
375     } ,
376     allcaps .value_forbidden:n = true ,
377 }

```

_zrefclever_process_language_options: Auxiliary function for _zrefclever_zcref:nnn, responsible for processing options from \zcDeclareLanguage. It is necessary to separate them from the reference options machinery because their behavior is language dependent, but the language itself can also be set as an option (lang, value stored in \l__zrefclever_ref_language_tl). Hence, we must validate these options after the reference options have been set. It is expected to be called right (or soon) after \keys_set:nn in _zrefclever_zcref:nnn, where current values for \l__zrefclever_ref_language_tl and \l__zrefclever_ref_decl_case_tl are in place.

```

378 \cs_new_protected:Npn \_zrefclever\_process\_language\_options:
379 {
380     \exp_args:NNx \prop_get:NnNTF \g__zrefclever_languages_prop
381     { \l__zrefclever_ref_language_tl }
382     \l__zrefclever_dict_language_tl
383     {

```

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

```

384 \exp_args:NNx \seq_set_from_clist:Nn
385 \l__zrefclever_dict_declension_seq
386 {
387   \prop_item:cn
388   {
389     g__zrefclever_dict_
390     \l__zrefclever_dict_language_tl _prop
391   }
392   { declension }
393 }
394 \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
395 {
396   \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
397   {
398     \msg_warning:nxxx { zref-clever }
399     { language-no-decl-ref }
400     { \l__zrefclever_ref_language_tl }
401     { \l__zrefclever_ref_decl_case_tl }
402     \tl_clear:N \l__zrefclever_ref_decl_case_tl
403   }
404 }
405 {
406   \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
407   {
408     \seq_get_left:NN \l__zrefclever_dict_declension_seq
409     \l__zrefclever_ref_decl_case_tl
410   }
411   {
412     \seq_if_in:NVF \l__zrefclever_dict_declension_seq
413     \l__zrefclever_ref_decl_case_tl
414     {
415       \msg_warning:nxxx { zref-clever }
416       { unknown-decl-case }
417       { \l__zrefclever_ref_decl_case_tl }
418       { \l__zrefclever_ref_language_tl }
419       \seq_get_left:NN \l__zrefclever_dict_declension_seq
420       \l__zrefclever_ref_decl_case_tl
421     }
422   }
423 }

```

Validate the gender (g) option against the declared genders for the reference language. If the user value for the latter does not match the genders declared for the former, clear `\l__zrefclever_ref_gender_tl` and warn.

```

424 \exp_args:NNx \seq_set_from_clist:Nn
425 \l__zrefclever_dict_gender_seq
426 {
427   \prop_item:cn
428   {
429     g__zrefclever_dict_
430     \l__zrefclever_dict_language_tl _prop
431   }
432   { gender }
433 }

```

```

434 \seq_if_empty:NTF \l__zrefclever_dict_gender_seq
435 {
436   \tl_if_empty:NF \l__zrefclever_ref_gender_tl
437   {
438     \msg_warning:nxxx { zref-clever }
439     { language-no-gender }
440     { \l__zrefclever_ref_language_tl }
441     { g }
442     { \l__zrefclever_ref_gender_tl }
443     \tl_clear:N \l__zrefclever_ref_gender_tl
444   }
445 }
446 {
447   \tl_if_empty:NF \l__zrefclever_ref_gender_tl
448   {
449     \seq_if_in:NVF \l__zrefclever_dict_gender_seq
450     \l__zrefclever_ref_gender_tl
451     {
452       \msg_warning:nxxx { zref-clever }
453       { gender-not-declared }
454       { \l__zrefclever_ref_language_tl }
455       { \l__zrefclever_ref_gender_tl }
456       \tl_clear:N \l__zrefclever_ref_gender_tl
457     }
458   }
459 }

```

Ensure `\l__zrefclever_capitalize_bool` is set to `true` when the language was declared with `allcaps` option.

```

460 \str_if_eq:eeT
461 {
462   \prop_item:cn
463   {
464     g__zrefclever_dict_
465     \l__zrefclever_dict_language_tl _prop
466   }
467   { allcaps }
468 }
469 { true }
470 { \bool_set_true:N \l__zrefclever_capitalize_bool }
471 }
472 {

```

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

```

473 \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
474 {
475   \msg_warning:nxxx { zref-clever } { unknown-language-decl }
476   { \l__zrefclever_ref_decl_case_tl }
477   { \l__zrefclever_ref_language_tl }
478   \tl_clear:N \l__zrefclever_ref_decl_case_tl
479 }
480 \tl_if_empty:NF \l__zrefclever_ref_gender_tl
481 {
482   \msg_warning:nxxx { zref-clever }

```



```

483         { language-no-gender }
484         { \l__zrefclever_ref_language_tl }
485         { g }
486         { \l__zrefclever_ref_gender_tl }
487     \tl_clear:N \l__zrefclever_ref_gender_tl
488 }
489 }
490 }

```

(End definition for `__zrefclever_process_language_options::`)

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 `begindocument` 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 `begindocument`. This includes `translator`, `translations`, but also `babel`’s `.ldf` files, and `biblatex`’s `.lbx` files. I’m not really well acquainted with this machinery, but as far as I grasp, they all rely on some variation of `\ProvidesFile` and `\input`. And they can be safely `\input` without generating spurious content, because they rely on being loaded before the document has actually started. As far as I can tell, `babel`’s “on the fly” functionality is not based on the `.ldf` files, but on the `.ini` files, and on `\babelprovide`. And the `.ini` files are not in this form, but actually resemble “configuration files” of sorts, which means they are read and processed somehow else than with just `\input`. So we do the more or less the same here. It seems a reasonable way to ensure we can load 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

| | |
|---|--|
| <code>\g__zrefclever_loaded_dictionaries_seq</code> | Used to keep track of whether a dictionary has already been loaded or not. 491 <code>\seq_new:N \g__zrefclever_loaded_dictionaries_seq</code> (End definition for <code>\g__zrefclever_loaded_dictionaries_seq</code> .) |
| <code>\l__zrefclever_load_dict_verbose_bool</code> | Controls whether <code>__zrefclever_provide_dictionary:n</code> fails silently or verbosely in case of unknown languages or dictionaries not found. 492 <code>\bool_new:N \l__zrefclever_load_dict_verbose_bool</code> (End definition for <code>\l__zrefclever_load_dict_verbose_bool</code> .) |
| <code>__zrefclever_provide_dictionary:n</code> | Load dictionary for known <code>⟨language⟩</code> if it is available and if it has not already been loaded. <pre> __zrefclever_provide_dictionary:n {⟨language⟩} 493 \cs_new_protected:Npn __zrefclever_provide_dictionary:n #1 494 { 495 \group_begin: 496 \@bsphack 497 \prop_get:NnNTF \g__zrefclever_languages_prop {#1} 498 \l__zrefclever_dict_language_tl 499 { 500 \seq_if_in:NVF 501 \g__zrefclever_loaded_dictionaries_seq 502 \l__zrefclever_dict_language_tl 503 { 504 \exp_args:Nx \file_get:nnNTF 505 { zref-clever- \l__zrefclever_dict_language_tl .dict } 506 { \ExplSyntaxOn } 507 \l_tmpa_tl 508 { 509 \tl_clear:N \l__zrefclever_setup_type_tl 510 \exp_args:NNx \seq_set_from_clist:Nn 511 \l__zrefclever_dict_declension_seq 512 { 513 \prop_item:cn 514 { 515 g__zrefclever_dict_ 516 \l__zrefclever_dict_language_tl _prop 517 } 518 { declension } 519 } 520 \seq_if_empty:NTF \l__zrefclever_dict_declension_seq 521 { \tl_clear:N \l__zrefclever_dict_decl_case_tl } 522 { 523 \seq_get_left:NN \l__zrefclever_dict_declension_seq 524 \l__zrefclever_dict_decl_case_tl 525 } </pre> |

```

526         \exp_args:NNx \seq_set_from_clist:Nn
527         \l__zrefclever_dict_gender_seq
528         {
529             \prop_item:cn
530             {
531                 g__zrefclever_dict_
532                 \l__zrefclever_dict_language_tl _prop
533             }
534             { gender }
535         }
536         \keys_set:nV { zref-clever / dictionary } \l_tmpa_tl
537         \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
538         \l__zrefclever_dict_language_tl
539         \msg_note:nmx { zref-clever } { dict-loaded }
540         { \l__zrefclever_dict_language_tl }
541     }
542     {
543         \bool_if:NT \l__zrefclever_load_dict_verbose_bool
544         {
545             \msg_warning:nmx { zref-clever } { dict-not-available }
546             { \l__zrefclever_dict_language_tl }
547         }
548     }

```

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.

```

548         \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
549         \l__zrefclever_dict_language_tl
550     }
551 }
552 }
553 {
554     \bool_if:NT \l__zrefclever_load_dict_verbose_bool
555     { \msg_warning:nnn { zref-clever } { unknown-language-load } {#1} }
556 }
557 \@esphack
558 \group_end:
559 }
560 \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>}
561 \cs_new_protected:Npn \__zrefclever_provide_dictionary_verbose:n #1
562 {
563     \group_begin:
564     \bool_set_true:N \l__zrefclever_load_dict_verbose_bool

```

```

565     \__zrefclever_provide_dictionary:n {#1}
566     \group_end:
567   }
568   \cs_generate_variant:Nn \__zrefclever_provide_dictionary_verbos:n { x }

```

(End definition for __zrefclever_provide_dictionary_verbos:n.)

_zrefclever_provide_dict_type_transl:nn
_zrefclever_provide_dict_default_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 $\langle key \rangle$ and $\langle translation \rangle$ 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>}

569 \cs_new_protected:Npn \__zrefclever_provide_dict_type_transl:nn #1#2
570 {
571   \exp_args:Nnx \prop_gput_if_new:cnn
572   { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
573   { type- \l__zrefclever_setup_type_tl - #1 } {#2}
574 }
575 \cs_new_protected:Npn \__zrefclever_provide_dict_default_transl:nn #1#2
576 {
577   \prop_gput_if_new:cnn
578   { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
579   { default- #1 } {#2}
580 }

```

(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.

```

581 \keys_define:nn { zref-clever / dictionary }
582 {
583   type .code:n =
584   {
585     \tl_if_empty:NTF {#1}
586     { \tl_clear:N \l__zrefclever_setup_type_tl }
587     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
588   } ,
589   case .code:n =
590   {
591     \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
592     {
593       \msg_info:nxxx { zref-clever } { language-no-decl-setup }
594       { \l__zrefclever_dict_language_tl } {#1}
595     }
596     {
597       \seq_if_in:NnTF \l__zrefclever_dict_declension_seq {#1}

```

```

598         { \tl_set:Nn \l__zrefclever_dict_decl_case_tl {#1} }
599     {
600         \msg_info:nnxx { zref-clever } { unknown-decl-case }
601         {#1} { \l__zrefclever_dict_language_tl }
602         \seq_get_left:NN \l__zrefclever_dict_declension_seq
603         \l__zrefclever_dict_decl_case_tl
604     }
605 }
606 },
607 case .value_required:n = true ,
608 gender .code:n =
609 {
610     \seq_if_empty:NTF \l__zrefclever_dict_gender_seq
611     {
612         \msg_info:nnxxx { zref-clever } { language-no-gender }
613         { \l__zrefclever_dict_language_tl } { gender } {#1}
614     }
615     {
616         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
617         {
618             \msg_info:nnn { zref-clever }
619             { option-only-type-specific } { gender }
620         }
621         {
622             \seq_if_in:NnTF \l__zrefclever_dict_gender_seq {#1}
623             { \__zrefclever_provide_dict_type_transl:nn { gender } {#1} }
624             {
625                 \msg_info:nnxx { zref-clever } { gender-not-declared }
626                 { \l__zrefclever_dict_language_tl } {#1}
627             }
628         }
629     }
630 },
631 gender .value_required:n = true ,
632 }
633 \seq_map_inline:Nn
634 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
635 {
636     \keys_define:nn { zref-clever / dictionary }
637     {
638         #1 .value_required:n = true ,
639         #1 .code:n =
640         {
641             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
642             { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
643             {
644                 \msg_info:nnn { zref-clever }
645                 { option-not-type-specific } {#1}
646             }
647         },
648     }
649 }
650 \seq_map_inline:Nn
651 \c__zrefclever_ref_options_possibly_type_specific_seq

```

```

652 {
653   \keys_define:nn { zref-clever / dictionary }
654   {
655     #1 .value_required:n = true ,
656     #1 .code:n =
657     {
658       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
659       { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
660       { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
661     } ,
662   }
663 }
664 \seq_map_inline:Nn
665   \c__zrefclever_ref_options_type_names_seq
666   {
667     \keys_define:nn { zref-clever / dictionary }
668     {
669       #1 .value_required:n = true ,
670       #1 .code:n =
671       {
672         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
673         {
674           \msg_info:nnn { zref-clever }
675           { option-only-type-specific } {#1}
676         }
677         {
678           \tl_if_empty:NTF \l__zrefclever_dict_decl_case_tl
679           { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
680           {
681             \__zrefclever_provide_dict_type_transl:nn
682             { \l__zrefclever_dict_decl_case_tl - #1 } {##1}
683           }
684         }
685       } ,
686     }
687   }

```

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.

```

688 \prop_new:N \g__zrefclever_fallback_dict_prop
689 \prop_gset_from_keyval:Nn \g__zrefclever_fallback_dict_prop

```

```

690 {
691   tpairsep = {,~} ,
692   tlistsep = {,~} ,
693   tlastsep = {,~} ,
694   notesep  = {~} ,
695   namesep  = {\nobreakspace} ,
696   pairsep  = {,~} ,
697   listsep  = {,~} ,
698   lastsep  = {,~} ,
699   rangesep = {\textendash} ,
700   refpre   = {} ,
701   refpos   = {} ,
702   refpre-in = {} ,
703   refpos-in = {} ,
704 }

```

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

705 \prg_new_protected_conditional:Npnn
706   \__zrefclever_get_type_transl:nnnN #1#2#3#4 { F }
707 {
708   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
709   \l__zrefclever_dict_language_tl
710   {
711     \prop_get:cnNTF
712       { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
713       { type- #2 - #3 } #4
714       { \prg_return_true: }
715       { \prg_return_false: }
716   }
717   { \prg_return_false: }
718 }
719 \prg_generate_conditional_variant:Nnn
720   \__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}
\langle tl variable \rangle {\false code}

721 \prg_new_protected_conditional:Npnn
722   \__zrefclever_get_default_transl:nnN #1#2#3 { F }
723 {
724   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}

```

```

725 \l__zrefclever_dict_language_tl
726 {
727   \prop_get:cnNTF
728   { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
729   { default- #2 } #3
730   { \prg_return_true: }
731   { \prg_return_false: }
732 }
733 { \prg_return_false: }
734 }
735 \prg_generate_conditional_variant:Nnn
736 \__zrefclever_get_default_transl:nnN { xnN } { F }

```

(End definition for __zrefclever_get_default_transl:nnNF.)

__zrefclever_get_fallback_transl:nNF 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:nNF {<key>}
{<tl variable>} {<false code>}

737 % {<key>}<tl var to set>
738 \prg_new_protected_conditional:Npnn
739 \__zrefclever_get_fallback_transl:nN #1#2 { F }
740 {
741   \prop_get:NnNTF \g__zrefclever_fallback_dict_prop
742   { #1 } #2
743   { \prg_return_true: }
744   { \prg_return_false: }
745 }

```

(End definition for __zrefclever_get_fallback_transl:nNF.)

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>}

746 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
747 {
748   \tl_if_empty:nTF {#3}
749   { \prop_remove:Nn #1 {#2} }
750   { \prop_put:Nnn #1 {#2} {#3} }
751 }

```

(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.

```
752 \tl_new:N \l__zrefclever_ref_property_tl
753 \keys_define:nn { zref-clever / reference }
754 {
755   ref .choice: ,
756   ref / default .code:n =
757     { \tl_set:Nn \l__zrefclever_ref_property_tl { default } } ,
758   ref / zc@thecnt .code:n =
759     { \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt } } ,
760   ref / page .code:n =
761     { \tl_set:Nn \l__zrefclever_ref_property_tl { page } } ,
762   ref / title .code:n =
763     {
764       \AddToHook { begindocument }
765       {
766         \@ifpackageloaded { zref-titleref }
767         { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
768         {
769           \msg_warning:nn { zref-clever } { missing-zref-titleref }
770           \tl_set:Nn \l__zrefclever_ref_property_tl { default }
771         }
772       }
773     } ,
774   ref .initial:n = default ,
775   ref .default:n = default ,
776   page .meta:n = { ref = page } ,
777   page .value_forbidden:n = true ,
778 }
779 \AddToHook { begindocument }
780 {
781   \@ifpackageloaded { zref-titleref }
782   {
783     \keys_define:nn { zref-clever / reference }
784     {
785       ref / title .code:n =
786         { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
787     }
788   }
789   {
790     \keys_define:nn { zref-clever / reference }
791     {
```

```

792         ref / title .code:n =
793         {
794             \msg_warning:nn { zref-clever } { missing-zref-titleref }
795             \tl_set:Nn \l__zrefclever_ref_property_tl { default }
796         }
797     }
798 }
799 }

```

typeset option

```

800 \bool_new:N \l__zrefclever_typeset_ref_bool
801 \bool_new:N \l__zrefclever_typeset_name_bool
802 \keys_define:nn { zref-clever / reference }
803 {
804     typeset .choice: ,
805     typeset / both .code:n =
806     {
807         \bool_set_true:N \l__zrefclever_typeset_ref_bool
808         \bool_set_true:N \l__zrefclever_typeset_name_bool
809     } ,
810     typeset / ref .code:n =
811     {
812         \bool_set_true:N \l__zrefclever_typeset_ref_bool
813         \bool_set_false:N \l__zrefclever_typeset_name_bool
814     } ,
815     typeset / name .code:n =
816     {
817         \bool_set_false:N \l__zrefclever_typeset_ref_bool
818         \bool_set_true:N \l__zrefclever_typeset_name_bool
819     } ,
820     typeset .initial:n = both ,
821     typeset .value_required:n = true ,
822
823     noname .meta:n = { typeset = ref },
824     noname .value_forbidden:n = true ,
825 }

```

sort option

```

826 \bool_new:N \l__zrefclever_typeset_sort_bool
827 \keys_define:nn { zref-clever / reference }
828 {
829     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
830     sort .initial:n = true ,
831     sort .default:n = true ,
832     nosort .meta:n = { sort = false },
833     nosort .value_forbidden:n = true ,
834 }

```

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`.

```

835 \seq_new:N \l__zrefclever_typesort_seq
836 \keys_define:nn { zref-clever / reference }
837 {
838   typesort .code:n =
839   {
840     \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
841     \seq_reverse:N \l__zrefclever_typesort_seq
842   } ,
843   typesort .initial:n =
844   { part , chapter , section , paragraph } ,
845   typesort .value_required:n = true ,
846   notypesort .code:n =
847   { \seq_clear:N \l__zrefclever_typesort_seq } ,
848   notypesort .value_forbidden:n = true ,
849 }

```

comp option

```

850 \bool_new:N \l__zrefclever_typeset_compress_bool
851 \keys_define:nn { zref-clever / reference }
852 {
853   comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
854   comp .initial:n = true ,
855   comp .default:n = true ,
856   nocomp .meta:n = { comp = false } ,
857   nocomp .value_forbidden:n = true ,
858 }

```

range option

```

859 \bool_new:N \l__zrefclever_typeset_range_bool
860 \keys_define:nn { zref-clever / reference }
861 {
862   range .bool_set:N = \l__zrefclever_typeset_range_bool ,
863   range .initial:n = false ,
864   range .default:n = true ,
865 }

```

cap and capfirst options

```

866 \bool_new:N \l__zrefclever_capitalize_bool
867 \bool_new:N \l__zrefclever_capitalize_first_bool
868 \keys_define:nn { zref-clever / reference }
869 {
870   cap .bool_set:N = \l__zrefclever_capitalize_bool ,
871   cap .initial:n = false ,
872   cap .default:n = true ,
873   nocap .meta:n = { cap = false } ,
874   nocap .value_forbidden:n = true ,
875
876   capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,
877   capfirst .initial:n = false ,
878   capfirst .default:n = true ,

```

```
879 }
```

abbrev and noabbrevfirst options

```
880 \bool_new:N \l__zrefclever_abbrev_bool
881 \bool_new:N \l__zrefclever_noabbrev_first_bool
882 \keys_define:nn { zref-clever / reference }
883 {
884   abbrev .bool_set:N = \l__zrefclever_abbrev_bool ,
885   abbrev .initial:n = false ,
886   abbrev .default:n = true ,
887   noabbrev .meta:n = { abbrev = false },
888   noabbrev .value_forbidden:n = true ,
889
890   noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
891   noabbrevfirst .initial:n = false ,
892   noabbrevfirst .default:n = true ,
893 }
```

S option

```
894 \keys_define:nn { zref-clever / reference }
895 {
896   S .meta:n =
897     { capfirst = true , noabbrevfirst = true },
898   S .value_forbidden:n = true ,
899 }
```

hyperref option

```
900 \bool_new:N \l__zrefclever_use_hyperref_bool
901 \bool_new:N \l__zrefclever_warn_hyperref_bool
902 \keys_define:nn { zref-clever / reference }
903 {
904   hyperref .choice: ,
905   hyperref / auto .code:n =
906     {
907       \bool_set_true:N \l__zrefclever_use_hyperref_bool
908       \bool_set_false:N \l__zrefclever_warn_hyperref_bool
909     } ,
910   hyperref / true .code:n =
911     {
912       \bool_set_true:N \l__zrefclever_use_hyperref_bool
913       \bool_set_true:N \l__zrefclever_warn_hyperref_bool
914     } ,
915   hyperref / false .code:n =
916     {
917       \bool_set_false:N \l__zrefclever_use_hyperref_bool
918       \bool_set_false:N \l__zrefclever_warn_hyperref_bool
919     } ,
920   hyperref .initial:n = auto ,
921   hyperref .default:n = auto
922 }
923 \AddToHook { begindocument }
924 {
925   \@ifpackageloaded { hyperref }
```

```

926     {
927         \bool_if:NT \l__zrefclever_use_hyperref_bool
928         { \RequirePackage { zref-hyperref } }
929     }
930     {
931         \bool_if:NT \l__zrefclever_warn_hyperref_bool
932         { \msg_warning:nn { zref-clever } { missing-hyperref } }
933         \bool_set_false:N \l__zrefclever_use_hyperref_bool
934     }
935     \keys_define:nn { zref-clever / reference }
936     {
937         hyperref .code:n =
938         { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }
939     }
940 }

```

nameinlink option

```

941 \str_new:N \l__zrefclever_nameinlink_str
942 \keys_define:nn { zref-clever / reference }
943 {
944     nameinlink .choice: ,
945     nameinlink / true .code:n =
946     { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
947     nameinlink / false .code:n =
948     { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
949     nameinlink / single .code:n =
950     { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
951     nameinlink / tsingle .code:n =
952     { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
953     nameinlink .initial:n = tsingle ,
954     nameinlink .default:n = true ,
955 }

```

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`.

```

956 \tl_new:N \l__zrefclever_ref_language_tl
957 \tl_new:N \l__zrefclever_main_language_tl
958 \tl_new:N \l__zrefclever_current_language_tl
959 \AddToHook { begindocument }
960 {
961   \ifpackageloaded { babel }
962   {
963     \tl_set:Nn \l__zrefclever_current_language_tl { \language }
964     \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
965   }
966   {
967     \ifpackageloaded { polyglossia }
968     {
969       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
970       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
971     }
972     {
973       \tl_set:Nn \l__zrefclever_current_language_tl { english }
974       \tl_set:Nn \l__zrefclever_main_language_tl { english }
975     }
976   }

```

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.

```

977   \tl_set:Nn \l__zrefclever_ref_language_tl
978   { \l__zrefclever_main_language_tl }
979 }
980 \keys_define:nn { zref-clever / reference }
981 {
982   lang .code:n =
983   {
984     \AddToHook { begindocument }
985     {
986       \str_case:nnF {#1}
987       {
988         { main }
989         {
990           \tl_set:Nn \l__zrefclever_ref_language_tl
991           { \l__zrefclever_main_language_tl }
992           \__zrefclever_provide_dictionary_verbosely:x
993           { \l__zrefclever_ref_language_tl }
994         }
995       }

```

```

996         { current }
997     {
998         \tl_set:Nn \l__zrefclever_ref_language_tl
999             { \l__zrefclever_current_language_tl }
1000         \__zrefclever_provide_dictionary_verbose:x
1001             { \l__zrefclever_ref_language_tl }
1002     }
1003 }
1004 {
1005     \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
1006     {
1007         \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
1008     }
1009     {
1010         \msg_warning:nnn { zref-clever }
1011             { unknown-language-opt } {#1}
1012         \tl_set:Nn \l__zrefclever_ref_language_tl
1013             { \l__zrefclever_main_language_tl }
1014     }
1015     \__zrefclever_provide_dictionary_verbose:x
1016         { \l__zrefclever_ref_language_tl }
1017 }
1018 }
1019 } ,
1020 lang .value_required:n = true ,
1021 }
1022 \AddToHook { begindocument / before }
1023 {
1024     \AddToHook { begindocument }
1025     {

```

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.

```

1026     \__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.

```

1027     \keys_define:nn { zref-clever / reference }
1028     {
1029         lang .code:n =
1030         {
1031             \str_case:nnF {#1}
1032             {
1033                 { main }
1034                 {
1035                     \tl_set:Nn \l__zrefclever_ref_language_tl
1036                         { \l__zrefclever_main_language_tl }
1037                     \__zrefclever_provide_dictionary:x
1038                         { \l__zrefclever_ref_language_tl }
1039                 }
1040             }
1041         { current }

```

```

1042         {
1043             \tl_set:Nn \l__zrefclever_ref_language_tl
1044                 { \l__zrefclever_current_language_tl }
1045             \__zrefclever_provide_dictionary:x
1046                 { \l__zrefclever_ref_language_tl }
1047         }
1048     }
1049     {
1050         \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
1051         {
1052             \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
1053         }
1054         {
1055             \msg_warning:nnn { zref-clever }
1056                 { unknown-language-opt } {#1}
1057             \tl_set:Nn \l__zrefclever_ref_language_tl
1058                 { \l__zrefclever_main_language_tl }
1059         }
1060         \__zrefclever_provide_dictionary:x
1061             { \l__zrefclever_ref_language_tl }
1062     }
1063     } ,
1064     lang .value_required:n = true ,
1065 }
1066 }
1067 }

```

d option

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

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.

```

1068 \tl_new:N \l__zrefclever_ref_decl_case_tl
1069 \keys_define:nn { zref-clever / reference }
1070 {
1071     d .code:n =
1072         { \msg_warning:nnn { zref-clever } { option-document-only } { d } } ,
1073 }
1074 \AddToHook { begindocument }
1075 {
1076     \keys_define:nn { zref-clever / reference }
1077     {

```

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

```

1078         d .tl_set:N = \l__zrefclever_ref_decl_case_tl ,
1079         d .value_required:n = true ,
1080     }
1081 }

```


nudge & Co. options

```

1082 \bool_new:N \l__zrefclever_nudge_enabled_bool
1083 \bool_new:N \l__zrefclever_nudge_multitype_bool
1084 \bool_new:N \l__zrefclever_nudge_comptosing_bool
1085 \bool_new:N \l__zrefclever_nudge_singular_bool
1086 \bool_new:N \l__zrefclever_nudge_gender_bool
1087 \tl_new:N \l__zrefclever_ref_gender_tl
1088 \keys_define:nn { zref-clever / reference }
1089 {
1090   nudge .choice: ,
1091   nudge / true .code:n =
1092     { \bool_set_true:N \l__zrefclever_nudge_enabled_bool } ,
1093   nudge / false .code:n =
1094     { \bool_set_false:N \l__zrefclever_nudge_enabled_bool } ,
1095   nudge / obeydraft .code:n =
1096     {
1097       \ifdraft
1098         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
1099         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
1100     } ,
1101   nudge / obeyfinal .code:n =
1102     {
1103       \ifoptionfinal
1104         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
1105         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
1106     } ,
1107   nudge .initial:n = false ,
1108   nudge .default:n = true ,
1109   nonnudge .meta:n = { nudge = false } ,
1110   nonnudge .value_forbidden:n = true ,
1111   nudgeif .code:n =
1112     {
1113       \bool_set_false:N \l__zrefclever_nudge_multitype_bool
1114       \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
1115       \bool_set_false:N \l__zrefclever_nudge_gender_bool
1116       \clist_map_inline:nn {#1}
1117       {
1118         \str_case:nnF {##1}
1119         {
1120           { multitype }
1121           { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
1122           { comptosing }
1123           { \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
1124           { gender }
1125           { \bool_set_true:N \l__zrefclever_nudge_gender_bool }
1126           { all }
1127           {
1128             \bool_set_true:N \l__zrefclever_nudge_multitype_bool
1129             \bool_set_true:N \l__zrefclever_nudge_comptosing_bool
1130             \bool_set_true:N \l__zrefclever_nudge_gender_bool
1131           }
1132         }
1133     }

```

```

1134         \msg_warning:nnn { zref-clever }
1135         { nudgeif-unknown-value } {##1}
1136     }
1137 }
1138 },
1139 nudgeif .value_required:n = true ,
1140 nudgeif .initial:n = all ,
1141 sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
1142 sg .initial:n = false ,
1143 sg .default:n = true ,
1144 g .code:n =
1145     { \msg_warning:nnn { zref-clever } { option-document-only } { g } } ,
1146 }
1147 \AddToHook { begindocument }
1148 {
1149     \keys_define:nn { zref-clever / reference }
1150     {

```

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

```

1151         g .tl_set:N = \l__zrefclever_ref_gender_tl ,
1152         g .value_required:n = true ,
1153     }
1154 }

```

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).

```

1155 \tl_new:N \l__zrefclever_ref_typeset_font_tl
1156 \keys_define:nn { zref-clever / reference }
1157 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

titleref option

```

1158 \keys_define:nn { zref-clever / reference }
1159 {
1160     titleref .code:n = { \RequirePackage { zref-titleref } } ,
1161     titleref .value_forbidden:n = true ,
1162 }
1163 \AddToHook { begindocument }
1164 {
1165     \keys_define:nn { zref-clever / reference }
1166     {
1167         titleref .code:n =
1168             { \msg_warning:nn { zref-clever } { titleref-preamble-only } }
1169     }
1170 }

```

note option

```

1171 \tl_new:N \l__zrefclever_zcref_note_tl
1172 \keys_define:nn { zref-clever / reference }

```

```

1173 {
1174     note .tl_set:N = \l__zrefclever_zceref_note_tl ,
1175     note .value_required:n = true ,
1176 }

```

check option

Integration with zref-check.

```

1177 \bool_new:N \l__zrefclever_zrefcheck_available_bool
1178 \bool_new:N \l__zrefclever_zceref_with_check_bool
1179 \keys_define:nn { zref-clever / reference }
1180 {
1181     check .code:n = { \RequirePackage { zref-check } } ,
1182     check .value_forbidden:n = true ,
1183 }
1184 \AddToHook { begindocument }
1185 {
1186     \@ifpackageloaded { zref-check }
1187     {
1188         \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
1189         \keys_define:nn { zref-clever / reference }
1190         {
1191             check .code:n =
1192             {
1193                 \bool_set_true:N \l__zrefclever_zceref_with_check_bool
1194                 \keys_set:nn { zref-check / zcheck } {#1}
1195             } ,
1196             check .value_required:n = true ,
1197         }
1198     }
1199     {
1200         \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
1201         \keys_define:nn { zref-clever / reference }
1202         {
1203             check .value_forbidden:n = false ,
1204             check .code:n =
1205             { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
1206         }
1207     }
1208 }

```

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`.

```

1209 \prop_new:N \l__zrefclever_counter_type_prop
1210 \keys_define:nn { zref-clever / label }
1211 {
1212     countertype .code:n =
1213     {
1214         \keyval_parse:nnn
1215         {

```

```

1216         \msg_warning:nnnn { zref-clever }
1217         { key-requires-value } { countertype }
1218     }
1219     {
1220         \__zrefclever_prop_put_non_empty:Nnn
1221         \l__zrefclever_counter_type_prop
1222     }
1223     {#1}
1224 } ,
1225 countertype .value_required:n = true ,
1226 countertype .initial:n =
1227 {
1228     subsection    = section ,
1229     subsubsection = section ,
1230     subparagraph  = paragraph ,
1231     enumi         = item ,
1232     enumii        = item ,
1233     enumiii       = item ,
1234     enumiv        = item ,
1235     mpfootnote    = footnote ,
1236 } ,
1237 }

```

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.

```

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

```

```

1259     paragraph ,
1260     subparagraph ,
1261   },
1262   counterresetters .value_required:n = true ,
1263 }

```

counterresetby option

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

```

1264 \prop_new:N \l__zrefclever_counter_resetby_prop
1265 \keys_define:nn { zref-clever / label }
1266 {
1267   counterresetby .code:n =
1268   {
1269     \keyval_parse:nnn
1270     {
1271       \msg_warning:nnn { zref-clever }
1272       { key-requires-value } { counterresetby }
1273     }
1274     {
1275       \__zrefclever_prop_put_non_empty:Nnn
1276       \l__zrefclever_counter_resetby_prop
1277     }
1278     {#1}
1279   } ,
1280   counterresetby .value_required:n = true ,
1281   counterresetby .initial:n =
1282   {

```

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.

```

1283     enumii = enumi ,
1284     enumiii = enumii ,
1285     enumiv = enumiii ,
1286   } ,
1287 }

```

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.

```

1288 \tl_new:N \l__zrefclever_current_counter_tl
1289 \keys_define:nn { zref-clever / label }
1290 {
1291   currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
1292   currentcounter .value_required:n = true ,

```

```

1293     currentcounter .initial:n = \@currentcounter ,
1294 }

```

Reference options

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

```

1295 \prop_new:N \l__zrefclever_ref_options_prop
1296 \seq_map_inline:Nn
1297   \c__zrefclever_ref_options_reference_seq
1298   {
1299     \keys_define:nn { zref-clever / reference }
1300     {
1301       #1 .default:V = \c_novalue_tl ,
1302       #1 .code:n =
1303       {
1304         \tl_if_novalue:nTF {##1}
1305         { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
1306         { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
1307       } ,
1308     }
1309   }

```

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.

```

1310 \keys_define:nn { }
1311 {
1312   zref-clever / zcsetup .inherit:n =
1313   {
1314     zref-clever / label ,
1315     zref-clever / reference ,
1316   }
1317 }

```

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

```

1318 \ProcessKeysOptions { zref-clever / zcsetup }

```

5 Configuration

5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

```

\zcsetup{<options>}

1319 \NewDocumentCommand \zcsetup { m }
1320 { \__zrefclever_zcsetup:n {#1} }

(End definition for \zcsetup.)

```

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

```

\__zrefclever_zcsetup:n{<options>}

1321 \cs_new_protected:Npn \__zrefclever_zcsetup:n #1
1322 { \keys_set:nn { zref-clever / zcsetup } {#1} }
1323 \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>}

1324 \NewDocumentCommand \zcRefTypeSetup { m m }
1325 {
1326   \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
1327   { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
1328   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
1329   \keys_set:nn { zref-clever / typesetup } {#2}
1330 }

```

(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>.

```

1331 \seq_map_inline:Nn
1332   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
1333   {
1334     \keys_define:nn { zref-clever / typesetup }
1335     {
1336       #1 .code:n =
1337       {
1338         \msg_warning:nnn { zref-clever }
1339         { option-not-type-specific } {#1}
1340       } ,
1341     }
1342   }
1343 \seq_map_inline:Nn
1344   \c__zrefclever_ref_options_typesetup_seq
1345   {
1346     \keys_define:nn { zref-clever / typesetup }
1347     {
1348       #1 .default:V = \c_novalue_tl ,
1349       #1 .code:n =
1350       {
1351         \tl_if_novalue:nTF {##1}
1352         {
1353           \prop_remove:cn
1354           {
1355             l__zrefclever_type_
1356             \l__zrefclever_setup_type_tl _options_prop
1357           }
1358           {#1}
1359         }
1360         {
1361           \prop_put:cnn
1362           {
1363             l__zrefclever_type_
1364             \l__zrefclever_setup_type_tl _options_prop
1365           }
1366           {#1} {##1}
1367         }
1368       } ,
1369     }
1370   }

```

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 $\langle options \rangle$ 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>}
1371 \NewDocumentCommand \zcLanguageSetup { m m }
1372 {
1373   \group_begin:
1374   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
1375   \l__zrefclever_dict_language_tl
1376   {
1377     \tl_clear:N \l__zrefclever_setup_type_tl
1378     \exp_args:NNx \seq_set_from_clist:Nn
1379     \l__zrefclever_dict_declension_seq
1380     {
1381       \prop_item:cn
1382       {
1383         g__zrefclever_dict_
1384         \l__zrefclever_dict_language_tl _prop
1385       }
1386       { declension }
1387     }
1388     \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
1389     { \tl_clear:N \l__zrefclever_dict_decl_case_tl }
1390     {
1391       \seq_get_left:NN \l__zrefclever_dict_declension_seq
1392       \l__zrefclever_dict_decl_case_tl
1393     }
1394     \exp_args:NNx \seq_set_from_clist:Nn
1395     \l__zrefclever_dict_gender_seq
1396     {
1397       \prop_item:cn
1398       {
1399         g__zrefclever_dict_
1400         \l__zrefclever_dict_language_tl _prop
1401       }
1402       { gender }
1403     }
1404     \keys_set:nn { zref-clever / langsetup } {#2}
1405   }
1406   { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
1407   \group_end:
1408 }
1409 \@onlypreamble \zcLanguageSetup

```

(End definition for \zcLanguageSetup.)

_zrefclever_declare_type_transl:nnnn A couple of auxiliary functions for the of zref-clever/translation keys set in
_zrefclever_declare_default_transl:nnn \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>}
1410 \cs_new_protected:Npn \_zrefclever_declare_type_transl:nnnn #1#2#3#4
1411 {
1412   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }

```

```

1413     { type- #2 - #3 } {#4}
1414   }
1415   \cs_generate_variant:Nn \__zrefclever_declare_type_transl:nnnn { VVnn , VVxn }
1416   \cs_new_protected:Npn \__zrefclever_declare_default_transl:nnn #1#2#3
1417   {
1418     \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
1419     { default- #2 } {#3}
1420   }
1421   \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.

```

1422 \keys_define:nn { zref-clever / langsetup }
1423 {
1424   type .code:n =
1425   {
1426     \tl_if_empty:NTF {#1}
1427     { \tl_clear:N \l__zrefclever_setup_type_tl }
1428     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
1429   } ,
1430   case .code:n =
1431   {
1432     \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
1433     {
1434       \msg_warning:nxxx { zref-clever } { language-no-decl-setup }
1435       { \l__zrefclever_dict_language_tl } {#1}
1436     }
1437     {
1438       \seq_if_in:NnTF \l__zrefclever_dict_declension_seq {#1}
1439       { \tl_set:Nn \l__zrefclever_dict_decl_case_tl {#1} }
1440       {
1441         \msg_warning:nxxx { zref-clever } { unknown-decl-case }
1442         {#1} { \l__zrefclever_dict_language_tl }
1443         \seq_get_left:NN \l__zrefclever_dict_declension_seq
1444         \l__zrefclever_dict_decl_case_tl
1445       }
1446     }
1447   } ,
1448   case .value_required:n = true ,
1449   gender .code:n =
1450   {
1451     \seq_if_empty:NTF \l__zrefclever_dict_gender_seq
1452     {
1453       \msg_warning:nxxx { zref-clever } { language-no-gender }
1454       { \l__zrefclever_dict_language_tl } { gender } {#1}
1455     }
1456     {
1457       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1458       {
1459         \msg_warning:nnn { zref-clever }
1460         { option-only-type-specific } { gender }
1461       }

```

```

1462         {
1463             \seq_if_in:NnTF \l__zrefclever_dict_gender_seq {#1}
1464             {
1465                 \__zrefclever_declare_type_transl:VWnn
1466                 \l__zrefclever_dict_language_tl
1467                 \l__zrefclever_setup_type_tl
1468                 { gender } {#1}
1469             }
1470             {
1471                 \msg_warning:nxxx { zref-clever } { gender-not-declared }
1472                 { \l__zrefclever_dict_language_tl } {#1}
1473             }
1474         }
1475     }
1476 },
1477 gender .value_required:n = true ,
1478 }
1479 \seq_map_inline:Nn
1480 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
1481 {
1482     \keys_define:nn { zref-clever / langsetup }
1483     {
1484         #1 .value_required:n = true ,
1485         #1 .code:n =
1486         {
1487             \tl_if_empty:NnTF \l__zrefclever_setup_type_tl
1488             {
1489                 \__zrefclever_declare_default_transl:Vnn
1490                 \l__zrefclever_dict_language_tl
1491                 {#1} {##1}
1492             }
1493             {
1494                 \msg_warning:nnn { zref-clever }
1495                 { option-not-type-specific } {#1}
1496             }
1497         } ,
1498     }
1499 }
1500 \seq_map_inline:Nn
1501 \c__zrefclever_ref_options_possibly_type_specific_seq
1502 {
1503     \keys_define:nn { zref-clever / langsetup }
1504     {
1505         #1 .value_required:n = true ,
1506         #1 .code:n =
1507         {
1508             \tl_if_empty:NnTF \l__zrefclever_setup_type_tl
1509             {
1510                 \__zrefclever_declare_default_transl:Vnn
1511                 \l__zrefclever_dict_language_tl
1512                 {#1} {##1}
1513             }
1514             {
1515                 \__zrefclever_declare_type_transl:VWnn

```

```

1516         \l__zrefclever_dict_language_tl
1517         \l__zrefclever_setup_type_tl
1518         {#1} {##1}
1519     }
1520 } ,
1521 }
1522 }
1523 \seq_map_inline:Nn
1524 \c__zrefclever_ref_options_type_names_seq
1525 {
1526     \keys_define:nn { zref-clever / langsetup }
1527     {
1528         #1 .value_required:n = true ,
1529         #1 .code:n =
1530         {
1531             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1532             {
1533                 \msg_warning:nnn { zref-clever }
1534                 { option-only-type-specific } {#1}
1535             }
1536             {
1537                 \tl_if_empty:NTF \l__zrefclever_dict_decl_case_tl
1538                 {
1539                     \__zrefclever_declare_type_transl:VVnn
1540                     \l__zrefclever_dict_language_tl
1541                     \l__zrefclever_setup_type_tl
1542                     {#1} {##1}
1543                 }
1544                 {
1545                     \__zrefclever_declare_type_transl:VVxn
1546                     \l__zrefclever_dict_language_tl
1547                     \l__zrefclever_setup_type_tl
1548                     { \l__zrefclever_dict_decl_case_tl - #1 } {##1}
1549                 }
1550             }
1551         } ,
1552     }
1553 }

```

6 User interface

6.1 \zcref

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

```
\zcref{*}[\langle options \rangle]{\langle labels \rangle}
```

```

1554 \NewDocumentCommand \zcref { s O { } m }
1555 { \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 `{\langle labels \rangle}` as first argument, so that it can be protected by `\zref@wrapper@babel` in `\zcref`.

```

    \__zrefclever_zcref:nnnn {\<labels\>} {\<*\>} {\<options\>}}
1556 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
1557 {
1558     \group_begin:
Set options.
1559     \keys_set:nn { zref-clever / reference } {#3}
Store arguments values.
1560     \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
1561     \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 lan-
guage may have changed outside our control. \__zrefclever_provide_dictionary:x
does nothing if the dictionary is already loaded.
1562     \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }
Process \zcDeclareLanguage options.
1563     \__zrefclever_process_language_options:
Integration with zref-check.
1564     \bool_lazy_and:nnT
1565     { \l__zrefclever_zrefcheck_available_bool }
1566     { \l__zrefclever_zcref_with_check_bool }
1567     { \zrefcheck_zcref_beg_label: }
Sort the labels.
1568     \bool_lazy_or:nnT
1569     { \l__zrefclever_typeset_sort_bool }
1570     { \l__zrefclever_typeset_range_bool }
1571     { \__zrefclever_sort_labels: }
Typeset the references. Also, set the reference font, and group it, so that it does not leak
to the note.
1572     \group_begin:
1573     \l__zrefclever_ref_typeset_font_tl
1574     \__zrefclever_typeset_refs:
1575     \group_end:
Typeset note.
1576     \tl_if_empty:NF \l__zrefclever_zcref_note_tl
1577     {
1578         \__zrefclever_get_ref_string:nN { notesep } \l_tmpa_tl
1579         \l_tmpa_tl
1580         \l__zrefclever_zcref_note_tl
1581     }
Integration with zref-check.
1582     \bool_lazy_and:nnT
1583     { \l__zrefclever_zrefcheck_available_bool }
1584     { \l__zrefclever_zcref_with_check_bool }
1585     {
1586         \zrefcheck_zcref_end_label_maybe:
1587         \zrefcheck_zcref_run_checks_on_labels:n
1588         { \l__zrefclever_zcref_labels_seq }
1589     }

```

Integration with mathtools.

```

1590 \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
1591 {
1592   \__zrefclever_mathtools_showonlyrefs:n
1593   { \l__zrefclever_zcref_labels_seq }
1594 }
1595 \group_end:
1596 }

```

(End definition for __zrefclever_zcref:nnnn.)

\l__zrefclever_zcref_labels_seq

\l__zrefclever_link_star_bool

```

1597 \seq_new:N \l__zrefclever_zcref_labels_seq
1598 \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{*}[\langle options \rangle]{\langle labels \rangle}

```

1599 \NewDocumentCommand \zcpageref { s O { } m }
1600 {
1601   \IfBooleanTF {#1}
1602   { \zcref*{#2, ref = page} {#3} }
1603   { \zcref [ #2, ref = page] {#3} }
1604 }

```

(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.

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.

\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

```

1605 \tl_new:N \l__zrefclever_label_type_a_tl
1606 \tl_new:N \l__zrefclever_label_type_b_tl
1607 \tl_new:N \l__zrefclever_label_enclval_a_tl
1608 \tl_new:N \l__zrefclever_label_enclval_b_tl
1609 \tl_new:N \l__zrefclever_label_extdoc_a_tl
1610 \tl_new:N \l__zrefclever_label_extdoc_b_tl

```

(End definition for \l__zrefclever_label_type_a_tl and others.)

\l__zrefclever_sort_decided_bool Auxiliary variable for __zrefclever_sort_default_same_type:nn, signals if the sorting between two labels has been decided or not.

```
1611 \bool_new:N \l__zrefclever_sort_decided_bool
```

(End definition for \l__zrefclever_sort_decided_bool.)

\l__zrefclever_sort_prior_a_int \l__zrefclever_sort_prior_b_int Auxiliary variables for __zrefclever_sort_default_different_types:nn. Store the sort priority of the “current” and “next” labels.

```
1612 \int_new:N \l__zrefclever_sort_prior_a_int
```

```
1613 \int_new:N \l__zrefclever_sort_prior_b_int
```

(End definition for \l__zrefclever_sort_prior_a_int and \l__zrefclever_sort_prior_b_int.)

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

```
1614 \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.

```
1615 \cs_new_protected:Npn \__zrefclever_sort_labels:
```

```
1616 {
```

Store label types sequence.

```
1617 \seq_clear:N \l__zrefclever_label_types_seq
```

```
1618 \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
```

```
1619 {
```

```
1620 \seq_map_function:NN \l__zrefclever_zcref_labels_seq
```

```
1621 \__zrefclever_label_type_put_new_right:n
```

```
1622 }
```

Sort.

```
1623 \seq_sort:Nn \l__zrefclever_zcref_labels_seq
```

```
1624 {
```

```
1625 \zref@ifrefundefined {##1}
```

```
1626 {
```

```
1627 \zref@ifrefundefined {##2}
```

```
1628 {
```

```
1629 % Neither label is defined.
```

```
1630 \sort_return_same:
```

```
1631 }
```

```
1632 {
```

```
1633 % The second label is defined, but the first isn't, leave the
```

```
1634 % undefined first (to be more visible).
```

```
1635 \sort_return_same:
```

```

1636     }
1637   }
1638   {
1639     \zref@ifrefundefined {##2}
1640     {
1641       % The first label is defined, but the second isn't, bring the
1642       % second forward.
1643       \sort_return_swapped:
1644     }
1645     {
1646       % The interesting case: both labels are defined. References
1647       % to the "default" property or to the "page" are quite
1648       % different with regard to sorting, so we branch them here to
1649       % specialized functions.
1650       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1651       { \__zrefclever_sort_page:nn {##1} {##2} }
1652       { \__zrefclever_sort_default:nn {##1} {##2} }
1653     }
1654   }
1655 }
1656 }

```

(End definition for __zrefclever_sort_labels:.)

__zrefclever_label_type_put_new_right:n

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

```

\__zrefclever_label_type_put_new_right:n {<label>}

1657 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
1658 {
1659   \__zrefclever_def_extract:Nnnn
1660   \l__zrefclever_label_type_a_tl {#1} {zc@type} { \c_empty_tl }
1661   \seq_if_in:NVF \l__zrefclever_label_types_seq
1662   \l__zrefclever_label_type_a_tl
1663   {
1664     \seq_put_right:NV \l__zrefclever_label_types_seq
1665     \l__zrefclever_label_type_a_tl
1666   }
1667 }

```

(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:.


```

1668 \__zrefclever_sort_default:nn {\label a}} {\label b}}
1669 {
1670   \__zrefclever_def_extract:Nnnn
1671   \l__zrefclever_label_type_a_tl {#1} {zc@type} {\c_empty_tl}
1672   \__zrefclever_def_extract:Nnnn
1673   \l__zrefclever_label_type_b_tl {#2} {zc@type} {\c_empty_tl}
1674
1675   \bool_if:nTF
1676   {
1677     % The second label has a type, but the first doesn't, leave the
1678     % undefined first (to be more visible).
1679     \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1680     ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1681   }
1682   { \sort_return_same: }
1683   {
1684     \bool_if:nTF
1685     {
1686       % The first label has a type, but the second doesn't, bring the
1687       % second forward.
1688       ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1689       \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1690     }
1691     { \sort_return_swapped: }
1692     {
1693       \bool_if:nTF
1694       {
1695         % The interesting case: both labels have a type...
1696         ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1697         ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1698       }
1699       {
1700         \tl_if_eq:NNTF
1701         \l__zrefclever_label_type_a_tl
1702         \l__zrefclever_label_type_b_tl
1703         % ...and it's the same type.
1704         { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
1705         % ...and they are different types.
1706         { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
1707       }
1708     }
1709     % Neither label has a type. We can't do much of meaningful
1710     % here, but if it's the same counter, compare it.
1711     \exp_args:Nxx \tl_if_eq:nnTF
1712     { \__zrefclever_extract_unexp:nnn {#1} {zc@counter} { } }
1713     { \__zrefclever_extract_unexp:nnn {#2} {zc@counter} { } }
1714     {
1715       \int_compare:nNnTF
1716       { \__zrefclever_extract:nnn {#1} {zc@cntval} { -1 } }
1717       >
1718       { \__zrefclever_extract:nnn {#2} {zc@cntval} { -1 } }
1719       { \sort_return_swapped: }

```

```

1720         { \sort_return_same: }
1721     }
1722     { \sort_return_same: }
1723 }
1724 }
1725 }
1726 }

```

(End definition for _zrefclever_sort_default:nn.)

```

\_zrefclever_sort_default_same_type:nn      \_zrefclever_sort_default_same_type:nn {\label a}\{\label b}\}
1727 \cs_new_protected:Npn \_zrefclever_sort_default_same_type:nn #1#2
1728 {
1729   \_zrefclever_def_extract:Nnnn \l__zrefclever_label_enclval_a_tl
1730   {#1} { zc@enclval } { \c_empty_tl }
1731   \tl_reverse:N \l__zrefclever_label_enclval_a_tl
1732   \_zrefclever_def_extract:Nnnn \l__zrefclever_label_enclval_b_tl
1733   {#2} { zc@enclval } { \c_empty_tl }
1734   \tl_reverse:N \l__zrefclever_label_enclval_b_tl
1735   \_zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_a_tl
1736   {#1} { externaldocument } { \c_empty_tl }
1737   \_zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_b_tl
1738   {#2} { externaldocument } { \c_empty_tl }
1739
1740   \bool_set_false:N \l__zrefclever_sort_decided_bool
1741
1742   % First we check if there's any "external document" difference (coming
1743   % from 'zref-xr') and, if so, sort based on that.
1744   \tl_if_eq:NNF
1745     \l__zrefclever_label_extdoc_a_tl
1746     \l__zrefclever_label_extdoc_b_tl
1747   {
1748     \bool_if:nTF
1749     {
1750       \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
1751       ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
1752     }
1753     {
1754       \bool_set_true:N \l__zrefclever_sort_decided_bool
1755       \sort_return_same:
1756     }
1757     {
1758       \bool_if:nTF
1759       {
1760         ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
1761         \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
1762       }
1763       {
1764         \bool_set_true:N \l__zrefclever_sort_decided_bool
1765         \sort_return_swapped:
1766       }
1767       {
1768         \bool_set_true:N \l__zrefclever_sort_decided_bool
1769         % Two different "external documents": last resort, sort by the

```

```

1770         % document name itself.
1771         \str_compare:eNeTF
1772         { \l__zrefclever_label_extdoc_b_tl } <
1773         { \l__zrefclever_label_extdoc_a_tl }
1774         { \sort_return_swapped: }
1775         { \sort_return_same: }
1776     }
1777 }
1778 }
1779
1780 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1781 {
1782     \bool_if:nTF
1783     {
1784         % Both are empty: neither label has any (further) "enclosing
1785         % counters" (left).
1786         \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
1787         \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
1788     }
1789     {
1790         \bool_set_true:N \l__zrefclever_sort_decided_bool
1791         \int_compare:nNnTF
1792         { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
1793         >
1794         { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
1795         { \sort_return_swapped: }
1796         { \sort_return_same: }
1797     }
1798     {
1799         \bool_if:nTF
1800         {
1801             % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
1802             \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
1803         }
1804         {
1805             \bool_set_true:N \l__zrefclever_sort_decided_bool
1806             \int_compare:nNnTF
1807             { \__zrefclever_extract:nnn {#1} { zc@cntval } { } }
1808             >
1809             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1810             { \sort_return_swapped: }
1811             { \sort_return_same: }
1812         }
1813     }
1814     \bool_if:nTF
1815     {
1816         % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
1817         \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
1818     }
1819     {
1820         \bool_set_true:N \l__zrefclever_sort_decided_bool
1821         \int_compare:nNnTF
1822         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1823         <

```

```

1824         { \_zrefclever_extract:nnn {#2} { zc@cntval } { } }
1825         { \sort_return_same:    }
1826         { \sort_return_swapped: }
1827     }
1828     {
1829         % Neither is empty: we can compare the values of the
1830         % current enclosing counter in the loop, if they are
1831         % equal, we are still in the loop, if they are not, a
1832         % sorting decision can be made directly.
1833         \int_compare:nNnTF
1834         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1835         =
1836         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1837         {
1838             \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1839             { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1840             \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1841             { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1842         }
1843         {
1844             \bool_set_true:N \l__zrefclever_sort_decided_bool
1845             \int_compare:nNnTF
1846             { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1847             >
1848             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1849             { \sort_return_swapped: }
1850             { \sort_return_same:    }
1851         }
1852     }
1853 }
1854 }
1855 }
1856 }

```

(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)}
1857 \cs_new_protected:Npn \_zrefclever_sort_default_different_types:nn #1#2
1858 {

```

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

```

1859     \int_zero:N \l__zrefclever_sort_prior_a_int
1860     \int_zero:N \l__zrefclever_sort_prior_b_int
1861     \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1862     {
1863         \tl_if_eq:nnTF {##2} {{othertypes}}
1864         {
1865             \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
1866             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1867             \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
1868             { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1869         }

```

```

1870     {
1871       \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1872       { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1873       {
1874         \tl_if_eq:NnTF \l__zrefclever_label_type_b_tl {##2}
1875         { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1876       }
1877     }
1878   }

```

Then do the actual sorting.

```

1879   \bool_if:nTF
1880   {
1881     \int_compare_p:nNn
1882     { \l__zrefclever_sort_prior_a_int } <
1883     { \l__zrefclever_sort_prior_b_int }
1884   }
1885   { \sort_return_same: }
1886   {
1887     \bool_if:nTF
1888     {
1889       \int_compare_p:nNn
1890       { \l__zrefclever_sort_prior_a_int } >
1891       { \l__zrefclever_sort_prior_b_int }
1892     }
1893     { \sort_return_swapped: }
1894     {
1895       % Sort priorities are equal: the type that occurs first in
1896       % ‘labels’, as given by the user, is kept (or brought) forward.
1897       \seq_map_inline:Nn \l__zrefclever_label_types_seq
1898       {
1899         \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1900         { \seq_map_break:n { \sort_return_same: } }
1901         {
1902           \tl_if_eq:NnTF \l__zrefclever_label_type_b_tl {##1}
1903           { \seq_map_break:n { \sort_return_swapped: } }
1904         }
1905       }
1906     }
1907   }
1908 }

```

(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)}

1909 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
1910 {
1911   \int_compare:nNnTF

```

```

1912     { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
1913     >
1914     { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
1915     { \sort_return_swapped: }
1916     { \sort_return_same: }
1917 }

```

(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
1918 \seq_new:N \l__zrefclever_typeset_labels_seq
1919 \bool_new:N \l__zrefclever_typeset_last_bool
1920 \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
1921 \int_new:N \l__zrefclever_type_count_int
1922 \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
1923 \tl_new:N \l__zrefclever_label_a_tl
1924 \tl_new:N \l__zrefclever_label_b_tl
1925 \tl_new:N \l__zrefclever_typeset_queue_prev_tl
1926 \tl_new:N \l__zrefclever_typeset_queue_curr_tl
1927 \tl_new:N \l__zrefclever_type_first_label_tl
```

```
1928 \tl_new:N \l__zrefclever_type_first_label_type_tl
```

(End definition for \l__zrefclever_label_a_tl and others.)

\l__zrefclever_type_name_tl Auxiliary variables for __zrefclever_typeset_refs: type name handling.

```
1929 \tl_new:N \l__zrefclever_type_name_tl
1930 \bool_new:N \l__zrefclever_name_in_link_bool
1931 \tl_new:N \l__zrefclever_name_format_tl
1932 \tl_new:N \l__zrefclever_name_format_fallback_tl
1933 \tl_new:N \l__zrefclever_type_name_gender_tl
```

(End definition for \l__zrefclever_type_name_tl and others.)

Auxiliary variables for __zrefclever_typeset_refs: range handling.

```
1934 \int_new:N \l__zrefclever_range_count_int
1935 \int_new:N \l__zrefclever_range_same_count_int
1936 \tl_new:N \l__zrefclever_range_beg_label_tl
1937 \bool_new:N \l__zrefclever_next_maybe_range_bool
1938 \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 Auxiliary variables for __zrefclever_typeset_refs: separators, refpre/pos and font options.

```
1939 \tl_new:N \l__zrefclever_tpairsep_tl
1940 \tl_new:N \l__zrefclever_tlistsep_tl
1941 \tl_new:N \l__zrefclever_tlastsep_tl
1942 \tl_new:N \l__zrefclever_namesep_tl
1943 \tl_new:N \l__zrefclever_pairsep_tl
1944 \tl_new:N \l__zrefclever_listsep_tl
1945 \tl_new:N \l__zrefclever_lastsep_tl
1946 \tl_new:N \l__zrefclever_rangesep_tl
1947 \tl_new:N \l__zrefclever_refpre_out_tl
1948 \tl_new:N \l__zrefclever_refpos_out_tl
1949 \tl_new:N \l__zrefclever_refpre_in_tl
1950 \tl_new:N \l__zrefclever_refpos_in_tl
1951 \tl_new:N \l__zrefclever_namefont_tl
1952 \tl_new:N \l__zrefclever_reffont_out_tl
1953 \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 \zcref.

```
1954 \cs_new_protected:Npn \__zrefclever_typeset_refs:
1955 {
1956   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
1957   \l__zrefclever_zcref_labels_seq
1958   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1959   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1960   \tl_clear:N \l__zrefclever_type_first_label_tl
1961   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1962   \tl_clear:N \l__zrefclever_range_beg_label_tl
```



```

1963 \int_zero:N \l__zrefclever_label_count_int
1964 \int_zero:N \l__zrefclever_type_count_int
1965 \int_zero:N \l__zrefclever_range_count_int
1966 \int_zero:N \l__zrefclever_range_same_count_int
1967
1968 % Get type block options (not type-specific).
1969 \__zrefclever_get_ref_string:nN { tpairsep }
1970 \l__zrefclever_tpairsep_tl
1971 \__zrefclever_get_ref_string:nN { tlistsep }
1972 \l__zrefclever_tlistsep_tl
1973 \__zrefclever_get_ref_string:nN { tlastsep }
1974 \l__zrefclever_tlastsep_tl
1975
1976 % Process label stack.
1977 \bool_set_false:N \l__zrefclever_typeset_last_bool
1978 \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1979 {
1980   \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
1981   \l__zrefclever_label_a_tl
1982   \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
1983   {
1984     \tl_clear:N \l__zrefclever_label_b_tl
1985     \bool_set_true:N \l__zrefclever_typeset_last_bool
1986   }
1987   {
1988     \seq_get_left:NN \l__zrefclever_typeset_labels_seq
1989     \l__zrefclever_label_b_tl
1990   }
1991
1992   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1993   {
1994     \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1995     \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1996   }
1997   {
1998     \__zrefclever_def_extract:Nvnn \l__zrefclever_label_type_a_tl
1999     \l__zrefclever_label_a_tl { zc@type } { \c_empty_tl }
2000     \__zrefclever_def_extract:Nvnn \l__zrefclever_label_type_b_tl
2001     \l__zrefclever_label_b_tl { zc@type } { \c_empty_tl }
2002   }
2003
2004   % First, we establish whether the "current label" (i.e. 'a') is the
2005   % last one of its type. This can happen because the "next label"
2006   % (i.e. 'b') is of a different type (or different definition status),
2007   % or because we are at the end of the list.
2008   \bool_if:NTF \l__zrefclever_typeset_last_bool
2009   { \bool_set_true:N \l__zrefclever_last_of_type_bool }
2010   {
2011     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
2012     {
2013       \zref@ifrefundefined { \l__zrefclever_label_b_tl }
2014       { \bool_set_false:N \l__zrefclever_last_of_type_bool }
2015       { \bool_set_true:N \l__zrefclever_last_of_type_bool }
2016     }
2017   }

```

```

2017 {
2018   \zref@ifrefundefined { \l__zrefclever_label_b_tl }
2019   { \bool_set_true:N \l__zrefclever_last_of_type_bool }
2020   {
2021     % Neither is undefined, we must check the types.
2022     \bool_if:nTF
2023     {
2024       % Both empty: same "type".
2025       \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
2026       \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
2027     }
2028     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
2029     {
2030       \bool_if:nTF
2031       {
2032         % Neither empty: compare types.
2033         ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl
2034         &&
2035         ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
2036       }
2037       {
2038         \tl_if_eq:NNTF
2039         \l__zrefclever_label_type_a_tl
2040         \l__zrefclever_label_type_b_tl
2041         {
2042           \bool_set_false:N
2043           \l__zrefclever_last_of_type_bool
2044         }
2045         {
2046           \bool_set_true:N
2047           \l__zrefclever_last_of_type_bool
2048         }
2049       }
2050       % One empty, the other not: different "types".
2051       {
2052         \bool_set_true:N
2053         \l__zrefclever_last_of_type_bool
2054       }
2055     }
2056   }
2057 }
2058
2059 % Handle warnings in case of reference or type undefined.
2060 \zref@refused { \l__zrefclever_label_a_tl }
2061 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
2062 {}
2063 {
2064   \tl_if_empty:NT \l__zrefclever_label_type_a_tl
2065   {
2066     \msg_warning:nxx { zref-clever } { missing-type }
2067     { \l__zrefclever_label_a_tl }
2068   }
2069 }
2070

```

```

2071
2072 % Get type-specific separators, refpre/pos and font options, once per
2073 % type.
2074 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
2075 {
2076   \__zrefclever_get_ref_string:nN { namesep      }
2077   \l__zrefclever_namesep_tl
2078   \__zrefclever_get_ref_string:nN { rangesep     }
2079   \l__zrefclever_rangesep_tl
2080   \__zrefclever_get_ref_string:nN { pairsep      }
2081   \l__zrefclever_pairsep_tl
2082   \__zrefclever_get_ref_string:nN { listsep      }
2083   \l__zrefclever_listsep_tl
2084   \__zrefclever_get_ref_string:nN { lastsep      }
2085   \l__zrefclever_lastsep_tl
2086   \__zrefclever_get_ref_string:nN { refpre       }
2087   \l__zrefclever_refpre_out_tl
2088   \__zrefclever_get_ref_string:nN { refpos       }
2089   \l__zrefclever_refpos_out_tl
2090   \__zrefclever_get_ref_string:nN { refpre-in    }
2091   \l__zrefclever_refpre_in_tl
2092   \__zrefclever_get_ref_string:nN { refpos-in    }
2093   \l__zrefclever_refpos_in_tl
2094   \__zrefclever_get_ref_font:nN   { namefont     }
2095   \l__zrefclever_namefont_tl
2096   \__zrefclever_get_ref_font:nN   { reffont      }
2097   \l__zrefclever_reffont_out_tl
2098   \__zrefclever_get_ref_font:nN   { reffont-in   }
2099   \l__zrefclever_reffont_in_tl
2100 }
2101
2102 % Here we send this to a couple of auxiliary functions.
2103 \bool_if:NTF \l__zrefclever_last_of_type_bool
2104 % There exists no next label of the same type as the current.
2105 { \__zrefclever_typeset_refs_last_of_type: }
2106 % There exists a next label of the same type as the current.
2107 { \__zrefclever_typeset_refs_not_last_of_type: }
2108 }
2109 }

```

(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.
2110 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
2111 {

```

```

2112 % Process the current label to the current queue.
2113 \int_case:nnF { \l__zrefclever_label_count_int }
2114 {
2115   % It is the last label of its type, but also the first one, and that's
2116   % what matters here: just store it.
2117   { 0 }
2118   {
2119     \tl_set:NV \l__zrefclever_type_first_label_tl
2120     \l__zrefclever_label_a_tl
2121     \tl_set:NV \l__zrefclever_type_first_label_type_tl
2122     \l__zrefclever_label_type_a_tl
2123   }
2124
2125   % The last is the second: we have a pair (if not repeated).
2126   { 1 }
2127   {
2128     \int_compare:nNnF { \l__zrefclever_range_same_count_int } = { 1 }
2129     {
2130       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2131       {
2132         \exp_not:V \l__zrefclever_pairsep_tl
2133         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2134       }
2135     }
2136   }
2137 }
2138 % Last is third or more of its type: without repetition, we'd have the
2139 % last element on a list, but control for possible repetition.
2140 {
2141   \int_case:nnF { \l__zrefclever_range_count_int }
2142   {
2143     % There was no range going on.
2144     { 0 }
2145     {
2146       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2147       {
2148         \exp_not:V \l__zrefclever_lastsep_tl
2149         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2150       }
2151     }
2152     % Last in the range is also the second in it.
2153     { 1 }
2154     {
2155       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2156       {
2157         % We know 'range_beg_label' is not empty, since this is the
2158         % second element in the range, but the third or more in the
2159         % type list.
2160         \exp_not:V \l__zrefclever_listsep_tl
2161         \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
2162         \int_compare:nNnF
2163         { \l__zrefclever_range_same_count_int } = { 1 }
2164         {
2165           \exp_not:V \l__zrefclever_lastsep_tl

```

```

2166         \l__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2167     }
2168 }
2169 }
2170 }
2171 % Last in the range is third or more in it.
2172 {
2173   \int_case:nnF
2174   {
2175     \l__zrefclever_range_count_int -
2176     \l__zrefclever_range_same_count_int
2177   }
2178   {
2179     % Repetition, not a range.
2180     { 0 }
2181     {
2182       % If 'range_beg_label' is empty, it means it was also the
2183       % first of the type, and hence was already handled.
2184       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2185       {
2186         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2187         {
2188           \exp_not:V \l__zrefclever_lastsep_tl
2189           \l__zrefclever_get_ref:V
2190           \l__zrefclever_range_beg_label_tl
2191         }
2192       }
2193     }
2194     % A 'range', but with no skipped value, treat as list.
2195     { 1 }
2196     {
2197       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2198       {
2199         % Ditto.
2200         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2201         {
2202           \exp_not:V \l__zrefclever_listsep_tl
2203           \l__zrefclever_get_ref:V
2204           \l__zrefclever_range_beg_label_tl
2205         }
2206         \exp_not:V \l__zrefclever_lastsep_tl
2207         \l__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2208       }
2209     }
2210   }
2211   {
2212     % An actual range.
2213     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2214     {
2215       % Ditto.
2216       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2217       {
2218         \exp_not:V \l__zrefclever_lastsep_tl
2219         \l__zrefclever_get_ref:V

```

```

2220         \l__zrefclever_range_beg_label_tl
2221     }
2222     \exp_not:V \l__zrefclever_rangesep_tl
2223     \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2224 }
2225 }
2226 }
2227 }
2228
2229 % Handle "range" option. The idea is simple: if the queue is not empty,
2230 % we replace it with the end of the range (or pair). We can still
2231 % retrieve the end of the range from 'label_a' since we know to be
2232 % processing the last label of its type at this point.
2233 \bool_if:NT \l__zrefclever_typeset_range_bool
2234 {
2235     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2236     {
2237         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2238         { }
2239         {
2240             \msg_warning:nxx { zref-clever } { single-element-range }
2241             { \l__zrefclever_type_first_label_type_tl }
2242         }
2243     }
2244     {
2245         \bool_set_false:N \l__zrefclever_next_maybe_range_bool
2246         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2247         { }
2248         {
2249             \__zrefclever_labels_in_sequence:nn
2250             { \l__zrefclever_type_first_label_tl }
2251             { \l__zrefclever_label_a_tl }
2252         }
2253         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
2254         {
2255             \bool_if:NTF \l__zrefclever_next_maybe_range_bool
2256             { \exp_not:V \l__zrefclever_pairsep_tl }
2257             { \exp_not:V \l__zrefclever_rangesep_tl }
2258             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2259         }
2260     }
2261 }
2262
2263 % Now that the type block is finished, we can add the name and the first
2264 % ref to the queue. Also, if "typeset" option is not "both", handle it
2265 % here as well.
2266 \__zrefclever_type_name_setup:
2267 \bool_if:NTF
2268 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
2269 {
2270     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
2271     { \__zrefclever_get_ref_first: }
2272 }
2273 {

```

```

2274 \bool_if:nTF
2275 { \l__zrefclever_typeset_ref_bool }
2276 {
2277   \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
2278   { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
2279 }
2280 {
2281   \bool_if:nTF
2282   { \l__zrefclever_typeset_name_bool }
2283   {
2284     \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
2285     {
2286       \bool_if:NTF \l__zrefclever_name_in_link_bool
2287       {
2288         \exp_not:N \group_begin:
2289         \exp_not:V \l__zrefclever_namefont_tl
2290         % It's two '@s', but escaped for DocStrip.
2291         \exp_not:N \hyper@@link
2292         {
2293           \__zrefclever_extract_url_unexp:V
2294           \l__zrefclever_type_first_label_tl
2295         }
2296         {
2297           \__zrefclever_extract_unexp:Vnn
2298           \l__zrefclever_type_first_label_tl
2299           { anchor } { }
2300         }
2301         { \exp_not:V \l__zrefclever_type_name_tl }
2302         \exp_not:N \group_end:
2303       }
2304       {
2305         \exp_not:N \group_begin:
2306         \exp_not:V \l__zrefclever_namefont_tl
2307         \exp_not:V \l__zrefclever_type_name_tl
2308         \exp_not:N \group_end:
2309       }
2310     }
2311   }
2312   {
2313     % Logically, this case would correspond to "typeset=none", but
2314     % it should not occur, given that the options are set up to
2315     % typeset either "ref" or "name". Still, leave here a
2316     % sensible fallback, equal to the behavior of "both".
2317     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
2318     { \__zrefclever_get_ref_first: }
2319   }
2320 }
2321 }
2322
2323 % Typeset the previous type, if there is one.
2324 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
2325 {
2326   \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
2327   { \l__zrefclever_tlistsep_tl }

```

```

2328     \l__zrefclever_typeset_queue_prev_tl
2329   }
2330
2331   % Wrap up loop, or prepare for next iteration.
2332   \bool_if:NTF \l__zrefclever_typeset_last_bool
2333   {
2334     % We are finishing, typeset the current queue.
2335     \int_case:nnF { \l__zrefclever_type_count_int }
2336     {
2337       % Single type.
2338       { 0 }
2339       { \l__zrefclever_typeset_queue_curr_tl }
2340       % Pair of types.
2341       { 1 }
2342       {
2343         \l__zrefclever_tpairsep_tl
2344         \l__zrefclever_typeset_queue_curr_tl
2345       }
2346     }
2347     {
2348       % Last in list of types.
2349       \l__zrefclever_tlastsep_tl
2350       \l__zrefclever_typeset_queue_curr_tl
2351     }
2352     % And nudge in case of multitype reference.
2353     \bool_lazy_all:nT
2354     {
2355       { \l__zrefclever_nudge_enabled_bool }
2356       { \l__zrefclever_nudge_multitype_bool }
2357       { \int_compare_p:nNn { \l__zrefclever_type_count_int } > { 1 } }
2358     }
2359     { \msg_warning:nn { zref-clever } { nudge-multitype } }
2360   }
2361   {
2362     % There are further labels, set variables for next iteration.
2363     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
2364       \l__zrefclever_typeset_queue_curr_tl
2365     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
2366     \tl_clear:N \l__zrefclever_type_first_label_tl
2367     \tl_clear:N \l__zrefclever_type_first_label_type_tl
2368     \tl_clear:N \l__zrefclever_range_beg_label_tl
2369     \int_zero:N \l__zrefclever_label_count_int
2370     \int_incr:N \l__zrefclever_type_count_int
2371     \int_zero:N \l__zrefclever_range_count_int
2372     \int_zero:N \l__zrefclever_range_same_count_int
2373   }
2374 }

```

(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.

```

2375 \cs_new_protected:Npn \_zrefclever_typeset_refs_not_last_of_type:
2376 {
2377   % Signal if next label may form a range with the current one (only

```



```

2378 % considered if compression is enabled in the first place).
2379 \bool_set_false:N \l__zrefclever_next_maybe_range_bool
2380 \bool_set_false:N \l__zrefclever_next_is_same_bool
2381 \bool_if:NT \l__zrefclever_typeset_compress_bool
2382 {
2383   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
2384   { }
2385   {
2386     \__zrefclever_labels_in_sequence:nn
2387     { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
2388   }
2389 }
2390
2391 % Process the current label to the current queue.
2392 \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
2393 {
2394   % Current label is the first of its type (also not the last, but it
2395   % doesn't matter here): just store the label.
2396   \tl_set:NV \l__zrefclever_type_first_label_tl
2397   \l__zrefclever_label_a_tl
2398   \tl_set:NV \l__zrefclever_type_first_label_type_tl
2399   \l__zrefclever_label_type_a_tl
2400
2401   % If the next label may be part of a range, we set 'range_beg_label'
2402   % to "empty" (we deal with it as the "first", and must do it there, to
2403   % handle hyperlinking), but also step the range counters.
2404   \bool_if:NT \l__zrefclever_next_maybe_range_bool
2405   {
2406     \tl_clear:N \l__zrefclever_range_beg_label_tl
2407     \int_incr:N \l__zrefclever_range_count_int
2408     \bool_if:NT \l__zrefclever_next_is_same_bool
2409     { \int_incr:N \l__zrefclever_range_same_count_int }
2410   }
2411 }
2412 {
2413   % Current label is neither the first (nor the last) of its type.
2414   \bool_if:NTF \l__zrefclever_next_maybe_range_bool
2415   {
2416     % Starting, or continuing a range.
2417     \int_compare:nNnTF
2418     { \l__zrefclever_range_count_int } = { 0 }
2419     {
2420       % There was no range going, we are starting one.
2421       \tl_set:NV \l__zrefclever_range_beg_label_tl
2422       \l__zrefclever_label_a_tl
2423       \int_incr:N \l__zrefclever_range_count_int
2424       \bool_if:NT \l__zrefclever_next_is_same_bool
2425       { \int_incr:N \l__zrefclever_range_same_count_int }
2426     }
2427     {
2428       % Second or more in the range, but not the last.
2429       \int_incr:N \l__zrefclever_range_count_int
2430       \bool_if:NT \l__zrefclever_next_is_same_bool
2431       { \int_incr:N \l__zrefclever_range_same_count_int }

```

```

2432     }
2433   }
2434   {
2435     % Next element is not in sequence: there was no range, or we are
2436     % closing one.
2437     \int_case:nnF { \l__zrefclever_range_count_int }
2438     {
2439       % There was no range going on.
2440       { 0 }
2441       {
2442         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2443         {
2444           \exp_not:V \l__zrefclever_listsep_tl
2445           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2446         }
2447       }
2448       % Last is second in the range: if 'range_same_count' is also
2449       % '1', it's a repetition (drop it), otherwise, it's a "pair
2450       % within a list", treat as list.
2451       { 1 }
2452       {
2453         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2454         {
2455           \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2456           {
2457             \exp_not:V \l__zrefclever_listsep_tl
2458             \__zrefclever_get_ref:V
2459             \l__zrefclever_range_beg_label_tl
2460           }
2461           \int_compare:nNnF
2462           { \l__zrefclever_range_same_count_int } = { 1 }
2463           {
2464             \exp_not:V \l__zrefclever_listsep_tl
2465             \__zrefclever_get_ref:V
2466             \l__zrefclever_label_a_tl
2467           }
2468         }
2469       }
2470     }
2471   {
2472     % Last is third or more in the range: if 'range_count' and
2473     % 'range_same_count' are the same, its a repetition (drop it),
2474     % if they differ by '1', its a list, if they differ by more,
2475     % it is a real range.
2476     \int_case:nnF
2477     {
2478       \l__zrefclever_range_count_int -
2479       \l__zrefclever_range_same_count_int
2480     }
2481     {
2482       { 0 }
2483       {
2484         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2485         {

```

```

2486         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2487         {
2488             \exp_not:V \l__zrefclever_listsep_tl
2489             \__zrefclever_get_ref:V
2490             \l__zrefclever_range_beg_label_tl
2491         }
2492     }
2493 }
2494 { 1 }
2495 {
2496     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2497     {
2498         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2499         {
2500             \exp_not:V \l__zrefclever_listsep_tl
2501             \__zrefclever_get_ref:V
2502             \l__zrefclever_range_beg_label_tl
2503         }
2504         \exp_not:V \l__zrefclever_listsep_tl
2505         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2506     }
2507 }
2508 }
2509 {
2510     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2511     {
2512         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2513         {
2514             \exp_not:V \l__zrefclever_listsep_tl
2515             \__zrefclever_get_ref:V
2516             \l__zrefclever_range_beg_label_tl
2517         }
2518         \exp_not:V \l__zrefclever_rangesep_tl
2519         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2520     }
2521 }
2522 }
2523 % Reset counters.
2524 \int_zero:N \l__zrefclever_range_count_int
2525 \int_zero:N \l__zrefclever_range_same_count_int
2526 }
2527 }
2528 % Step label counter for next iteration.
2529 \int_incr:N \l__zrefclever_label_count_int
2530 }

```

(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 nicl 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.

```

2531 \cs_new_protected:Npn \__zrefclever_ref_default:
2532   { \zref@default }
2533 \cs_new_protected:Npn \__zrefclever_name_default:
2534   { \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>}

2535 \cs_new:Npn \__zrefclever_get_ref:n #1
2536   {
2537     \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
2538     {
2539       \bool_if:nTF
2540       {
2541         \l__zrefclever_use_hyperref_bool &&
2542         ! \l__zrefclever_link_star_bool
2543       }
2544       {
2545         \exp_not:N \group_begin:
2546         \exp_not:V \l__zrefclever_reffont_out_tl
2547         \exp_not:V \l__zrefclever_refpre_out_tl
2548         \exp_not:N \group_begin:
2549         \exp_not:V \l__zrefclever_reffont_in_tl
2550         % It's two '@s', but escaped for DocStrip.
2551         \exp_not:N \hyper@@link
2552         { \__zrefclever_extract_url_unexp:n {#1} }
2553         { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }

```

```

2554         {
2555             \exp_not:V \l__zrefclever_refpre_in_tl
2556             \__zrefclever_extract_unexp:nvn {#1}
2557             { \l__zrefclever_ref_property_tl } { }
2558             \exp_not:V \l__zrefclever_refpos_in_tl
2559         }
2560         \exp_not:N \group_end:
2561         \exp_not:V \l__zrefclever_refpos_out_tl
2562         \exp_not:N \group_end:
2563     }
2564     {
2565         \exp_not:N \group_begin:
2566         \exp_not:V \l__zrefclever_reffont_out_tl
2567         \exp_not:V \l__zrefclever_refpre_out_tl
2568         \exp_not:N \group_begin:
2569         \exp_not:V \l__zrefclever_reffont_in_tl
2570         \exp_not:V \l__zrefclever_refpre_in_tl
2571         \__zrefclever_extract_unexp:nvn {#1}
2572         { \l__zrefclever_ref_property_tl } { }
2573         \exp_not:V \l__zrefclever_refpos_in_tl
2574         \exp_not:N \group_end:
2575         \exp_not:V \l__zrefclever_refpos_out_tl
2576         \exp_not:N \group_end:
2577     }
2578 }
2579 { \__zrefclever_ref_default: }
2580 }
2581 \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.

```

2582 \cs_new:Npn \__zrefclever_get_ref_first:
2583 {
2584     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2585     { \__zrefclever_ref_default: }
2586     {
2587         \bool_if:NTF \l__zrefclever_name_in_link_bool
2588         {
2589             \zref@ifrefcontainsprop
2590             { \l__zrefclever_type_first_label_tl }
2591             { \l__zrefclever_ref_property_tl }
2592             {
2593                 % It's two '@s', but escaped for DocStrip.
2594                 \exp_not:N \hyper@@link
2595                 {
2596                     \__zrefclever_extract_url_unexp:V

```

```

2597         \l__zrefclever_type_first_label_tl
2598     }
2599     {
2600         \__zrefclever_extract_unexp:Vnn
2601         \l__zrefclever_type_first_label_tl { anchor } { }
2602     }
2603     {
2604         \exp_not:N \group_begin:
2605         \exp_not:V \l__zrefclever_namefont_tl
2606         \exp_not:V \l__zrefclever_type_name_tl
2607         \exp_not:N \group_end:
2608         \exp_not:V \l__zrefclever_namesep_tl
2609         \exp_not:N \group_begin:
2610         \exp_not:V \l__zrefclever_reffont_out_tl
2611         \exp_not:V \l__zrefclever_refpre_out_tl
2612         \exp_not:N \group_begin:
2613         \exp_not:V \l__zrefclever_reffont_in_tl
2614         \exp_not:V \l__zrefclever_refpre_in_tl
2615         \__zrefclever_extract_unexp:Vnn
2616         \l__zrefclever_type_first_label_tl
2617         { \l__zrefclever_ref_property_tl } { }
2618         \exp_not:V \l__zrefclever_refpos_in_tl
2619         \exp_not:N \group_end:
2620         % hyperlink makes it's own group, we'd like to close the
2621         % 'refpre-out' group after 'refpos-out', but... we close
2622         % it here, and give the trailing 'refpos-out' its own
2623         % group. This will result that formatting given to
2624         % 'refpre-out' will not reach 'refpos-out', but I see no
2625         % alternative, and this has to be handled specially.
2626         \exp_not:N \group_end:
2627     }
2628     \exp_not:N \group_begin:
2629     % Ditto: special treatment.
2630     \exp_not:V \l__zrefclever_reffont_out_tl
2631     \exp_not:V \l__zrefclever_refpos_out_tl
2632     \exp_not:N \group_end:
2633 }
2634 {
2635     \exp_not:N \group_begin:
2636     \exp_not:V \l__zrefclever_namefont_tl
2637     \exp_not:V \l__zrefclever_type_name_tl
2638     \exp_not:N \group_end:
2639     \exp_not:V \l__zrefclever_namesep_tl
2640     \__zrefclever_ref_default:
2641 }
2642 }
2643 {
2644     \tl_if_empty:NTF \l__zrefclever_type_name_tl
2645     {
2646         \__zrefclever_name_default:
2647         \exp_not:V \l__zrefclever_namesep_tl
2648     }
2649     {
2650         \exp_not:N \group_begin:

```

```

2651         \exp_not:V \l__zrefclever_namefont_tl
2652         \exp_not:V \l__zrefclever_type_name_tl
2653         \exp_not:N \group_end:
2654         \exp_not:V \l__zrefclever_namesep_tl
2655     }
2656 \zref@ifrefcontainsprop
2657 { \l__zrefclever_type_first_label_tl }
2658 { \l__zrefclever_ref_property_tl }
2659 {
2660     \bool_if:nTF
2661     {
2662         \l__zrefclever_use_hyperref_bool &&
2663         ! \l__zrefclever_link_star_bool
2664     }
2665     {
2666         \exp_not:N \group_begin:
2667         \exp_not:V \l__zrefclever_reffont_out_tl
2668         \exp_not:V \l__zrefclever_refpre_out_tl
2669         \exp_not:N \group_begin:
2670         \exp_not:V \l__zrefclever_reffont_in_tl
2671         % It's two '@s', but escaped for DocStrip.
2672         \exp_not:N \hyper@@link
2673         {
2674             \__zrefclever_extract_url_unexp:V
2675             \l__zrefclever_type_first_label_tl
2676         }
2677         {
2678             \__zrefclever_extract_unexp:Vnn
2679             \l__zrefclever_type_first_label_tl { anchor } { }
2680         }
2681         {
2682             \exp_not:V \l__zrefclever_refpre_in_tl
2683             \__zrefclever_extract_unexp:Vnn
2684             \l__zrefclever_type_first_label_tl
2685             { \l__zrefclever_ref_property_tl } { }
2686             \exp_not:V \l__zrefclever_refpos_in_tl
2687         }
2688         \exp_not:N \group_end:
2689         \exp_not:V \l__zrefclever_refpos_out_tl
2690         \exp_not:N \group_end:
2691     }
2692     {
2693         \exp_not:N \group_begin:
2694         \exp_not:V \l__zrefclever_reffont_out_tl
2695         \exp_not:V \l__zrefclever_refpre_out_tl
2696         \exp_not:N \group_begin:
2697         \exp_not:V \l__zrefclever_reffont_in_tl
2698         \exp_not:V \l__zrefclever_refpre_in_tl
2699         \__zrefclever_extract_unexp:Vnn
2700         \l__zrefclever_type_first_label_tl
2701         { \l__zrefclever_ref_property_tl } { }
2702         \exp_not:V \l__zrefclever_refpos_in_tl
2703         \exp_not:N \group_end:
2704         \exp_not:V \l__zrefclever_refpos_out_tl

```

```

2705         \exp_not:N \group_end:
2706     }
2707 }
2708 { \__zrefclever_ref_default: }
2709 }
2710 }
2711 }

```

(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.

```

2712 \cs_new_protected:Npn \__zrefclever_type_name_setup:
2713 {
2714   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2715   { \tl_clear:N \l__zrefclever_type_name_tl }
2716   {
2717     \tl_if_empty:NTF \l__zrefclever_type_first_label_type_tl
2718     { \tl_clear:N \l__zrefclever_type_name_tl }
2719     {
2720       % Determine whether we should use capitalization, abbreviation,
2721       % and plural.
2722       \bool_lazy_or:nnTF
2723       { \l__zrefclever_capitalize_bool }
2724       {
2725         \l__zrefclever_capitalize_first_bool &&
2726         \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2727       }
2728       { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
2729       { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
2730       % If the queue is empty, we have a singular, otherwise, plural.
2731       \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2732       { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
2733       { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
2734       \bool_lazy_and:nnTF
2735       { \l__zrefclever_abbrev_bool }
2736       {
2737         ! \int_compare_p:nNn
2738         { \l__zrefclever_type_count_int } = { 0 } ||
2739         ! \l__zrefclever_noabbrev_first_bool
2740       }
2741       {
2742         \tl_set:NV \l__zrefclever_name_format_fallback_tl
2743         \l__zrefclever_name_format_tl

```



```

2744 \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
2745 }
2746 { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
2747
2748 % Handle number and gender nudges.
2749 \bool_if:NT \l__zrefclever_nudge_enabled_bool
2750 {
2751   \bool_if:NTF \l__zrefclever_nudge_singular_bool
2752   {
2753     \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
2754     {
2755       \msg_warning:nnx { zref-clever }
2756       { nudge-plural-when-sg }
2757       { \l__zrefclever_type_first_label_type_tl }
2758     }
2759   }
2760   {
2761     \bool_lazy_all:nT
2762     {
2763       { \l__zrefclever_nudge_comptosing_bool }
2764       { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
2765       {
2766         \int_compare_p:nNn
2767         { \l__zrefclever_label_count_int } > { 0 }
2768       }
2769     }
2770     {
2771       \msg_warning:nnx { zref-clever }
2772       { nudge-comptosing }
2773       { \l__zrefclever_type_first_label_type_tl }
2774     }
2775   }
2776   \bool_lazy_and:nnT
2777   { \l__zrefclever_nudge_gender_bool }
2778   { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
2779   {
2780     \__zrefclever_get_type_transl:xxnNF
2781     { \l__zrefclever_ref_language_tl }
2782     { \l__zrefclever_type_first_label_type_tl }
2783     { gender }
2784     \l__zrefclever_type_name_gender_tl
2785     { \tl_clear:N \l__zrefclever_type_name_gender_tl }
2786     \tl_if_eq:NNF
2787     \l__zrefclever_ref_gender_tl
2788     \l__zrefclever_type_name_gender_tl
2789     {
2790       \tl_if_empty:NTF \l__zrefclever_type_name_gender_tl
2791       {
2792         \msg_warning:nnxxx { zref-clever }
2793         { nudge-gender-not-declared-for-type }
2794         { \l__zrefclever_ref_gender_tl }
2795         { \l__zrefclever_type_first_label_type_tl }
2796         { \l__zrefclever_ref_language_tl }
2797       }

```

```

2798         {
2799             \msg_warning:nnxxxx { zref-clever }
2800             { nudge-gender-mismatch }
2801             { \l__zrefclever_type_first_label_type_tl }
2802             { \l__zrefclever_ref_gender_tl }
2803             { \l__zrefclever_type_name_gender_tl }
2804             { \l__zrefclever_ref_language_tl }
2805         }
2806     }
2807 }
2808 }
2809
2810 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
2811 {
2812     \prop_get:cVNF
2813     {
2814         l__zrefclever_type_
2815         \l__zrefclever_type_first_label_type_tl _options_prop
2816     }
2817     \l__zrefclever_name_format_tl
2818     \l__zrefclever_type_name_tl
2819     {
2820         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
2821         {
2822             \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
2823             \tl_put_left:NV \l__zrefclever_name_format_tl
2824             \l__zrefclever_ref_decl_case_tl
2825         }
2826         \__zrefclever_get_type_transl:xxxNF
2827         { \l__zrefclever_ref_language_tl }
2828         { \l__zrefclever_type_first_label_type_tl }
2829         { \l__zrefclever_name_format_tl }
2830         \l__zrefclever_type_name_tl
2831         {
2832             \tl_clear:N \l__zrefclever_type_name_tl
2833             \msg_warning:nnxx { zref-clever } { missing-name }
2834             { \l__zrefclever_name_format_tl }
2835             { \l__zrefclever_type_first_label_type_tl }
2836         }
2837     }
2838 }
2839 {
2840     \prop_get:cVNF
2841     {
2842         l__zrefclever_type_
2843         \l__zrefclever_type_first_label_type_tl _options_prop
2844     }
2845     \l__zrefclever_name_format_tl
2846     \l__zrefclever_type_name_tl
2847     {
2848         \prop_get:cVNF
2849         {
2850             l__zrefclever_type_
2851             \l__zrefclever_type_first_label_type_tl _options_prop

```

```

2852     }
2853     \l__zrefclever_name_format_fallback_tl
2854     \l__zrefclever_type_name_tl
2855     {
2856         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
2857         {
2858             \tl_put_left:Nn
2859             \l__zrefclever_name_format_tl { - }
2860             \tl_put_left:NV \l__zrefclever_name_format_tl
2861             \l__zrefclever_ref_decl_case_tl
2862             \tl_put_left:Nn
2863             \l__zrefclever_name_format_fallback_tl { - }
2864             \tl_put_left:NV
2865             \l__zrefclever_name_format_fallback_tl
2866             \l__zrefclever_ref_decl_case_tl
2867         }
2868         \__zrefclever_get_type_transl:xxxNF
2869         { \l__zrefclever_ref_language_tl }
2870         { \l__zrefclever_type_first_label_type_tl }
2871         { \l__zrefclever_name_format_tl }
2872         \l__zrefclever_type_name_tl
2873         {
2874             \__zrefclever_get_type_transl:xxxNF
2875             { \l__zrefclever_ref_language_tl }
2876             { \l__zrefclever_type_first_label_type_tl }
2877             { \l__zrefclever_name_format_fallback_tl }
2878             \l__zrefclever_type_name_tl
2879             {
2880                 \tl_clear:N \l__zrefclever_type_name_tl
2881                 \msg_warning:nxxx { zref-clever }
2882                 { missing-name }
2883                 { \l__zrefclever_name_format_tl }
2884                 { \l__zrefclever_type_first_label_type_tl }
2885             }
2886         }
2887     }
2888 }
2889 }
2890 }
2891 }
2892
2893 % Signal whether the type name is to be included in the hyperlink or not.
2894 \bool_lazy_any:nTF
2895 {
2896     { ! \l__zrefclever_use_hyperref_bool }
2897     { \l__zrefclever_link_star_bool }
2898     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
2899     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2900 }
2901 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2902 {
2903     \bool_lazy_any:nTF
2904     {
2905         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }

```

```

2906         {
2907             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2908             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2909         }
2910         {
2911             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2912             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2913             \l__zrefclever_typeset_last_bool &&
2914             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2915         }
2916     }
2917     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2918     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2919 }
2920 }

```

(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.

```

2921 \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
2922 {
2923     \zref@ifpropundefined { urluse }
2924     { \__zrefclever_extract_unexp:nnn {#1} { url } { \c_empty_tl } }
2925     {
2926         \zref@ifrefcontainsprop {#1} { urluse }
2927         { \__zrefclever_extract_unexp:nnn {#1} { urluse } { \c_empty_tl } }
2928         { \__zrefclever_extract_unexp:nnn {#1} { url } { \c_empty_tl } }
2929     }
2930 }
2931 \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>}

2932 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
2933 {
2934     \__zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_a_tl
2935     {#1} { externaldocument } { \c_empty_tl }
2936     \__zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_b_tl
2937     {#2} { externaldocument } { \c_empty_tl }
2938
2939     \tl_if_eq:NNT

```

```

2940 \l__zrefclever_label_extdoc_a_tl
2941 \l__zrefclever_label_extdoc_b_tl
2942 {
2943   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2944   {
2945     \exp_args:Nxx \tl_if_eq:nnT
2946     { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
2947     { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
2948     {
2949       \int_compare:nNnTF
2950       { \__zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
2951       =
2952       { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
2953       { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2954       {
2955         \int_compare:nNnTF
2956         { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
2957         =
2958         { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
2959         {
2960           \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2961           \bool_set_true:N \l__zrefclever_next_is_same_bool
2962         }
2963       }
2964     }
2965   }
2966   {
2967     \exp_args:Nxx \tl_if_eq:nnT
2968     { \__zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
2969     { \__zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
2970     {
2971       \exp_args:Nxx \tl_if_eq:nnT
2972       { \__zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
2973       { \__zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
2974       {
2975         \int_compare:nNnTF
2976         { \__zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
2977         =
2978         { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
2979         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2980         {
2981           \int_compare:nNnTF
2982           { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
2983           =
2984           { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
2985           {
2986             \bool_set_true:N
2987             \l__zrefclever_next_maybe_range_bool
2988             \exp_args:Nxx \tl_if_eq:nnT
2989             {
2990               \__zrefclever_extract_unexp:nvn {#1}
2991               { \l__zrefclever_ref_property_tl } { }
2992             }
2993             {

```

```

2994         \__zrefclever_extract_unexp:nvn {#2}
2995         { l__zrefclever_ref_property_tl } { }
2996     }
2997     {
2998         \bool_set_true:N
2999         \l__zrefclever_next_is_same_bool
3000     }
3001 }
3002 }
3003 }
3004 }
3005 }
3006 }
3007 }

```

(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>}
3008 \cs_new_protected:Npn \__zrefclever_get_ref_string:nN #1#2
3009 {
3010     % First attempt: general options.
3011     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
3012     {
3013         % If not found, try type specific options.
3014         \bool_lazy_all:nTF
3015         {
3016             { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
3017             {
3018                 \prop_if_exist_p:c
3019                 {
3020                     l__zrefclever_type_
3021                     \l__zrefclever_label_type_a_tl _options_prop
3022                 }
3023             }
3024             {
3025                 \prop_if_in_p:cn
3026                 {
3027                     l__zrefclever_type_
3028                     \l__zrefclever_label_type_a_tl _options_prop
3029                 }
3030                 {#1}

```

```

3031     }
3032   }
3033   {
3034     \prop_get:cnN
3035     {
3036       l__zrefclever_type_
3037       \l__zrefclever_label_type_a_tl _options_prop
3038     }
3039     {#1} #2
3040   }
3041   {
3042     % If not found, try type specific translations.
3043     \__zrefclever_get_type_transl:xnNF
3044     { \l__zrefclever_ref_language_tl }
3045     { \l__zrefclever_label_type_a_tl }
3046     {#1} #2
3047     {
3048       % If not found, try default translations.
3049       \__zrefclever_get_default_transl:xnNF
3050       { \l__zrefclever_ref_language_tl }
3051       {#1} #2
3052       {
3053         % If not found, try fallback.
3054         \__zrefclever_get_fallback_transl:nNF {#1} #2
3055         {
3056           \tl_clear:N #2
3057           \msg_warning:nnn { zref-clever }
3058             { missing-string } {#1}
3059         }
3060       }
3061     }
3062   }
3063 }
3064 }

```

(End definition for __zrefclever_get_ref_string:nN.)

```

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

```

```

3081         \prop_get:cnNF
3082         {
3083             l__zrefclever_type_
3084             \l__zrefclever_label_type_a_tl _options_prop
3085         }
3086         {#1} #2
3087         { \tl_clear:N #2 }
3088     }
3089     { \tl_clear:N #2 }
3090 }
3091 }

```

(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.

```

3092 \tl_new:N \l__zrefclever_footnote_type_tl
3093 \tl_set:Nn \l__zrefclever_footnote_type_tl { footnote }
3094 \AddToHook { env / minipage / begin }
3095 { \tl_set:Nn \l__zrefclever_footnote_type_tl { mpfootnote } }
3096 \AddToHook { cmd / @makeintext / before }
3097 {
3098     \__zrefclever_zcsetup:x
3099     { currentcounter = \l__zrefclever_footnote_type_tl }
3100 }

```

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.

```

3101 \AddToHook { cmd / appendix / before }
3102 {
3103   \__zrefclever_zcsetup:n
3104   {
3105     countertype =
3106     {
3107       chapter      = appendix ,
3108       section      = appendix ,
3109       subsection   = appendix ,
3110       subsubsection = appendix ,
3111     }
3112   }
3113 }

```

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.

```

3114 \AddToHook { begindocument }
3115 {
3116   \@ifpackageloaded { appendix }
3117   {
3118     \newcounter { zc@appendix }
3119     \newcounter { zc@save@appendix }

```

```

3120 \setcounter { zc@appendix } { 0 }
3121 \setcounter { zc@save@appendix } { 0 }
3122 \cs_if_exist:cTF { chapter }
3123 {
3124   \__zrefclever_zcsetup:n
3125   { counterresetby = { chapter = zc@appendix } }
3126 }
3127 {
3128   \cs_if_exist:cT { section }
3129   {
3130     \__zrefclever_zcsetup:n
3131     { counterresetby = { section = zc@appendix } }
3132   }
3133 }
3134 \AddToHook { env / appendices / begin }
3135 {
3136   \stepcounter { zc@save@appendix }
3137   \setcounter { zc@appendix } { \value { zc@save@appendix } }
3138   \__zrefclever_zcsetup:n
3139   {
3140     countertype =
3141     {
3142       chapter      = appendix ,
3143       section      = appendix ,
3144       subsection   = appendix ,
3145       subsubsection = appendix ,
3146     }
3147   }
3148 }
3149 \AddToHook { env / appendices / end }
3150 { \setcounter { zc@appendix } { 0 } }
3151 \AddToHook { cmd / appendix / before }
3152 {
3153   \stepcounter { zc@save@appendix }
3154   \setcounter { zc@appendix } { \value { zc@save@appendix } }
3155 }
3156 \AddToHook { env / subappendices / begin }
3157 {
3158   \__zrefclever_zcsetup:n
3159   {
3160     countertype =
3161     {
3162       section      = appendix ,
3163       subsection   = appendix ,
3164       subsubsection = appendix ,
3165     } ,
3166   }
3167 }
3168 \msg_info:nnn { zref-clever } { compat-package } { appendix }
3169 }
3170 {}
3171 }

```

9.4 amsmath package

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

```

3172 \AddToHook { begindocument }
3173 {
3174   \@ifpackageloaded { amsmath }
3175   {

```

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...).

```

3176     \cs_set_nopar:Npn \__zrefclever_ltxlabel:n #1
3177     {
3178       \__zrefclever_orig_ltxlabel:n {#1}
3179       \zref@wrapper@babel \zref@label {#1}
3180     }

```

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 `\zceref`. 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 `\ceref` to make references to those. CHECK Should I just make this no-op in case ‘`cleveref`’ is loaded?

```

3181     \IfFormatAtLeastTF { 2021-11-15 }
3182     {
3183       \@ifpackageloaded { hyperref }
3184       {
3185         \AddToHook { package / nameref / after }
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         \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3193         \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3194       }
3195     }
3196     {
3197       \@ifpackageloaded { hyperref }

```

```

3198 {
3199   \ifpackageloaded { nameref }
3200   {
3201     \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3202     \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3203   }
3204   {
3205     \AddToHook { package / after / nameref }
3206     {
3207       \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3208       \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3209     }
3210   }
3211 }
3212 {
3213   \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3214   \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3215 }
3216 }

```

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).

```

3217 \AddToHook { env / subequations / begin }
3218 {
3219   \__zrefclever_zcsetup:x
3220   {
3221     counterresetby =
3222     {
3223       parentequation =
3224       \__zrefclever_counter_reset_by:n { equation } ,
3225       equation = parentequation ,
3226     } ,
3227     currentcounter = parentequation ,
3228     countertype = { parentequation = equation } ,
3229   }
3230 }

```

`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 it 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.

```

3231 \clist_map_inline:nn
3232 {
3233     equation ,
3234     equation* ,
3235     align ,
3236     align* ,
3237     alignat ,
3238     alignat* ,
3239     flalign ,
3240     flalign* ,
3241     xalignat ,
3242     xalignat* ,
3243     gather ,
3244     gather* ,
3245     multiline ,
3246     multiline* ,
3247 }
3248 {
3249     \AddToHook { env / #1 / begin }
3250     { \__zrefclever_zcsetup:n { currentcounter = equation } }
3251 }

```

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

```

3252 \zcRefTypeSetup { equation } { reffont = \upshape }
3253 \msg_info:nnn { zref-clever } { compat-package } { amsmath }
3254 }
3255 {}
3256 }

```

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.

```

3257 \bool_new:N \l__zrefclever_mathtools_showonlyrefs_bool
3258 \AddToHook { begindocument }
3259 {
3260     \@ifpackageloaded { mathtools }
3261     {
3262         \MH_if_boolean:nT { show_only_refs }
3263         {
3264             \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
3265             \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
3266             {

```

```

3267         \@bsphack
3268         \seq_map_inline:Nn #1
3269         {
3270             \exp_args:Nx \tl_if_eq:nnTF
3271             { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
3272             { equation }
3273             {
3274                 \protected@write \@auxout { }
3275                 { \string \MT@newlabel {##1} }
3276             }
3277             {
3278                 \exp_args:Nx \tl_if_eq:nnT
3279                 { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
3280                 { parentequation }
3281                 {
3282                     \protected@write \@auxout { }
3283                     { \string \MT@newlabel {##1} }
3284                 }
3285             }
3286         }
3287     \@esphack
3288 }
3289 \msg_info:nnn { zref-clever } { compat-package } { mathtools }
3290 }
3291 {}
3292 }
3293 }

```

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”.

```

3294 \AddToHook { begindocument }
3295 {
3296     \@ifpackageloaded { breqn }
3297     {

```

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.

```

3298     \AddToHook { env / dgroup / begin }
3299     {
3300         \__zrefclever_zcsetup:x
3301         {
3302             counterresetby =
3303             {
3304                 parentequation =
3305                 \__zrefclever_counter_reset_by:n { equation } ,

```

```

3306         equation = parentequation ,
3307     } ,
3308     currentcounter = parentequation ,
3309     countertype = { parentequation = equation } ,
3310 }
3311 }
3312 \clist_map_inline:nn
3313 {
3314     dmath ,
3315     dseries ,
3316     darray ,
3317 }
3318 {
3319     \AddToHook { env / #1 / begin }
3320     { \__zrefclever_zcsetup:n { currentcounter = equation } }
3321 }
3322 }
3323 {}
3324 }

```

9.7 listings package

```

3325 \AddToHook { begindocument }
3326 {
3327     \ifpackageloaded { listings }
3328     {
3329         \__zrefclever_zcsetup:n
3330         {
3331             countertype =
3332             {
3333                 lstlisting = listing ,
3334                 lstnumber = line ,
3335             } ,
3336             counterresetby = { lstnumber = lstlisting } ,
3337         }
3338         \lst@AddToHook { Init }
3339         {

```

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

```

3340         \tl_if_empty:NF \lst@label
3341         { \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.

```

3342         \__zrefclever_zcsetup:n { currentcounter = lstnumber }
3343     }
3344     \msg_info:nnn { zref-clever } { compat-package } { listings }

```

```

3345     }
3346     {}
3347 }

```

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 $\langle \max\text{-depth} \rangle$. `\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.

```

3348 \AddToHook { begindocument }
3349 {
3350   \@ifpackageloaded { enumitem }
3351   {
3352     \int_set:Nn \l_tmpa_int { 5 }
3353     \bool_while_do:nn
3354     {
3355       \cs_if_exist_p:c
3356       { c@ enum \int_to_roman:n { \l_tmpa_int } }
3357     }
3358     {
3359       \__zrefclever_zcsetup:x
3360       {
3361         counterresetby =
3362         {
3363           enum \int_to_roman:n { \l_tmpa_int } =
3364           enum \int_to_roman:n { \l_tmpa_int - 1 }
3365         } ,
3366         countertype =
3367         { enum \int_to_roman:n { \l_tmpa_int } = item } ,
3368       }
3369       \int_incr:N \l_tmpa_int
3370     }
3371     \int_compare:nNnT { \l_tmpa_int } > { 5 }
3372     { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
3373   }
3374   {}
3375 }
3376 \</package>

```


10 Dictionaries

10.1 English

```
3377 <*package>
3378 \zcDeclareLanguage { english }
3379 \zcDeclareLanguageAlias { american } { english }
3380 \zcDeclareLanguageAlias { australian } { english }
3381 \zcDeclareLanguageAlias { british } { english }
3382 \zcDeclareLanguageAlias { canadian } { english }
3383 \zcDeclareLanguageAlias { newzealand } { english }
3384 \zcDeclareLanguageAlias { UKenglish } { english }
3385 \zcDeclareLanguageAlias { USenglish } { english }
3386 </package>
3387 <*dict-english>
3388 namesep = {\nobreakspace} ,
3389 pairsep = {\~and\nobreakspace} ,
3390 listsep = {,~} ,
3391 lastsep = {\~and\nobreakspace} ,
3392 tpairsep = {\~and\nobreakspace} ,
3393 tlistsep = {,~} ,
3394 tlastsep = {\~and\nobreakspace} ,
3395 notesep = {\~} ,
3396 rangesep = {\~to\nobreakspace} ,
3397
3398 type = part ,
3399   Name-sg = Part ,
3400   name-sg = part ,
3401   Name-pl = Parts ,
3402   name-pl = parts ,
3403
3404 type = chapter ,
3405   Name-sg = Chapter ,
3406   name-sg = chapter ,
3407   Name-pl = Chapters ,
3408   name-pl = chapters ,
3409
3410 type = section ,
3411   Name-sg = Section ,
3412   name-sg = section ,
3413   Name-pl = Sections ,
3414   name-pl = sections ,
3415
3416 type = paragraph ,
3417   Name-sg = Paragraph ,
3418   name-sg = paragraph ,
3419   Name-pl = Paragraphs ,
3420   name-pl = paragraphs ,
3421   Name-sg-ab = Par. ,
3422   name-sg-ab = par. ,
3423   Name-pl-ab = Par. ,
3424   name-pl-ab = par. ,
3425
3426 type = appendix ,
```

```

3427 Name-sg = Appendix ,
3428 name-sg = appendix ,
3429 Name-pl = Appendices ,
3430 name-pl = appendices ,
3431
3432 type = subappendix ,
3433 Name-sg = Appendix ,
3434 name-sg = appendix ,
3435 Name-pl = Appendices ,
3436 name-pl = appendices ,
3437
3438 type = page ,
3439 Name-sg = Page ,
3440 name-sg = page ,
3441 Name-pl = Pages ,
3442 name-pl = pages ,
3443 name-sg-ab = p. ,
3444 name-pl-ab = pp. ,
3445 rangesep = {\textendash} ,
3446
3447 type = line ,
3448 Name-sg = Line ,
3449 name-sg = line ,
3450 Name-pl = Lines ,
3451 name-pl = lines ,
3452
3453 type = figure ,
3454 Name-sg = Figure ,
3455 name-sg = figure ,
3456 Name-pl = Figures ,
3457 name-pl = figures ,
3458 Name-sg-ab = Fig. ,
3459 name-sg-ab = fig. ,
3460 Name-pl-ab = Figs. ,
3461 name-pl-ab = figs. ,
3462
3463 type = table ,
3464 Name-sg = Table ,
3465 name-sg = table ,
3466 Name-pl = Tables ,
3467 name-pl = tables ,
3468
3469 type = item ,
3470 Name-sg = Item ,
3471 name-sg = item ,
3472 Name-pl = Items ,
3473 name-pl = items ,
3474
3475 type = footnote ,
3476 Name-sg = Footnote ,
3477 name-sg = footnote ,
3478 Name-pl = Footnotes ,
3479 name-pl = footnotes ,
3480

```

```

3481 type = note ,
3482     Name-sg = Note ,
3483     name-sg = note ,
3484     Name-pl = Notes ,
3485     name-pl = notes ,
3486
3487 type = equation ,
3488     Name-sg = Equation ,
3489     name-sg = equation ,
3490     Name-pl = Equations ,
3491     name-pl = equations ,
3492     Name-sg-ab = Eq. ,
3493     name-sg-ab = eq. ,
3494     Name-pl-ab = Eqs. ,
3495     name-pl-ab = eqs. ,
3496     refpre-in = {() ,
3497     refpos-in = {} } ,
3498
3499 type = theorem ,
3500     Name-sg = Theorem ,
3501     name-sg = theorem ,
3502     Name-pl = Theorems ,
3503     name-pl = theorems ,
3504
3505 type = lemma ,
3506     Name-sg = Lemma ,
3507     name-sg = lemma ,
3508     Name-pl = Lemmas ,
3509     name-pl = lemmas ,
3510
3511 type = corollary ,
3512     Name-sg = Corollary ,
3513     name-sg = corollary ,
3514     Name-pl = Corollaries ,
3515     name-pl = corollaries ,
3516
3517 type = proposition ,
3518     Name-sg = Proposition ,
3519     name-sg = proposition ,
3520     Name-pl = Propositions ,
3521     name-pl = propositions ,
3522
3523 type = definition ,
3524     Name-sg = Definition ,
3525     name-sg = definition ,
3526     Name-pl = Definitions ,
3527     name-pl = definitions ,
3528
3529 type = proof ,
3530     Name-sg = Proof ,
3531     name-sg = proof ,
3532     Name-pl = Proofs ,
3533     name-pl = proofs ,
3534

```

```

3535 type = result ,
3536   Name-sg = Result ,
3537   name-sg = result ,
3538   Name-pl = Results ,
3539   name-pl = results ,
3540
3541 type = remark ,
3542   Name-sg = Remark ,
3543   name-sg = remark ,
3544   Name-pl = Remarks ,
3545   name-pl = remarks ,
3546
3547 type = example ,
3548   Name-sg = Example ,
3549   name-sg = example ,
3550   Name-pl = Examples ,
3551   name-pl = examples ,
3552
3553 type = algorithm ,
3554   Name-sg = Algorithm ,
3555   name-sg = algorithm ,
3556   Name-pl = Algorithms ,
3557   name-pl = algorithms ,
3558
3559 type = listing ,
3560   Name-sg = Listing ,
3561   name-sg = listing ,
3562   Name-pl = Listings ,
3563   name-pl = listings ,
3564
3565 type = exercise ,
3566   Name-sg = Exercise ,
3567   name-sg = exercise ,
3568   Name-pl = Exercises ,
3569   name-pl = exercises ,
3570
3571 type = solution ,
3572   Name-sg = Solution ,
3573   name-sg = solution ,
3574   Name-pl = Solutions ,
3575   name-pl = solutions ,
3576 </dict-english>

```

10.2 German

```

3577 <*package>
3578 \zcDeclareLanguage
3579 [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
3580 { german }
3581 \zcDeclareLanguageAlias { austrian } { german }
3582 \zcDeclareLanguageAlias { germanb } { german }
3583 \zcDeclareLanguageAlias { ngerman } { german }
3584 \zcDeclareLanguageAlias { naustrian } { german }
3585 \zcDeclareLanguageAlias { nswissgerman } { german }
3586 \zcDeclareLanguageAlias { swissgerman } { german }

```

```

3587 </package>
3588 <*dict-german>
3589 namesep = {\nobreakspace} ,
3590 pairsep = {\~und\nobreakspace} ,
3591 listsep = {\,~} ,
3592 lastsep = {\~und\nobreakspace} ,
3593 tpairsep = {\~und\nobreakspace} ,
3594 tlistsep = {\,~} ,
3595 tlastsep = {\~und\nobreakspace} ,
3596 notesep = {\~} ,
3597 rangesep = {\~bis\nobreakspace} ,
3598
3599 type = part ,
3600   gender = m ,
3601   case = N ,
3602     Name-sg = Teil ,
3603     Name-pl = Teile ,
3604   case = A ,
3605     Name-sg = Teil ,
3606     Name-pl = Teile ,
3607   case = D ,
3608     Name-sg = Teil ,
3609     Name-pl = Teilen ,
3610   case = G ,
3611     Name-sg = Teiles ,
3612     Name-pl = Teile ,
3613
3614 type = chapter ,
3615   gender = n ,
3616   case = N ,
3617     Name-sg = Kapitel ,
3618     Name-pl = Kapitel ,
3619   case = A ,
3620     Name-sg = Kapitel ,
3621     Name-pl = Kapitel ,
3622   case = D ,
3623     Name-sg = Kapitel ,
3624     Name-pl = Kapiteln ,
3625   case = G ,
3626     Name-sg = Kapitels ,
3627     Name-pl = Kapitel ,
3628
3629 type = section ,
3630   gender = m ,
3631   case = N ,
3632     Name-sg = Abschnitt ,
3633     Name-pl = Abschnitte ,
3634   case = A ,
3635     Name-sg = Abschnitt ,
3636     Name-pl = Abschnitte ,
3637   case = D ,
3638     Name-sg = Abschnitt ,
3639     Name-pl = Abschnitten ,

```

```

3640     case = G ,
3641         Name-sg = Abschnitts ,
3642         Name-pl = Abschnitte ,
3643
3644 type = paragraph ,
3645     gender = m ,
3646     case = N ,
3647         Name-sg = Absatz ,
3648         Name-pl = Absätze ,
3649     case = A ,
3650         Name-sg = Absatz ,
3651         Name-pl = Absätze ,
3652     case = D ,
3653         Name-sg = Absatz ,
3654         Name-pl = Absätzen ,
3655     case = G ,
3656         Name-sg = Absatzes ,
3657         Name-pl = Absätze ,
3658
3659 type = appendix ,
3660     gender = m ,
3661     case = N ,
3662         Name-sg = Anhang ,
3663         Name-pl = Anhänge ,
3664     case = A ,
3665         Name-sg = Anhang ,
3666         Name-pl = Anhänge ,
3667     case = D ,
3668         Name-sg = Anhang ,
3669         Name-pl = Anhängen ,
3670     case = G ,
3671         Name-sg = Anhangs ,
3672         Name-pl = Anhänge ,
3673
3674 type = subappendix ,
3675     gender = m ,
3676     case = N ,
3677         Name-sg = Anhang ,
3678         Name-pl = Anhänge ,
3679     case = A ,
3680         Name-sg = Anhang ,
3681         Name-pl = Anhänge ,
3682     case = D ,
3683         Name-sg = Anhang ,
3684         Name-pl = Anhängen ,
3685     case = G ,
3686         Name-sg = Anhangs ,
3687         Name-pl = Anhänge ,
3688
3689 type = page ,
3690     gender = f ,
3691     case = N ,
3692         Name-sg = Seite ,
3693         Name-pl = Seiten ,

```

```

3694 case = A ,
3695     Name-sg = Seite ,
3696     Name-pl = Seiten ,
3697 case = D ,
3698     Name-sg = Seite ,
3699     Name-pl = Seiten ,
3700 case = G ,
3701     Name-sg = Seite ,
3702     Name-pl = Seiten ,
3703 rangesep = {\textendash} ,
3704
3705 type = line ,
3706     gender = f ,
3707     case = N ,
3708         Name-sg = Zeile ,
3709         Name-pl = Zeilen ,
3710 case = A ,
3711     Name-sg = Zeile ,
3712     Name-pl = Zeilen ,
3713 case = D ,
3714     Name-sg = Zeile ,
3715     Name-pl = Zeilen ,
3716 case = G ,
3717     Name-sg = Zeile ,
3718     Name-pl = Zeilen ,
3719
3720 type = figure ,
3721     gender = f ,
3722     case = N ,
3723         Name-sg = Abbildung ,
3724         Name-pl = Abbildungen ,
3725         Name-sg-ab = Abb. ,
3726         Name-pl-ab = Abb. ,
3727 case = A ,
3728     Name-sg = Abbildung ,
3729     Name-pl = Abbildungen ,
3730     Name-sg-ab = Abb. ,
3731     Name-pl-ab = Abb. ,
3732 case = D ,
3733     Name-sg = Abbildung ,
3734     Name-pl = Abbildungen ,
3735     Name-sg-ab = Abb. ,
3736     Name-pl-ab = Abb. ,
3737 case = G ,
3738     Name-sg = Abbildung ,
3739     Name-pl = Abbildungen ,
3740     Name-sg-ab = Abb. ,
3741     Name-pl-ab = Abb. ,
3742
3743 type = table ,
3744     gender = f ,
3745     case = N ,
3746         Name-sg = Tabelle ,
3747         Name-pl = Tabellen ,

```

```

3748 case = A ,
3749     Name-sg = Tabelle ,
3750     Name-pl = Tabellen ,
3751 case = D ,
3752     Name-sg = Tabelle ,
3753     Name-pl = Tabellen ,
3754 case = G ,
3755     Name-sg = Tabelle ,
3756     Name-pl = Tabellen ,
3757
3758 type = item ,
3759     gender = m ,
3760     case = N ,
3761         Name-sg = Punkt ,
3762         Name-pl = Punkte ,
3763     case = A ,
3764         Name-sg = Punkt ,
3765         Name-pl = Punkte ,
3766     case = D ,
3767         Name-sg = Punkt ,
3768         Name-pl = Punkten ,
3769     case = G ,
3770         Name-sg = Punktes ,
3771         Name-pl = Punkte ,
3772
3773 type = footnote ,
3774     gender = f ,
3775     case = N ,
3776         Name-sg = Fußnote ,
3777         Name-pl = Fußnoten ,
3778     case = A ,
3779         Name-sg = Fußnote ,
3780         Name-pl = Fußnoten ,
3781     case = D ,
3782         Name-sg = Fußnote ,
3783         Name-pl = Fußnoten ,
3784     case = G ,
3785         Name-sg = Fußnote ,
3786         Name-pl = Fußnoten ,
3787
3788 type = note ,
3789     gender = f ,
3790     case = N ,
3791         Name-sg = Anmerkung ,
3792         Name-pl = Anmerkungen ,
3793     case = A ,
3794         Name-sg = Anmerkung ,
3795         Name-pl = Anmerkungen ,
3796     case = D ,
3797         Name-sg = Anmerkung ,
3798         Name-pl = Anmerkungen ,
3799     case = G ,
3800         Name-sg = Anmerkung ,
3801         Name-pl = Anmerkungen ,

```



```

3802
3803 type = equation ,
3804     gender = f ,
3805     case = N ,
3806         Name-sg = Gleichung ,
3807         Name-pl = Gleichungen ,
3808     case = A ,
3809         Name-sg = Gleichung ,
3810         Name-pl = Gleichungen ,
3811     case = D ,
3812         Name-sg = Gleichung ,
3813         Name-pl = Gleichungen ,
3814     case = G ,
3815         Name-sg = Gleichung ,
3816         Name-pl = Gleichungen ,
3817     refpre-in = {()} ,
3818     refpos-in = {} ,
3819
3820 type = theorem ,
3821     gender = n ,
3822     case = N ,
3823         Name-sg = Theorem ,
3824         Name-pl = Theoreme ,
3825     case = A ,
3826         Name-sg = Theorem ,
3827         Name-pl = Theoreme ,
3828     case = D ,
3829         Name-sg = Theorem ,
3830         Name-pl = Theoremen ,
3831     case = G ,
3832         Name-sg = Theorems ,
3833         Name-pl = Theoreme ,
3834
3835 type = lemma ,
3836     gender = n ,
3837     case = N ,
3838         Name-sg = Lemma ,
3839         Name-pl = Lemmata ,
3840     case = A ,
3841         Name-sg = Lemma ,
3842         Name-pl = Lemmata ,
3843     case = D ,
3844         Name-sg = Lemma ,
3845         Name-pl = Lemmata ,
3846     case = G ,
3847         Name-sg = Lemmas ,
3848         Name-pl = Lemmata ,
3849
3850 type = corollary ,
3851     gender = n ,
3852     case = N ,
3853         Name-sg = Korollar ,
3854         Name-pl = Korollare ,
3855     case = A ,

```

```

3856     Name-sg = Korollar ,
3857     Name-pl = Korollare ,
3858     case = D ,
3859     Name-sg = Korollar ,
3860     Name-pl = Korollaren ,
3861     case = G ,
3862     Name-sg = Korollars ,
3863     Name-pl = Korollare ,
3864
3865 type = proposition ,
3866     gender = m ,
3867     case = N ,
3868     Name-sg = Satz ,
3869     Name-pl = Sätze ,
3870     case = A ,
3871     Name-sg = Satz ,
3872     Name-pl = Sätze ,
3873     case = D ,
3874     Name-sg = Satz ,
3875     Name-pl = Sätzen ,
3876     case = G ,
3877     Name-sg = Satzes ,
3878     Name-pl = Sätze ,
3879
3880 type = definition ,
3881     gender = f ,
3882     case = N ,
3883     Name-sg = Definition ,
3884     Name-pl = Definitionen ,
3885     case = A ,
3886     Name-sg = Definition ,
3887     Name-pl = Definitionen ,
3888     case = D ,
3889     Name-sg = Definition ,
3890     Name-pl = Definitionen ,
3891     case = G ,
3892     Name-sg = Definition ,
3893     Name-pl = Definitionen ,
3894
3895 type = proof ,
3896     gender = m ,
3897     case = N ,
3898     Name-sg = Beweis ,
3899     Name-pl = Beweise ,
3900     case = A ,
3901     Name-sg = Beweis ,
3902     Name-pl = Beweise ,
3903     case = D ,
3904     Name-sg = Beweis ,
3905     Name-pl = Beweisen ,
3906     case = G ,
3907     Name-sg = Beweises ,
3908     Name-pl = Beweise ,
3909

```

```

3910 type = result ,
3911     gender = n ,
3912     case = N ,
3913         Name-sg = Ergebnis ,
3914         Name-pl = Ergebnisse ,
3915     case = A ,
3916         Name-sg = Ergebnis ,
3917         Name-pl = Ergebnisse ,
3918     case = D ,
3919         Name-sg = Ergebnis ,
3920         Name-pl = Ergebnissen ,
3921     case = G ,
3922         Name-sg = Ergebnisses ,
3923         Name-pl = Ergebnisse ,
3924
3925 type = remark ,
3926     gender = f ,
3927     case = N ,
3928         Name-sg = Bemerkung ,
3929         Name-pl = Bemerkungen ,
3930     case = A ,
3931         Name-sg = Bemerkung ,
3932         Name-pl = Bemerkungen ,
3933     case = D ,
3934         Name-sg = Bemerkung ,
3935         Name-pl = Bemerkungen ,
3936     case = G ,
3937         Name-sg = Bemerkung ,
3938         Name-pl = Bemerkungen ,
3939
3940 type = example ,
3941     gender = n ,
3942     case = N ,
3943         Name-sg = Beispiel ,
3944         Name-pl = Beispiele ,
3945     case = A ,
3946         Name-sg = Beispiel ,
3947         Name-pl = Beispiele ,
3948     case = D ,
3949         Name-sg = Beispiel ,
3950         Name-pl = Beispielen ,
3951     case = G ,
3952         Name-sg = Beispiels ,
3953         Name-pl = Beispiele ,
3954
3955 type = algorithm ,
3956     gender = m ,
3957     case = N ,
3958         Name-sg = Algorithmus ,
3959         Name-pl = Algorithmen ,
3960     case = A ,
3961         Name-sg = Algorithmus ,
3962         Name-pl = Algorithmen ,
3963     case = D ,

```

```

3964     Name-sg = Algorithmus ,
3965     Name-pl = Algorithmen ,
3966     case = G ,
3967     Name-sg = Algorithmus ,
3968     Name-pl = Algorithmen ,
3969
3970 type = listing ,
3971     gender = n ,
3972     case = N ,
3973     Name-sg = Listing ,
3974     Name-pl = Listings ,
3975     case = A ,
3976     Name-sg = Listing ,
3977     Name-pl = Listings ,
3978     case = D ,
3979     Name-sg = Listing ,
3980     Name-pl = Listings ,
3981     case = G ,
3982     Name-sg = Listings ,
3983     Name-pl = Listings ,
3984
3985 type = exercise ,
3986     gender = f ,
3987     case = N ,
3988     Name-sg = Übungsaufgabe ,
3989     Name-pl = Übungsaufgaben ,
3990     case = A ,
3991     Name-sg = Übungsaufgabe ,
3992     Name-pl = Übungsaufgaben ,
3993     case = D ,
3994     Name-sg = Übungsaufgabe ,
3995     Name-pl = Übungsaufgaben ,
3996     case = G ,
3997     Name-sg = Übungsaufgabe ,
3998     Name-pl = Übungsaufgaben ,
3999
4000 type = solution ,
4001     gender = f ,
4002     case = N ,
4003     Name-sg = Lösung ,
4004     Name-pl = Lösungen ,
4005     case = A ,
4006     Name-sg = Lösung ,
4007     Name-pl = Lösungen ,
4008     case = D ,
4009     Name-sg = Lösung ,
4010     Name-pl = Lösungen ,
4011     case = G ,
4012     Name-sg = Lösung ,
4013     Name-pl = Lösungen ,
4014 </dict-german>

```

10.3 French

```

4015 <*package>

```

```

4016 \zcDeclareLanguage [ gender = { f , m } ] { french }
4017 \zcDeclareLanguageAlias { acadian } { french }
4018 \zcDeclareLanguageAlias { canadien } { french }
4019 \zcDeclareLanguageAlias { francais } { french }
4020 \zcDeclareLanguageAlias { frenchb } { french }
4021 \</package>
4022 \<dict-french>
4023 namesep = {\nobreakspace} ,
4024 pairsep = {\~et\nobreakspace} ,
4025 listsep = {,~} ,
4026 lastsep = {\~et\nobreakspace} ,
4027 tpairsep = {\~et\nobreakspace} ,
4028 tlistsep = {,~} ,
4029 tlastsep = {\~et\nobreakspace} ,
4030 notesep = {\~} ,
4031 rangesep = {\~à\nobreakspace} ,
4032
4033 type = part ,
4034     gender = f ,
4035     Name-sg = Partie ,
4036     name-sg = partie ,
4037     Name-pl = Parties ,
4038     name-pl = parties ,
4039
4040 type = chapter ,
4041     gender = m ,
4042     Name-sg = Chapitre ,
4043     name-sg = chapitre ,
4044     Name-pl = Chapitres ,
4045     name-pl = chapitres ,
4046
4047 type = section ,
4048     gender = f ,
4049     Name-sg = Section ,
4050     name-sg = section ,
4051     Name-pl = Sections ,
4052     name-pl = sections ,
4053
4054 type = paragraph ,
4055     gender = m ,
4056     Name-sg = Paragraphe ,
4057     name-sg = paragraphe ,
4058     Name-pl = Paragraphes ,
4059     name-pl = paragraphes ,
4060
4061 type = appendix ,
4062     gender = f ,
4063     Name-sg = Annexe ,
4064     name-sg = annexe ,
4065     Name-pl = Annexes ,
4066     name-pl = annexes ,
4067
4068 type = subappendix ,

```

```

4069     gender = f ,
4070     Name-sg = Annexe ,
4071     name-sg = annexe ,
4072     Name-pl = Annexes ,
4073     name-pl = annexes ,
4074
4075     type = page ,
4076     gender = f ,
4077     Name-sg = Page ,
4078     name-sg = page ,
4079     Name-pl = Pages ,
4080     name-pl = pages ,
4081     rangesep = {\textendash} ,
4082
4083     type = line ,
4084     gender = f ,
4085     Name-sg = Ligne ,
4086     name-sg = ligne ,
4087     Name-pl = Lignes ,
4088     name-pl = lignes ,
4089
4090     type = figure ,
4091     gender = f ,
4092     Name-sg = Figure ,
4093     name-sg = figure ,
4094     Name-pl = Figures ,
4095     name-pl = figures ,
4096
4097     type = table ,
4098     gender = f ,
4099     Name-sg = Table ,
4100     name-sg = table ,
4101     Name-pl = Tables ,
4102     name-pl = tables ,
4103
4104     type = item ,
4105     gender = m ,
4106     Name-sg = Point ,
4107     name-sg = point ,
4108     Name-pl = Points ,
4109     name-pl = points ,
4110
4111     type = footnote ,
4112     gender = f ,
4113     Name-sg = Note ,
4114     name-sg = note ,
4115     Name-pl = Notes ,
4116     name-pl = notes ,
4117
4118     type = note ,
4119     gender = f ,
4120     Name-sg = Note ,
4121     name-sg = note ,
4122     Name-pl = Notes ,

```

```

4123     name-pl = notes ,
4124
4125 type = equation ,
4126     gender = f ,
4127     Name-sg = Équation ,
4128     name-sg = équation ,
4129     Name-pl = Équations ,
4130     name-pl = équations ,
4131     refpre-in = {()} ,
4132     refpos-in = {} ,
4133
4134 type = theorem ,
4135     gender = m ,
4136     Name-sg = Théorème ,
4137     name-sg = théorème ,
4138     Name-pl = Théorèmes ,
4139     name-pl = théorèmes ,
4140
4141 type = lemma ,
4142     gender = m ,
4143     Name-sg = Lemme ,
4144     name-sg = lemme ,
4145     Name-pl = Lemmes ,
4146     name-pl = lemmes ,
4147
4148 type = corollary ,
4149     gender = m ,
4150     Name-sg = Corollaire ,
4151     name-sg = corollaire ,
4152     Name-pl = Corollaires ,
4153     name-pl = corollaires ,
4154
4155 type = proposition ,
4156     gender = f ,
4157     Name-sg = Proposition ,
4158     name-sg = proposition ,
4159     Name-pl = Propositions ,
4160     name-pl = propositions ,
4161
4162 type = definition ,
4163     gender = f ,
4164     Name-sg = Définition ,
4165     name-sg = définition ,
4166     Name-pl = Définitions ,
4167     name-pl = définitions ,
4168
4169 type = proof ,
4170     gender = f ,
4171     Name-sg = Démonstration ,
4172     name-sg = démonstration ,
4173     Name-pl = Démonstrations ,
4174     name-pl = démonstrations ,
4175
4176 type = result ,

```

```

4177     gender = m ,
4178     Name-sg = Résultat ,
4179     name-sg = résultat ,
4180     Name-pl = Résultats ,
4181     name-pl = résultats ,
4182
4183     type = remark ,
4184     gender = f ,
4185     Name-sg = Remarque ,
4186     name-sg = remarque ,
4187     Name-pl = Remarques ,
4188     name-pl = remarques ,
4189
4190     type = example ,
4191     gender = m ,
4192     Name-sg = Exemple ,
4193     name-sg = exemple ,
4194     Name-pl = Exemples ,
4195     name-pl = exemples ,
4196
4197     type = algorithm ,
4198     gender = m ,
4199     Name-sg = Algorithme ,
4200     name-sg = algorithme ,
4201     Name-pl = Algorithmes ,
4202     name-pl = algorithmes ,
4203
4204     type = listing ,
4205     gender = f ,
4206     Name-sg = Liste ,
4207     name-sg = liste ,
4208     Name-pl = Listes ,
4209     name-pl = listes ,
4210
4211     type = exercise ,
4212     gender = m ,
4213     Name-sg = Exercice ,
4214     name-sg = exercice ,
4215     Name-pl = Exercices ,
4216     name-pl = exercices ,
4217
4218     type = solution ,
4219     gender = f ,
4220     Name-sg = Solution ,
4221     name-sg = solution ,
4222     Name-pl = Solutions ,
4223     name-pl = solutions ,
4224 </dict-french>

```

10.4 Portuguese

```

4225 <*package>
4226 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
4227 \zcDeclareLanguageAlias { brazilian } { portuguese }
4228 \zcDeclareLanguageAlias { brazil    } { portuguese }

```



```

4229 \zcDeclareLanguageAlias { portuges } { portuguese }
4230 \</package>
4231 \<*dict-portuguese>
4232 namesep = {\nobreakspace} ,
4233 pairsep = {\~e\nobreakspace} ,
4234 listsep = {\,~} ,
4235 lastsep = {\~e\nobreakspace} ,
4236 tpairsep = {\~e\nobreakspace} ,
4237 tlistsep = {\,~} ,
4238 tlastsep = {\~e\nobreakspace} ,
4239 notesep = {\~} ,
4240 rangesep = {\~a\nobreakspace} ,
4241
4242 type = part ,
4243   gender = f ,
4244   Name-sg = Parte ,
4245   name-sg = parte ,
4246   Name-pl = Partes ,
4247   name-pl = partes ,
4248
4249 type = chapter ,
4250   gender = m ,
4251   Name-sg = Capítulo ,
4252   name-sg = capítulo ,
4253   Name-pl = Capítulos ,
4254   name-pl = capítulos ,
4255
4256 type = section ,
4257   gender = f ,
4258   Name-sg = Seção ,
4259   name-sg = seção ,
4260   Name-pl = Seções ,
4261   name-pl = seções ,
4262
4263 type = paragraph ,
4264   gender = m ,
4265   Name-sg = Parágrafo ,
4266   name-sg = parágrafo ,
4267   Name-pl = Parágrafos ,
4268   name-pl = parágrafos ,
4269   Name-sg-ab = Par. ,
4270   name-sg-ab = par. ,
4271   Name-pl-ab = Par. ,
4272   name-pl-ab = par. ,
4273
4274 type = appendix ,
4275   gender = m ,
4276   Name-sg = Apêndice ,
4277   name-sg = apêndice ,
4278   Name-pl = Apêndices ,
4279   name-pl = apêndices ,
4280
4281 type = subappendix ,

```

```

4282     gender = m ,
4283     Name-sg = Apêndice ,
4284     name-sg = apêndice ,
4285     Name-pl = Apêndices ,
4286     name-pl = apêndices ,
4287
4288     type = page ,
4289     gender = f ,
4290     Name-sg = Página ,
4291     name-sg = página ,
4292     Name-pl = Páginas ,
4293     name-pl = páginas ,
4294     name-sg-ab = p. ,
4295     name-pl-ab = pp. ,
4296     rangesep = {\textendash} ,
4297
4298     type = line ,
4299     gender = f ,
4300     Name-sg = Linha ,
4301     name-sg = linha ,
4302     Name-pl = Linhas ,
4303     name-pl = linhas ,
4304
4305     type = figure ,
4306     gender = f ,
4307     Name-sg = Figura ,
4308     name-sg = figura ,
4309     Name-pl = Figuras ,
4310     name-pl = figuras ,
4311     Name-sg-ab = Fig. ,
4312     name-sg-ab = fig. ,
4313     Name-pl-ab = Figs. ,
4314     name-pl-ab = figs. ,
4315
4316     type = table ,
4317     gender = f ,
4318     Name-sg = Tabela ,
4319     name-sg = tabela ,
4320     Name-pl = Tabelas ,
4321     name-pl = tabelas ,
4322
4323     type = item ,
4324     gender = m ,
4325     Name-sg = Item ,
4326     name-sg = item ,
4327     Name-pl = Itens ,
4328     name-pl = itens ,
4329
4330     type = footnote ,
4331     gender = f ,
4332     Name-sg = Nota ,
4333     name-sg = nota ,
4334     Name-pl = Notas ,
4335     name-pl = notas ,

```

```

4336
4337 type = note ,
4338     gender = f ,
4339     Name-sg = Nota ,
4340     name-sg = nota ,
4341     Name-pl = Notas ,
4342     name-pl = notas ,
4343
4344 type = equation ,
4345     gender = f ,
4346     Name-sg = Equação ,
4347     name-sg = equação ,
4348     Name-pl = Equações ,
4349     name-pl = equações ,
4350     Name-sg-ab = Eq. ,
4351     name-sg-ab = eq. ,
4352     Name-pl-ab = Eqs. ,
4353     name-pl-ab = eqs. ,
4354     refpre-in = {()} ,
4355     refpos-in = {} ,
4356
4357 type = theorem ,
4358     gender = m ,
4359     Name-sg = Teorema ,
4360     name-sg = teorema ,
4361     Name-pl = Teoremas ,
4362     name-pl = teoremas ,
4363
4364 type = lemma ,
4365     gender = m ,
4366     Name-sg = Lema ,
4367     name-sg = lema ,
4368     Name-pl = Lemas ,
4369     name-pl = lemas ,
4370
4371 type = corollary ,
4372     gender = m ,
4373     Name-sg = Corolário ,
4374     name-sg = corolário ,
4375     Name-pl = Corolários ,
4376     name-pl = corolários ,
4377
4378 type = proposition ,
4379     gender = f ,
4380     Name-sg = Proposição ,
4381     name-sg = proposição ,
4382     Name-pl = Proposições ,
4383     name-pl = proposições ,
4384
4385 type = definition ,
4386     gender = f ,
4387     Name-sg = Definição ,
4388     name-sg = definição ,
4389     Name-pl = Definições ,

```

```

4390     name-pl = definições ,
4391
4392     type = proof ,
4393     gender = f ,
4394     Name-sg = Demonstração ,
4395     name-sg = demonstração ,
4396     Name-pl = Demonstrações ,
4397     name-pl = demonstrações ,
4398
4399     type = result ,
4400     gender = m ,
4401     Name-sg = Resultado ,
4402     name-sg = resultado ,
4403     Name-pl = Resultados ,
4404     name-pl = resultados ,
4405
4406     type = remark ,
4407     gender = f ,
4408     Name-sg = Observação ,
4409     name-sg = observação ,
4410     Name-pl = Observações ,
4411     name-pl = observações ,
4412
4413     type = example ,
4414     gender = m ,
4415     Name-sg = Exemplo ,
4416     name-sg = exemplo ,
4417     Name-pl = Exemplos ,
4418     name-pl = exemplos ,
4419
4420     type = algorithm ,
4421     gender = m ,
4422     Name-sg = Algoritmo ,
4423     name-sg = algoritmo ,
4424     Name-pl = Algoritmos ,
4425     name-pl = algoritmos ,
4426
4427     type = listing ,
4428     gender = f ,
4429     Name-sg = Listagem ,
4430     name-sg = listagem ,
4431     Name-pl = Listagens ,
4432     name-pl = listagens ,
4433
4434     type = exercise ,
4435     gender = m ,
4436     Name-sg = Exercício ,
4437     name-sg = exercício ,
4438     Name-pl = Exercícios ,
4439     name-pl = exercícios ,
4440
4441     type = solution ,
4442     gender = f ,
4443     Name-sg = Solução ,

```

```

4444 name-sg = solução ,
4445 Name-pl = Soluções ,
4446 name-pl = soluções ,
4447 </dict-portuguese>

```

10.5 Spanish

```

4448 <*package>
4449 \zcDeclareLanguage [ gender = { f , m } ] { spanish }
4450 </package>
4451 <*dict-spanish>
4452 namesep = {\nobreakspace} ,
4453 pairsep = {\~y\nobreakspace} ,
4454 listsep = { , ~ } ,
4455 lastsep = {\~y\nobreakspace} ,
4456 tpairsep = {\~y\nobreakspace} ,
4457 tlistsep = { , ~ } ,
4458 tlastsep = {\~y\nobreakspace} ,
4459 notesep = { ~ } ,
4460 rangesep = {\~a\nobreakspace} ,
4461
4462 type = part ,
4463 gender = f ,
4464 Name-sg = Parte ,
4465 name-sg = parte ,
4466 Name-pl = Partes ,
4467 name-pl = partes ,
4468
4469 type = chapter ,
4470 gender = m ,
4471 Name-sg = Capítulo ,
4472 name-sg = capítulo ,
4473 Name-pl = Capítulos ,
4474 name-pl = capítulos ,
4475
4476 type = section ,
4477 gender = f ,
4478 Name-sg = Sección ,
4479 name-sg = sección ,
4480 Name-pl = Secciones ,
4481 name-pl = secciones ,
4482
4483 type = paragraph ,
4484 gender = m ,
4485 Name-sg = Párrafo ,
4486 name-sg = párrafo ,
4487 Name-pl = Párrafos ,
4488 name-pl = párrafos ,
4489
4490 type = appendix ,
4491 gender = m ,
4492 Name-sg = Apéndice ,
4493 name-sg = apéndice ,
4494 Name-pl = Apéndices ,

```

```

4495     name-pl = apéndices ,
4496
4497 type = subappendix ,
4498     gender = m ,
4499     Name-sg = Apéndice ,
4500     name-sg = apéndice ,
4501     Name-pl = Apéndices ,
4502     name-pl = apéndices ,
4503
4504 type = page ,
4505     gender = f ,
4506     Name-sg = Página ,
4507     name-sg = página ,
4508     Name-pl = Páginas ,
4509     name-pl = páginas ,
4510     rangesep = {\textendash} ,
4511
4512 type = line ,
4513     gender = f ,
4514     Name-sg = Línea ,
4515     name-sg = línea ,
4516     Name-pl = Líneas ,
4517     name-pl = líneas ,
4518
4519 type = figure ,
4520     gender = f ,
4521     Name-sg = Figura ,
4522     name-sg = figura ,
4523     Name-pl = Figuras ,
4524     name-pl = figuras ,
4525
4526 type = table ,
4527     gender = m ,
4528     Name-sg = Cuadro ,
4529     name-sg = cuadro ,
4530     Name-pl = Cuadros ,
4531     name-pl = cuadros ,
4532
4533 type = item ,
4534     gender = m ,
4535     Name-sg = Punto ,
4536     name-sg = punto ,
4537     Name-pl = Puntos ,
4538     name-pl = puntos ,
4539
4540 type = footnote ,
4541     gender = f ,
4542     Name-sg = Nota ,
4543     name-sg = nota ,
4544     Name-pl = Notas ,
4545     name-pl = notas ,
4546
4547 type = note ,
4548     gender = f ,

```

```

4549   Name-sg = Nota ,
4550   name-sg = nota ,
4551   Name-pl = Notas ,
4552   name-pl = notas ,
4553
4554   type = equation ,
4555   gender = f ,
4556   Name-sg = Ecuación ,
4557   name-sg = ecuación ,
4558   Name-pl = Ecuaciones ,
4559   name-pl = ecuaciones ,
4560   refpre-in = {} ,
4561   refpos-in = {} ,
4562
4563   type = theorem ,
4564   gender = m ,
4565   Name-sg = Teorema ,
4566   name-sg = teorema ,
4567   Name-pl = Teoremas ,
4568   name-pl = teoremas ,
4569
4570   type = lemma ,
4571   gender = m ,
4572   Name-sg = Lema ,
4573   name-sg = lema ,
4574   Name-pl = Lemas ,
4575   name-pl = lemas ,
4576
4577   type = corollary ,
4578   gender = m ,
4579   Name-sg = Corolario ,
4580   name-sg = corolario ,
4581   Name-pl = Corolarios ,
4582   name-pl = corolarios ,
4583
4584   type = proposition ,
4585   gender = f ,
4586   Name-sg = Proposición ,
4587   name-sg = proposición ,
4588   Name-pl = Proposiciones ,
4589   name-pl = proposiciones ,
4590
4591   type = definition ,
4592   gender = f ,
4593   Name-sg = Definición ,
4594   name-sg = definición ,
4595   Name-pl = Definiciones ,
4596   name-pl = definiciones ,
4597
4598   type = proof ,
4599   gender = f ,
4600   Name-sg = Demostración ,
4601   name-sg = demostración ,
4602   Name-pl = Demostraciones ,

```

```

4603     name-pl = demostraciones ,
4604
4605 type = result ,
4606     gender = m ,
4607     Name-sg = Resultado ,
4608     name-sg = resultado ,
4609     Name-pl = Resultados ,
4610     name-pl = resultados ,
4611
4612 type = remark ,
4613     gender = f ,
4614     Name-sg = Observación ,
4615     name-sg = observación ,
4616     Name-pl = Observaciones ,
4617     name-pl = observaciones ,
4618
4619 type = example ,
4620     gender = m ,
4621     Name-sg = Ejemplo ,
4622     name-sg = ejemplo ,
4623     Name-pl = Ejemplos ,
4624     name-pl = ejemplos ,
4625
4626 type = algorithm ,
4627     gender = m ,
4628     Name-sg = Algoritmo ,
4629     name-sg = algoritmo ,
4630     Name-pl = Algoritmos ,
4631     name-pl = algoritmos ,
4632
4633 type = listing ,
4634     gender = m ,
4635     Name-sg = Listado ,
4636     name-sg = listado ,
4637     Name-pl = Listados ,
4638     name-pl = listados ,
4639
4640 type = exercise ,
4641     gender = m ,
4642     Name-sg = Ejercicio ,
4643     name-sg = ejercicio ,
4644     Name-pl = Ejercicios ,
4645     name-pl = ejercicios ,
4646
4647 type = solution ,
4648     gender = f ,
4649     Name-sg = Solución ,
4650     name-sg = solución ,
4651     Name-pl = Soluciones ,
4652     name-pl = soluciones ,
4653 </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 | |
|---|---|
| <code>\</code> | 112, 127, 128, 133, 134, 139, 140, 145, 146, 198, 206, 207, 217 |
| <code>\{</code> | 198 |
| <code>\}</code> | 198 |
| † internal commands: | |
| <code>\i_zrefclever_current_counter_tl</code> | 5 |
| A | |
| <code>\AddToHook</code> | 100, 764, 779, 923, 959, 984, 1022, 1024, 1074, 1147, 1163, 1184, 3094, 3096, 3101, 3114, 3134, 3149, 3151, 3156, 3172, 3185, 3205, 3217, 3249, 3258, 3294, 3298, 3319, 3325, 3348 |
| <code>\appendix</code> | 80, 81 |
| <code>\appendixname</code> | 80 |
| B | |
| <code>\babelname</code> | 969 |
| <code>\babelprovide</code> | 17, 29 |
| <code>\begin</code> | 84 |
| bool commands: | |
| <code>\bool_case_true:</code> | 2 |
| <code>\bool_if:NTF</code> | 543, 554, 927, 931, 1590, 2008, 2103, 2233, 2255, 2286, 2332, 2381, 2404, 2408, 2414, 2424, 2430, 2587, 2749, 2751 |
| <code>\bool_if:nTF</code> | 68, 1675, 1684, 1693, 1748, 1758, 1782, 1799, 1814, 1879, 1887, 2022, 2030, 2267, 2274, 2281, 2539, 2660 |
| <code>\bool_lazy_all:nTF</code> | 2353, 2761, 3014 |
| <code>\bool_lazy_and:nnTF</code> | 1564, 1582, 2734, 2776, 3071 |
| <code>\bool_lazy_any:nTF</code> | 2894, 2903 |
| <code>\bool_lazy_or:nnTF</code> | 1568, 2722 |
| <code>\bool_new:N</code> | 492, 800, 801, 826, 850, 859, 866, 867, 880, 881, 900, 901, 1082, 1083, 1084, 1085, 1086, 1177, 1178, 1598, 1611, 1919, 1920, 1930, 1937, 1938, 3257 |
| <code>\bool_set:Nn</code> | 1561 |
| <code>\bool_set_false:N</code> | 813, 817, 908, 917, 918, 933, 1094, 1098, 1105, 1113, 1114, 1115, 1200, 1740, 1977, 2014, 2028, 2042, 2245, 2379, 2380, 2901, 2918 |
| <code>\bool_set_true:N</code> | 470, 564, 807, 808, 812, 818, 907, 912, 913, 1092, 1099, 1104, 1121, 1123, 1125, 1128, 1129, 1130, 1188, 1193, 1754, 1764, 1768, 1790, 1805, 1820, 1844, 1985, 2009, 2015, 2019, 2046, 2052, 2917, 2953, 2960, 2961, 2979, 2986, 2998, 3264 |
| <code>\bool_until_do:Nn</code> | 1780, 1978 |
| <code>\bool_while_do:nn</code> | 3353 |
| C | |
| clist commands: | |
| <code>\clist_map_inline:nn</code> | 1116, 1243, 3231, 3312 |
| <code>\counterwithin</code> | 4 |
| <code>\cref</code> | 83 |
| cs commands: | |
| <code>\cs_generate_variant:Nn</code> | 65, 251, 257, 560, 568, 1323, 1415, 1421, 2581, 2931 |
| <code>\cs_if_exist:NTF</code> | 25, 28, 46, 49, 58, 78, 3122, 3128 |
| <code>\cs_if_exist_p:N</code> | 3355 |
| <code>\cs_new:Npn</code> | 56, 66, 76, 87, 252, 258, 2535, 2582, 2921 |
| <code>\cs_new_eq:NN</code> | 3187, 3192, 3201, 3207, 3213 |
| <code>\cs_new_protected:Npn</code> | 246, 378, 493, 561, 569, 575, 746, 1321, 1410, 1416, 1556, 1615, 1657, 1668, 1727, 1857, 1909, 1954, 2110, 2375, 2531, 2533, 2712, 2932, 3008, 3065, 3265 |
| <code>\cs_new_protected:Npx</code> | 99 |
| <code>\cs_set_eq:NN</code> | 103, 3188, 3193, 3202, 3208, 3214 |
| <code>\cs_set_nopar:Npn</code> | 3176 |
| E | |
| <code>\endinput</code> | 12 |
| exp commands: | |
| <code>\exp_args:NNe</code> | 35, 38 |
| <code>\exp_args:NNNo</code> | 248 |
| <code>\exp_args:NNnx</code> | 346 |
| <code>\exp_args:NNo</code> | 248, 254 |
| <code>\exp_args:NNx</code> | 380, 384, 424, 510, 526, 1378, 1394 |
| <code>\exp_args:Nnx</code> | 571 |
| <code>\exp_args:No</code> | 254 |
| <code>\exp_args:Nx</code> | 504, 3270, 3278 |

| | | | |
|------------------------------|---|------------------------------|--|
| \exp_args:Nxx | 1711, 2945, 2967, 2971, 2988 | \int_compare_p:nNn | 1881, 1889, 2357, 2726, 2737, 2766, 2914 |
| \exp_not:N | 68, 2288, 2291, 2302, 2305, 2308, 2545, 2548, 2551, 2560, 2562, 2565, 2568, 2574, 2576, 2594, 2604, 2607, 2609, 2612, 2619, 2626, 2628, 2632, 2635, 2638, 2650, 2653, 2666, 2669, 2672, 2688, 2690, 2693, 2696, 2703, 2705 | \int_eval:n | 99 |
| \exp_not:n | 255, 2132, 2148, 2160, 2165, 2188, 2202, 2206, 2218, 2222, 2256, 2257, 2289, 2301, 2306, 2307, 2444, 2457, 2464, 2488, 2500, 2504, 2514, 2518, 2546, 2547, 2549, 2555, 2558, 2561, 2566, 2567, 2569, 2570, 2573, 2575, 2605, 2606, 2608, 2610, 2611, 2613, 2614, 2618, 2630, 2631, 2636, 2637, 2639, 2647, 2651, 2652, 2654, 2667, 2668, 2670, 2682, 2686, 2689, 2694, 2695, 2697, 2698, 2702, 2704 | \int_incr:N | 2370, 2407, 2409, 2423, 2425, 2429, 2431, 2529, 3369 |
| \ExplSyntaxOn | 17, 506 | \int_new:N | 1612, 1613, 1921, 1922, 1934, 1935 |
| F | | \int_set:Nn | 1866, 1868, 1872, 1875, 3352 |
| file commands: | | \int_to_roman:n | 3356, 3363, 3364, 3367 |
| \file_get:nnNTF | 504 | \int_use:N | 47, 50, 54, 60 |
| \fmtversion | 3 | \int_zero:N | 1859, 1860, 1963, 1964, 1965, 1966, 2369, 2371, 2372, 2524, 2525 |
| \footnote | 80 | \l_tmpa_int | 3352, 3356, 3363, 3364, 3367, 3369, 3371 |
| G | | iow commands: | |
| group commands: | | \iow_char:N | 112, 127, 128, 133, 134, 139, 140, 145, 146, 198, 206, 207, 217 |
| \group_begin: | 102, 325, 495, 563, 1373, 1558, 1572, 2288, 2305, 2545, 2548, 2565, 2568, 2604, 2609, 2612, 2628, 2635, 2650, 2666, 2669, 2693, 2696 | K | |
| \group_end: | 105, 337, 558, 566, 1407, 1575, 1595, 2302, 2308, 2560, 2562, 2574, 2576, 2607, 2619, 2626, 2632, 2638, 2653, 2688, 2690, 2703, 2705 | keys commands: | |
| I | | \keys_define:nn | 39, 354, 581, 636, 653, 667, 753, 783, 790, 802, 827, 836, 851, 860, 868, 882, 894, 902, 935, 942, 980, 1027, 1069, 1076, 1088, 1149, 1156, 1158, 1165, 1172, 1179, 1189, 1201, 1210, 1239, 1265, 1289, 1299, 1310, 1334, 1346, 1422, 1482, 1503, 1526 |
| \IfBooleanTF | 1601 | \keys_set:nn | 14, 17, 32, 34, 39, 45, 334, 536, 1194, 1322, 1329, 1404, 1559 |
| \ifdraft | 1097 | keyval commands: | |
| \IfFormatAtLeastTF | 3, 4, 3181 | \keyval_parse:nnn | 1214, 1269 |
| \ifoptionfinal | 1103 | L | |
| \input | 17 | \label | 82, 83, 86 |
| int commands: | | \labelformat | 3 |
| \int_case:nnTF | 2113, 2141, 2173, 2335, 2437, 2476 | \languagename | 29, 963 |
| \int_compare:nNnTF | 1715, 1791, 1806, 1821, 1833, 1845, 1865, 1867, 1911, 2074, 2128, 2162, 2324, 2326, 2392, 2417, 2461, 2949, 2955, 2975, 2981, 3371 | M | |
| J | | \mainbabelname | 29, 970 |
| K | | \MessageBreak | 10 |
| L | | MH commands: | |
| M | | \MH_if_boolean:nTF | 3262 |
| N | | msg commands: | |
| O | | \msg_info:nnn | 618, 644, 674, 3168, 3253, 3289, 3344, 3372 |
| P | | \msg_info:nnnn | 593, 600, 625 |
| Q | | \msg_info:nnnnn | 612 |
| R | | \msg_line_context: | 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>\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>\prop_if_in_p:Nn</code> 69, 3025 |
| <code>\msg_note:nnn</code> 539 | <code>\prop_item:Nn</code> 38, 70, 348, 387, 427, 462, 513, 529, 1381, 1397 |
| <code>\msg_warning:nn</code> 769, 794, 932, 938, 1168, 1205, 2359 | <code>\prop_new:N</code> 322, 332, 688, 1209, 1264, 1295, 1327 |
| <code>\msg_warning:nnn</code> 329, 350, 545, 555, 1010, 1055, 1072, 1134, 1145, 1271, 1338, 1406, 1459, 1494, 1533, 2067, 2240, 2755, 2771, 3057 | <code>\prop_put:Nnn</code> 750, 1306, 1361 |
| <code>\msg_warning:nnnn</code> .. 398, 415, 452, 475, 1216, 1434, 1441, 1471, 2833, 2881 | <code>\prop_remove:Nn</code> 749, 1305, 1353 |
| <code>\msg_warning:nnnnn</code> 438, 482, 1453, 2792 | <code>\providecommand</code> 3 |
| <code>\msg_warning:nnnnnn</code> 2799 | <code>\ProvidesExplPackage</code> 14 |
| | <code>\ProvidesFile</code> 17 |
| | R |
| | <code>\refstepcounter</code> 3, 80, 84, 87 |
| | <code>\renewlist</code> 87 |
| | <code>\RequirePackage</code> 16, 17, 18, 19, 20, 928, 1160, 1181 |
| N | S |
| <code>\newcounter</code> 4, 3118, 3119 | <code>\scantokens</code> 81 |
| <code>\NewDocumentCommand</code> 323, 340, 1319, 1324, 1371, 1554, 1599 | seq commands: |
| <code>\nobreakspace</code> 695, 3388, 3389, 3391, 3392, 3394, 3396, 3589, 3590, 3592, 3593, 3595, 3597, 4023, 4024, 4026, 4027, 4029, 4031, 4232, 4233, 4235, 4236, 4238, 4240, 4452, 4453, 4455, 4456, 4458, 4460 | <code>\seq_clear:N</code> 847, 1617 |
| | <code>\seq_const_from_clist:Nn</code> 265, 273, 286, 298, 301 |
| | <code>\seq_gconcat:NNN</code> ... 309, 312, 316, 319 |
| | <code>\seq_get_left:NN</code> 408, 419, 523, 602, 1391, 1443, 1988 |
| | <code>\seq_gput_right:Nn</code> 537, 548 |
| | <code>\seq_if_empty:NnTF</code> 394, 434, 520, 591, 610, 1388, 1432, 1451, 1982 |
| P | <code>\seq_if_in:NnTF</code> 412, 449, 500, 597, 622, 1245, 1438, 1463, 1661 |
| <code>\PackageError</code> 7 | <code>\seq_map_break:n</code> 90, 1900, 1903 |
| <code>\pagenumbering</code> 7 | <code>\seq_map_function:NN</code> 1620 |
| <code>\pageref</code> 46 | <code>\seq_map_indexed_inline:Nn</code> . 26, 1861 |
| prg commands: | <code>\seq_map_inline:Nn</code> 633, 650, 664, 1296, 1331, 1343, 1479, 1500, 1523, 1897, 3268 |
| <code>\prg_generate_conditional_</code> <code>variant:Nnn</code> 719, 735 | <code>\seq_map_tokens:Nn</code> 72 |
| <code>\prg_new_protected_conditional:Npnn</code> 705, 721, 738 | <code>\seq_new:N</code> 263, 264, 308, 315, 491, 835, 1238, 1597, 1614, 1918 |
| <code>\prg_return_false:</code> 715, 717, 731, 733, 744 | <code>\seq_pop_left:NN</code> 1980 |
| <code>\prg_return_true:</code> 714, 730, 743 | <code>\seq_put_right:Nn</code> 1247, 1664 |
| <code>\ProcessKeysOptions</code> 1318 | <code>\seq_reverse:N</code> 841 |
| prop commands: | <code>\seq_set_eq:NN</code> 1956 |
| <code>\prop_get:NnN</code> 3034 | <code>\seq_set_from_clist:Nn</code> 384, 424, 510, 526, 840, 1378, 1394, 1560 |
| <code>\prop_get:NnNTF</code> 380, 497, 708, 711, 724, 727, 741, 1374, 2812, 2840, 2848, 3011, 3068, 3081 | <code>\seq_sort:Nn</code> 48, 1623 |
| <code>\prop_gput:Nnn</code> 331, 347, 358, 365, 372, 1412, 1418 | <code>\setcounter</code> .. 3120, 3121, 3137, 3150, 3154 |
| <code>\prop_gput_if_new:Nnn</code> 571, 577 | sort commands: |
| <code>\prop_gset_from_keyval:Nn</code> 689 | <code>\sort_return_same:</code> 48, 53, 1630, 1635, 1682, 1720, 1722, 1755, 1775, 1796, 1811, 1825, 1850, 1885, 1900, 1916 |
| <code>\prop_if_exist:NnTF</code> 1326 | |
| <code>\prop_if_exist_p:N</code> 3018, 3074 | |
| <code>\prop_if_in:NnTF</code> 35, 328, 344, 1005, 1050 | |

| | |
|--|--|
| <code>\sort_return_swapped:</code> . . . 48, 53, 1643, 1691, 1719, 1765, 1774, 1795, 1810, 1826, 1849, 1893, 1903, 1915 | <code>\zref@ifrefcontainsprop</code> 24, 2537, 2589, 2656, 2926 |
| <code>\stepcounter</code> 3136, 3153 | <code>\zref@ifrefundefined</code> 1625, 1627, 1639, 2011, 2013, 2018, 2062, 2237, 2246, 2383, 2584, 2714 |
| str commands: | <code>\zref@label</code> 82, 3179 |
| <code>\str_case:nnTF</code> 986, 1031, 1118 | <code>\ZREF@mainlist</code> 22, 32, 43, 53, 55, 97, 108 |
| <code>\str_compare:nNnTF</code> 1771 | <code>\zref@newprop</code> 5, 7, 21, 23, 33, 44, 54, 92, 107 |
| <code>\str_if_eq:nnTF</code> 89, 460 | <code>\zref@refused</code> 2061 |
| <code>\str_if_eq_p:nn</code> 2899, 2905, 2907, 2911 | <code>\zref@wrapper@babel</code> 44, 82, 1555, 3179 |
| <code>\str_new:N</code> 941 | <code>\textendash</code> 699, 3445, 3703, 4081, 4296, 4510 |
| <code>\str_set:Nn</code> 946, 948, 950, 952 | <code>\textup</code> 85 |
| <code>\string</code> 3275, 3283 | <code>\the</code> 3 |
| T | <code>\thechapter</code> 80 |
| <code>\tag</code> 84, 86 | <code>\thelstnumber</code> 87 |
| TeX and L ^A T _E X 2 _ε commands: | <code>\thepage</code> 6, 7, 104 |
| <code>\@Alph</code> 80 | <code>\thesection</code> 80 |
| <code>\@addtoreset</code> 4 | tl commands: |
| <code>\@auxout</code> 3274, 3282 | <code>\c_empty_tl</code> 1660, 1671, 1673, 1730, 1733, 1736, 1738, 1999, 2001, 2924, 2927, 2928, 2935, 2937 |
| <code>\@bsphack</code> 496, 3267 | <code>\c_novalue_tl</code> 1301, 1348 |
| <code>\@chapapp</code> 80 | <code>\tl_clear:N</code> 402, 443, 456, 478, 487, 509, 521, 586, 1377, 1389, 1427, 1958, 1959, 1960, 1961, 1962, 1984, 2365, 2366, 2367, 2368, 2406, 2715, 2718, 2746, 2785, 2832, 2880, 3056, 3087, 3089 |
| <code>\@currentcounter</code> 3, 5, 37, 80, 84, 87, 28, 29, 49, 50, 1293 | <code>\tl_gset:Nn</code> 104 |
| <code>\@currentlabel</code> 3, 84, 87 | <code>\tl_head:N</code> 1809, 1822, 1834, 1836, 1846, 1848 |
| <code>\@elt</code> 5 | <code>\tl_if_empty:NTF</code> 80, 396, 406, 436, 447, 473, 480, 616, 641, 658, 672, 678, 1457, 1487, 1508, 1531, 1537, 1576, 2065, 2235, 2644, 2731, 2753, 2790, 2810, 2820, 2856, 3340 |
| <code>\@esphack</code> 557, 3287 | <code>\tl_if_empty:nTF</code> 326, 342, 585, 748, 1426, 2184, 2200, 2216, 2455, 2486, 2498, 2512, 2717 |
| <code>\@ifl@t@r</code> 3 | <code>\tl_if_empty_p:N</code> 1679, 1680, 1688, 1689, 1696, 1697, 2025, 2026, 2033, 2035, 2764, 2778, 2898, 2908, 2912, 3016, 3072 |
| <code>\@ifpackageloaded</code> 766, 781, 925, 961, 967, 1186, 3116, 3174, 3183, 3197, 3199, 3260, 3296, 3327, 3350 | <code>\tl_if_empty_p:n</code> 1750, 1751, 1760, 1761, 1786, 1787, 1802, 1817 |
| <code>\@onlypreamble</code> 339, 353, 1409 | <code>\tl_if_eq:NNTF</code> 1700, 1744, 2038, 2786, 2939 |
| <code>\@raw@opt<(package)>.sty</code> 34 | <code>\tl_if_eq:NnTF</code> 1618, 1650, 1871, 1874, 1899, 1902, 1992, 2943 |
| <code>\bbl@loaded</code> 29 | <code>\tl_if_eq:nnTF</code> 1711, 1863, 2945, 2967, 2971, 2988, 3270, 3278 |
| <code>\bbl@main@language</code> 29, 964 | <code>\tl_if_novalue:nTF</code> 1304, 1351 |
| <code>\c@</code> 4 | <code>\tl_map_break:n</code> 90 |
| <code>\c@enumN</code> 88 | |
| <code>\c@lstnumber</code> 87 | |
| <code>\c@page</code> 7, 103 | |
| <code>\cl@</code> 5 | |
| <code>\hyper@@link</code> 68, 2291, 2551, 2594, 2672 | |
| <code>\lst@AddToHook</code> 3338 | |
| <code>\lst@label</code> 3340, 3341 | |
| <code>\ltx@gobble</code> 82 | |
| <code>\ltx@label</code> 83, 3187, 3188, 3192, 3193, 3201, 3202, 3207, 3208, 3213, 3214 | |
| <code>\MT@newlabel</code> 3275, 3283 | |
| <code>\p@...</code> 3 | |
| <code>\protected@write</code> 3274, 3282 | |
| <code>\zref@addprop</code> 22, 32, 43, 53, 55, 97, 108 | |
| <code>\zref@default</code> 68, 2532, 2534 | |
| <code>\zref@extractdefault</code> 10, 76, 249, 255, 259 | |
| <code>\zref@ifpropundefined</code> 24, 2923 | |

| | | | |
|--|--|--|---|
| <code>\tl_map_tokens:Nn</code> | 82 | <code>\zrefcheck_zceref_run_checks_on-</code> | |
| <code>\tl_new:N</code> | 98, 260, 261, 262, 752, 956, 957, 958, 1068, 1087, 1155, 1171, 1288, 1605, 1606, 1607, 1608, 1609, 1610, 1923, 1924, 1925, 1926, 1927, 1928, 1929, 1931, 1932, 1933, 1936, 1939, 1940, 1941, 1942, 1943, 1944, 1945, 1946, 1947, 1948, 1949, 1950, 1951, 1952, 1953, 3092 | <code>labels:n</code> | 1587 |
| <code>\tl_put_left:Nn</code> | 2270, 2277, 2317, 2822, 2823, 2858, 2860, 2862, 2864 | zrefclever internal commands: | |
| <code>\tl_put_right:Nn</code> | 2130, 2146, 2155, 2186, 2197, 2213, 2442, 2453, 2484, 2496, 2510, 2732, 2733, 2744 | <code>\l_zrefclever_abbrev_bool</code> | 880, 884, 2735 |
| <code>\tl_reverse:N</code> | 1731, 1734 | <code>\l_zrefclever_capitalize_bool</code> | 16, 470, 866, 870, 2723 |
| <code>\tl_set:Nn</code> | . 248, 333, 587, 598, 757, 759, 761, 767, 770, 786, 795, 963, 964, 969, 970, 973, 974, 977, 990, 998, 1007, 1012, 1035, 1043, 1052, 1057, 1328, 1428, 1439, 1838, 1840, 1994, 1995, 2119, 2121, 2253, 2284, 2396, 2398, 2421, 2728, 2729, 2742, 3093, 3095 | <code>\l_zrefclever_capitalize_first-</code> <code>bool</code> | 867, 876, 2725 |
| <code>\tl_set_eq:NN</code> | 2363 | <code>__zrefclever_counter_reset_by:n</code> | 6, 36, 58, 60, 62, 66, 3224, 3305 |
| <code>\tl_tail:N</code> | 1839, 1841 | <code>__zrefclever_counter_reset_by-</code> <code>aux:nn</code> | 73, 76 |
| <code>\l_tmpa_tl</code> | 507, 536, 1578, 1579 | <code>__zrefclever_counter_reset_by-</code> <code>aux:nnn</code> | 83, 87 |
| U | | | |
| <code>\upshape</code> | 3252 | <code>\l_zrefclever_counter_resetby-</code> <code>prop</code> | 5, 36, 69, 70, 1264, 1276 |
| use commands: | | | |
| <code>\use:N</code> | 26, 29 | <code>\l_zrefclever_counter_resettters-</code> <code>seq</code> | 5, 36, 72, 1238, 1245, 1248 |
| V | | | |
| <code>\value</code> | 3137, 3154 | <code>\l_zrefclever_counter_type_prop</code> | 4, 35, 35, 38, 1209, 1221 |
| Z | | | |
| <code>\zcDeclareLanguage</code> | 13, 14, 45, 323, 3378, 3578, 4016, 4226, 4449 | <code>\l_zrefclever_current_counter-</code> <code>tl</code> | 3, 37, 21, 25, 26, 36, 39, 41, 46, 47, 95, 1288, 1291 |
| <code>\zcDeclareLanguageAlias</code> | 13, 340, 3379, 3380, 3381, 3382, 3383, 3384, 3385, 3581, 3582, 3583, 3584, 3585, 3586, 4017, 4018, 4019, 4020, 4227, 4228, 4229 | <code>\l_zrefclever_current_language-</code> <code>tl</code> | 29, 958, 963, 969, 973, 999, 1044 |
| <code>\zcLanguageSetup</code> | 11, 17, 19, 39–41, 1371 | <code>__zrefclever_declare_default-</code> <code>transl:nnn</code> | 41, 1410, 1489, 1510 |
| <code>\zcpageref</code> | 46, 1599 | <code>__zrefclever_declare_type-</code> <code>transl:nnnn</code> | 41, 1410, 1465, 1515, 1539, 1545 |
| <code>\zceref</code> | 31, 34, 37, 38, 44, 46–48, 55, 56, 83, 85, 1554, 1602, 1603 | <code>__zrefclever_def_extract:Nnnn</code> | 10, 246, 1659, 1670, 1672, 1729, 1732, 1735, 1737, 1998, 2000, 2934, 2936 |
| <code>\zcRefTypeSetup</code> | 11, 39, 1324, 3252 | <code>\g_zrefclever_dict_{language}_prop</code> | 17 |
| <code>\zcsetup</code> | 29, 34, 37, 38, 1319 | <code>\l_zrefclever_dict_decl_case_tl</code> | 260, 521, 524, 598, 603, 678, 682, 1389, 1392, 1439, 1444, 1537, 1548 |
| <code>\zlabel</code> | 82, 84, 86, 87, 3341 | <code>\l_zrefclever_dict_declension-</code> <code>seq</code> | 260, 385, 394, 408, 412, 419, 511, 520, 523, 591, 597, 602, 1379, 1388, 1391, 1432, 1438, 1443 |
| zrefcheck commands: | | | |
| <code>\zrefcheck_zceref_beg_label:</code> | 1567 | <code>\l_zrefclever_dict_gender_seq</code> | 260, 425, 434, 449, 527, 610, 622, 1395, 1451, 1463 |
| <code>\zrefcheck_zceref_end_label-</code> <code>maybe:</code> | 1586 | <code>\l_zrefclever_dict_language_tl</code> | 260, 333, 359, 366, 373, 382, 390, 430, 465, 498, 502, 505, 516, 532, 538, 540, 546, 549, 572, 578, 594, 601, 613, 626, 709, 712, 725, 728, 1375, |

1384, 1400, 1435, 1442, 1454, 1466,
 1472, 1490, 1511, 1516, 1540, 1546
 _zrefclever_extract:nnn
 .. 10, 258, 1716, 1718, 1792, 1794,
 1807, 1824, 1912, 1914, 2950, 2952,
 2956, 2958, 2976, 2978, 2982, 2984
 _zrefclever_extract_unexp:nnn .
 10, 76, 252,
 1712, 1713, 2297, 2553, 2556, 2571,
 2600, 2615, 2678, 2683, 2699, 2924,
 2927, 2928, 2946, 2947, 2968, 2969,
 2972, 2973, 2990, 2994, 3271, 3279
 _zrefclever_extract_url_-
 unexp:n 2293, 2552, 2596, 2674, 2921
 \g_zrefclever_fallback_dict_-
 prop 11, 688, 689, 741
 \l_zrefclever_footnote_type_tl .
 3092, 3093, 3095, 3099
 _zrefclever_get_default_-
 transl:nnN 11, 722, 736
 _zrefclever_get_default_-
 transl:nnNTF 23, 721, 3049
 _zrefclever_get_enclosing_-
 counters_value:n . 5, 6, 56, 61, 94
 _zrefclever_get_fallback_-
 transl:nN 739
 _zrefclever_get_fallback_-
 transl:nNTF 24, 737, 3054
 _zrefclever_get_ref:n
 67, 68, 2133, 2149,
 2161, 2166, 2189, 2203, 2207, 2219,
 2223, 2258, 2278, 2445, 2458, 2465,
 2489, 2501, 2505, 2515, 2519, 2535
 _zrefclever_get_ref_first: . . .
 67, 68, 72, 2271, 2318, 2582
 _zrefclever_get_ref_font:nN 11,
 22, 37, 78, 79, 2094, 2096, 2098, 3065
 _zrefclever_get_ref_string:nN .
 11, 22, 37, 78, 1578, 1969,
 1971, 1973, 2076, 2078, 2080, 2082,
 2084, 2086, 2088, 2090, 2092, 3008
 _zrefclever_get_type_transl:nnnN
 11, 706, 720
 _zrefclever_get_type_transl:nnnNTF
 23, 705, 2780, 2826, 2868, 2874, 3043
 \l_zrefclever_label_a_tl
 . 54, 1923, 1981, 1999, 2011, 2061,
 2062, 2068, 2120, 2133, 2149, 2166,
 2207, 2223, 2251, 2258, 2383, 2387,
 2397, 2422, 2445, 2466, 2505, 2519
 \l_zrefclever_label_b_tl
 54, 1923,
 1984, 1989, 2001, 2013, 2018, 2387
 \l_zrefclever_label_count_int ..
 54, 1921, 1963,
 2074, 2113, 2369, 2392, 2529, 2767
 \l_zrefclever_label_enclval_a_-
 tl 1605, 1729, 1731, 1786,
 1802, 1822, 1834, 1838, 1839, 1846
 \l_zrefclever_label_enclval_b_-
 tl 1605, 1732, 1734, 1787,
 1809, 1817, 1836, 1840, 1841, 1848
 \l_zrefclever_label_extdoc_a_tl
 1605, 1735,
 1745, 1750, 1760, 1773, 2934, 2940
 \l_zrefclever_label_extdoc_b_tl
 1605, 1737,
 1746, 1751, 1761, 1772, 2936, 2941
 \l_zrefclever_label_type_a_tl ..
 78, 1605, 1660, 1662,
 1665, 1671, 1679, 1688, 1696, 1701,
 1871, 1899, 1994, 1998, 2025, 2033,
 2039, 2065, 2122, 2399, 3016, 3021,
 3028, 3037, 3045, 3072, 3077, 3084
 \l_zrefclever_label_type_b_tl ..
 1605,
 1673, 1680, 1689, 1697, 1702, 1874,
 1902, 1995, 2000, 2026, 2035, 2040
 _zrefclever_label_type_put_-
 new_right:n 47, 48, 1621, 1657
 \l_zrefclever_label_types_seq ..
 48, 1614, 1617, 1661, 1664, 1897
 _zrefclever_labels_in_sequence:nn
 55, 76, 2249, 2386, 2932
 \g_zrefclever_languages_prop ...
 . 13, 322, 328, 331, 344, 347, 348,
 380, 497, 708, 724, 1005, 1050, 1374
 \l_zrefclever_last_of_type_bool
 54, 1918, 2009, 2014, 2015,
 2019, 2028, 2043, 2047, 2053, 2103
 \l_zrefclever_lastsep_tl . 1939,
 2085, 2148, 2165, 2188, 2206, 2218
 \l_zrefclever_link_star_bool ...
 1561, 1597, 2542, 2663, 2897
 \l_zrefclever_listsep_tl
 ... 1939, 2083, 2160, 2202, 2444,
 2457, 2464, 2488, 2500, 2504, 2514
 \l_zrefclever_load_dict_-
 verbose_bool ... 492, 543, 554, 564
 \g_zrefclever_loaded_dictionaries_-
 seq 491, 501, 537, 548
 _zrefclever_ltxlabel:n
 83, 3176, 3188, 3193, 3202, 3208, 3214
 \l_zrefclever_main_language_tl .
 29, 957, 964,
 970, 974, 978, 991, 1013, 1036, 1058

```

\__zrefclever_mathtools_showonlyrefs:n
..... 1592, 3265
\l__zrefclever_mathtools_-
showonlyrefs_bool 1590, 3257, 3264
\__zrefclever_name_default: ....
..... 2531, 2646
\l__zrefclever_name_format_-
fallback_tl ..... 1929, 2742,
2746, 2810, 2853, 2863, 2865, 2877
\l__zrefclever_name_format_tl ...
... 1929, 2728, 2729, 2732, 2733,
2743, 2744, 2817, 2822, 2823, 2829,
2834, 2845, 2859, 2860, 2871, 2883
\l__zrefclever_name_in_link_bool
..... 69,
72, 1929, 2286, 2587, 2901, 2917, 2918
\l__zrefclever_namefont_tl 1939,
2095, 2289, 2306, 2605, 2636, 2651
\l__zrefclever_nameinlink_str ...
..... 941, 946,
948, 950, 952, 2899, 2905, 2907, 2911
\l__zrefclever_namesep_tl .....
.. 1939, 2077, 2608, 2639, 2647, 2654
\l__zrefclever_next_is_same_bool
..... 55, 76, 1934,
2380, 2408, 2424, 2430, 2961, 2999
\l__zrefclever_next_maybe_range_-
bool .....
.. 55, 76, 1934, 2245, 2255, 2379,
2404, 2414, 2953, 2960, 2979, 2987
\l__zrefclever_noabbrev_first_-
bool ..... 881, 890, 2739
\l__zrefclever_nudge_comptosing_-
bool ... 1084, 1114, 1123, 1129, 2763
\l__zrefclever_nudge_enabled_-
bool ..... 1082, 1092, 1094,
1098, 1099, 1104, 1105, 2355, 2749
\l__zrefclever_nudge_gender_bool
..... 1086, 1115, 1125, 1130, 2777
\l__zrefclever_nudge_multitype_-
bool ... 1083, 1113, 1121, 1128, 2356
\l__zrefclever_nudge_singular_-
bool ..... 1085, 1141, 2751
\__zrefclever_orig_ltxlabel:n ...
.. 3178, 3187, 3192, 3201, 3207, 3213
\__zrefclever_page_format_aux: ..
..... 99, 103
\g__zrefclever_page_format_tl ...
..... 7, 98, 104, 107
\l__zrefclever_pairsep_tl .....
..... 1939, 2081, 2132, 2256
\__zrefclever_process_language_-
options: ..... 32, 34, 378, 1563
\__zrefclever_prop_put_non_-
empty:Nnn .... 24, 746, 1220, 1275
\__zrefclever_provide_dict_-
default_transl:nn 20, 569, 642, 659
\__zrefclever_provide_dict_type_-
transl:nn 20, 569, 623, 660, 679, 681
\__zrefclever_provide_dictionary:n
..... 11, 17-20, 45,
493, 565, 1026, 1037, 1045, 1060, 1562
\__zrefclever_provide_dictionary_-
verbose:n . 19, 561, 992, 1000, 1015
\l__zrefclever_range_beg_label_-
tl ..... 55, 1934, 1962,
2161, 2184, 2190, 2200, 2204, 2216,
2220, 2368, 2406, 2421, 2455, 2459,
2486, 2490, 2498, 2502, 2512, 2516
\l__zrefclever_range_count_int ..
..... 54,
1934, 1965, 2141, 2175, 2371, 2407,
2418, 2423, 2429, 2437, 2478, 2524
\l__zrefclever_range_same_count_-
int ..... 54,
1934, 1966, 2128, 2163, 2176, 2372,
2409, 2425, 2431, 2462, 2479, 2525
\l__zrefclever_rangesep_tl .....
..... 1939, 2079, 2222, 2257, 2518
\l__zrefclever_ref_decl_case_tl .
..... 14, 396, 401, 402, 406, 409,
413, 417, 420, 473, 476, 478, 1068,
1078, 2820, 2824, 2856, 2861, 2866
\__zrefclever_ref_default: .....
..... 2531, 2579, 2585, 2640, 2708
\l__zrefclever_ref_gender_tl ...
..... 15, 436, 442,
443, 447, 450, 455, 456, 480, 486,
487, 1087, 1151, 2778, 2787, 2794, 2802
\l__zrefclever_ref_language_tl ..
..... 14, 29, 30, 381, 400,
418, 440, 454, 477, 484, 956, 977,
990, 993, 998, 1001, 1007, 1012,
1016, 1026, 1035, 1038, 1043, 1046,
1052, 1057, 1061, 1562, 2781, 2796,
2804, 2827, 2869, 2875, 3044, 3050
\c__zrefclever_ref_options_font_-
seq ..... 12, 22, 265
\c__zrefclever_ref_options_-
genders_seq ..... 265
\c__zrefclever_ref_options_-
necessarily_not_type_specific_-
seq ..... 22, 265, 634, 1332, 1480
\c__zrefclever_ref_options_-
possibly_type_specific_seq ..
..... 22, 265, 651, 1501

```


`\l_zrefclever_ref_options_prop .`
 . [37](#), [39](#), [1295](#), [1305](#), [1306](#), [3011](#), [3068](#)
`\c_zrefclever_ref_options_-`
 reference_seq [265](#), [1297](#)
`\c_zrefclever_ref_options_type_-`
 names_seq [265](#), [665](#), [1524](#)
`\c_zrefclever_ref_options_-`
 typesetup_seq [265](#), [1344](#)
`\l_zrefclever_ref_property_tl .`
 [24](#), [752](#), [757](#),
 [759](#), [761](#), [767](#), [770](#), [786](#), [795](#), [1618](#),
 [1650](#), [1992](#), [2537](#), [2591](#), [2658](#), [2943](#)
`\l_zrefclever_ref_typeset_font_-`
 tl [1155](#), [1157](#), [1573](#)
`\l_zrefclever_reffont_in_tl` [1939](#),
 [2099](#), [2549](#), [2569](#), [2613](#), [2670](#), [2697](#)
`\l_zrefclever_reffont_out_tl .`
 [1939](#), [2097](#),
 [2546](#), [2566](#), [2610](#), [2630](#), [2667](#), [2694](#)
`\l_zrefclever_refpos_in_tl` [1939](#),
 [2093](#), [2558](#), [2573](#), [2618](#), [2686](#), [2702](#)
`\l_zrefclever_refpos_out_tl` [1939](#),
 [2089](#), [2561](#), [2575](#), [2631](#), [2689](#), [2704](#)
`\l_zrefclever_refpre_in_tl` [1939](#),
 [2091](#), [2555](#), [2570](#), [2614](#), [2682](#), [2698](#)
`\l_zrefclever_refpre_out_tl` [1939](#),
 [2087](#), [2547](#), [2567](#), [2611](#), [2668](#), [2695](#)
`\l_zrefclever_setup_type_tl .`
 [20](#), [260](#), [509](#), [573](#), [586](#),
 [587](#), [616](#), [641](#), [658](#), [672](#), [1328](#), [1356](#),
 [1364](#), [1377](#), [1427](#), [1428](#), [1457](#), [1467](#),
 [1487](#), [1508](#), [1517](#), [1531](#), [1541](#), [1547](#)
`\l_zrefclever_sort_decided_bool`
 [1611](#), [1740](#), [1754](#), [1764](#),
 [1768](#), [1780](#), [1790](#), [1805](#), [1820](#), [1844](#)
`_zrefclever_sort_default:nn .`
 [48](#), [1652](#), [1668](#)
`_zrefclever_sort_default_-`
 different_types:nn
 [26](#), [47](#), [52](#), [1706](#), [1857](#)
`_zrefclever_sort_default_same_-`
 type:nn [46](#), [50](#), [1704](#), [1727](#)
`_zrefclever_sort_labels: .`
 [47](#), [48](#), [53](#), [1571](#), [1615](#)
`_zrefclever_sort_page:nn .`
 [53](#), [1651](#), [1909](#)
`\l_zrefclever_sort_prior_a_int .`
 [1612](#),
 [1859](#), [1865](#), [1866](#), [1872](#), [1882](#), [1890](#)
`\l_zrefclever_sort_prior_b_int .`
 [1612](#),
 [1860](#), [1867](#), [1868](#), [1875](#), [1883](#), [1891](#)
`\l_zrefclever_tlastsep_tl .`
 [1939](#), [1974](#), [2349](#)

`\l_zrefclever_tlistsep_tl .`
 [1939](#), [1972](#), [2327](#)
`\l_zrefclever_tpairsep_tl .`
 [1939](#), [1970](#), [2343](#)
`\l_zrefclever_type_<type>-`
 options_prop [39](#)
`\l_zrefclever_type_count_int .`
 .. [54](#), [72](#), [1921](#), [1964](#), [2324](#), [2326](#),
 [2335](#), [2357](#), [2370](#), [2726](#), [2738](#), [2914](#)
`\l_zrefclever_type_first_label_-`
 tl [54](#), [69](#), [1923](#), [1960](#), [2119](#), [2237](#),
 [2246](#), [2250](#), [2278](#), [2294](#), [2298](#), [2366](#),
 [2396](#), [2584](#), [2590](#), [2597](#), [2601](#), [2616](#),
 [2657](#), [2675](#), [2679](#), [2684](#), [2700](#), [2714](#)
`\l_zrefclever_type_first_label_-`
 type_tl [54](#), [72](#), [1923](#), [1961](#),
 [2121](#), [2241](#), [2367](#), [2398](#), [2717](#), [2757](#),
 [2773](#), [2782](#), [2795](#), [2801](#), [2815](#), [2828](#),
 [2835](#), [2843](#), [2851](#), [2870](#), [2876](#), [2884](#)
`\l_zrefclever_type_name_gender_-`
 tl [1929](#), [2784](#), [2785](#), [2788](#), [2790](#), [2803](#)
`_zrefclever_type_name_setup: .`
 [11](#), [69](#), [2266](#), [2712](#)
`\l_zrefclever_type_name_tl .`
 [69](#), [71](#), [72](#),
 [1929](#), [2301](#), [2307](#), [2606](#), [2637](#), [2644](#),
 [2652](#), [2715](#), [2718](#), [2818](#), [2830](#), [2832](#),
 [2846](#), [2854](#), [2872](#), [2878](#), [2880](#), [2898](#)
`\l_zrefclever_typeset_compress_-`
 bool [850](#), [853](#), [2381](#)
`\l_zrefclever_typeset_labels_-`
 seq [54](#), [1918](#), [1956](#), [1980](#), [1982](#), [1988](#)
`\l_zrefclever_typeset_last_bool`
 [54](#), [1918](#),
 [1977](#), [1978](#), [1985](#), [2008](#), [2332](#), [2913](#)
`\l_zrefclever_typeset_name_bool`
 [801](#), [808](#), [813](#), [818](#), [2268](#), [2282](#)
`\l_zrefclever_typeset_queue_-`
 curr_tl [54](#), [67](#),
 [72](#), [1923](#), [1959](#), [2130](#), [2146](#), [2155](#),
 [2186](#), [2197](#), [2213](#), [2235](#), [2253](#), [2270](#),
 [2277](#), [2284](#), [2317](#), [2339](#), [2344](#), [2350](#),
 [2364](#), [2365](#), [2442](#), [2453](#), [2484](#), [2496](#),
 [2510](#), [2731](#), [2753](#), [2764](#), [2908](#), [2912](#)
`\l_zrefclever_typeset_queue_-`
 prev_tl . [54](#), [1923](#), [1958](#), [2328](#), [2363](#)
`\l_zrefclever_typeset_range_-`
 bool [859](#), [862](#), [1570](#), [2233](#)
`\l_zrefclever_typeset_ref_bool .`
 [800](#), [807](#), [812](#), [817](#), [2268](#), [2275](#)
`_zrefclever_typeset_refs: .`
 [54-56](#), [1574](#), [1954](#)
`_zrefclever_typeset_refs_last_-`
 of_type: [59](#), [67](#), [69](#), [71](#), [72](#), [2105](#), [2110](#)

| | |
|--|---|
| _zrefclever_typeset_refs_not_- | \l_zrefclever_zceref_labels_seq . |
| last_of_type: | 47 , 48 , 1560 , |
| 55 , 59 , 67 , 76 , 2107 , 2375 | 1588 , 1593 , 1597 , 1620 , 1623 , 1957 |
| \l_zrefclever_typeset_sort_bool | \l_zrefclever_zceref_note_tl ... |
| 826 , 829 , 1569 | 1171 , 1174 , 1576 , 1580 |
| \l_zrefclever_typesort_seq | \l_zrefclever_zceref_with_check_- |
| 26 , 52 , 835 , 840 , 841 , 847 , 1861 | bool 1178 , 1193 , 1566 , 1584 |
| \l_zrefclever_use_hyperref_bool | _zrefclever_zcsetup:n |
| 900 , 907 , | 38 , 1320 , 1321 , 3098 , |
| 912 , 917 , 927 , 933 , 2541 , 2662 , 2896 | 3103 , 3124 , 3130 , 3138 , 3158 , 3219 , |
| \l_zrefclever_warn_hyperref_- | 3250 , 3300 , 3320 , 3329 , 3342 , 3359 |
| bool 901 , 908 , 913 , 918 , 931 | \l_zrefclever_zrefcheck_- |
| _zrefclever_zceref:nnn 14 , 1555 , 1556 | available_bool |
| _zrefclever_zceref:nnnn 44 , 47 , 1556 | 1177 , 1188 , 1200 , 1565 , 1583 |